Eugene Asarin Patricia Bouyer (Eds.)

Formal Modeling and Analysis of Timed Systems

4th International Conference, FORMATS 2006
Paris, France, September 25-27, 2006
Proceedings

Springer

Volume Editors

Eugene Asarin
LIAFA, CNRS and Université Paris 7
Case 7014, 2 place Jussieu, 75251 Paris Cedex 05, France
E-mail: Eugene.Asarin@liafa.jussieu.fr

Patricia Bouyer
LSV, CNRS and ENS de Cachan
61, avenue du Président Wilson, 94235 Cachan Cedex, France
E-mail: bouyer@lsv.ens-cachan.fr

Library of Congress Control Number: 2006932409

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 3-540-45026-2 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-45026-9 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11867340 06/3142 5 4 3 2 1 0

Preface

This volume contains the proceedings of the 4th International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS 2006), held in Paris (France) on September 25-27, 2006. FORMATS aims to be a major annual event dedicated to the study of timed systems, uniting three independently started workshops: MTCS, RT-TOOLS, and TPTS. The first three FORMATS conferences were held in Marseille (2003), Grenoble (2004), and Uppsala (2005).

Timing aspects of systems have been treated independently in separate scientific disciplines, and there is a growing awareness of the difficult problems common to all of them, suggesting the interdisciplinary study of timed systems. The unifying theme underlying all these domains is that they concern systems whose behavior depends upon combinations of logical and temporal constraints, e.g., constraints on the distance between occurrences of events.

The aim of FORMATS is to promote the study of fundamental and practical aspects of timed systems, and to bring together researchers from different disciplines that share interests in modelling and analysis of timed systems. In this volume, there are articles on:

- *Foundations and Semantics:* contributions to the theoretical foundations of timed systems and timed formal languages as well as comparison between different models used by different communities (timed automata, timed Petri nets, timed MSCs, hybrid automata, timed process algebra, timed temporal logics, timed abstract state machines, as well as probabilistic models).
- *Methods and Tools:* techniques, algorithms, data structures, and software tools for analyzing timed systems and resolving temporal constraints (model-checking, simulation, robustness analysis, scheduling, etc).
- *Applications:* adaptation and specialization of timing technology to the modelling and analysis of applications in certain types of domains in which timing plays an important role (real-time software, hardware circuits, and network protocols).

We received 50 submissions out of which 22 were selected for publication. Each submission received 3 or 4 reviews. The conference program included four invited talks, by Alan Burns (University of York, UK), Thomas A. Henzinger (EPFL, Switzerland), Edward A. Lee (UC Berkeley, USA), and Alexander Rabinovitch (Tel Aviv University, Israel). We would like to thank all the program committee members and their sub-reviewers for their effort during the reviewing and selection process.

July 2006 Eugene Asarin and Patricia Bouyer

Organization

FORMATS 2006 was organized by LSV (CNRS and ENS de Cachan) and LIAFA (CNRS and University Paris 7). We are grateful to all persons who helped us in the organization of the conference.

Program Committee

Parosh Aziz Abdulla (Uppsala University, Sweden)
Luca de Alfaro (UC Santa Cruz, USA)
Rajeev Alur (University of Pennsylvania, USA)
Eugene Asarin, co-chair (LIAFA, University Paris 7 and CNRS, France)
Danièle Beauquier (LACL, University Paris 12, France)
Gerd Behrmann (Aalborg University, Denmark)
Bernard Berthomieu (LAAS, CNRS, France)
Patricia Bouyer, co-chair (LSV, CNRS and ENS de Cachan, France)
Marius Bozga (Verimag, CNRS, France)
Deepak D'Souza (IISc, Bangalore, India)
Holger Hermanns (Saarland University, Germany)
Salvatore La Torre (University of Salerno, Italy)
Gerald Lüttgen (University of York, UK)
Nicolas Markey (LSV, CNRS and ENS Cachan, France)
Peter Niebert (LIF, University of Provence, France)
Joël Ouaknine (Oxford University, UK)
Paul Pettersson (Uppsala University, Sweden)
Jean-François Raskin (Université Libre de Bruxelles, Belgium)
Lothar Thiele (ETH Zurich, Switzerland)
Walter Vogler (University of Augsburg, Germany)
Sergio Yovine (Verimag, CNRS, France)

Steering Committee

Rajeev Alur (University of Pennsylvania, USA)
Flavio Corradini (University of Camerino, Italy)
Kim G. Larsen (Aalborg University, Denmark)
Oded Maler (Verimag, CNRS, France)
Walter Vogler (University of Augsburg, Germany)
Wang Yi (Uppsala University, Sweden)

Referees

We would like to thank the following sub-reviewers for their help in evaluating the papers submitted to the conference:

Noomene Ben Henda
Marco Bernardo
Elmar Bihler
Benedikt Bollig
Victor Braberman
Jeremy Bradley
Manuela Bujorianu
Fabrice Chevalier
Olivier Constant
Alexandre David
Matteo Dell'Amico
Martin De Wulf
Cătălin Dima
Alexandre Donze
Laurent Doyen
Marie Duflot
Jean Fanchon
Emmanuel Fleury
Martin Fränzle
Gilles Geeraerts
Gregor Goessler
Madhu Gopinathan
Susanne Graf
Serge Grigorieff
Olga Grinchtein
Daniel Grosse
Stefan Haar
Anders Hessel
Jochen Hoenicke
Hardi Hungar
John Håkansson
Joost-Pieter Katoen
Pavel Krčál
Daniel Kröning
Ruggero Lanotte
Luciano Lavagno
Didier Lime
Birgitta Lindström
Michael Mislove
Raj Mohan M.
Leonid Mokrushin
Laurent Mounier
Jan Tobias Mühlberg
Anca Muscholl
Margherita Napoli
Dejan Nickovic
Vincent Nicomette
Damian Niwinski
Gethin Norman
Alfredo Olivero
Richard Paige
Paritosh Pandya
Mimmo Parente
Florent Peres
Antoine Petit
Pavithra Prabhakar
Reza Pulungan
Alexander Rabinovich
Jacob Illum Rasmussen
Arend Rensink
Pierre-Alain Reynier
Christophe Rippert
Mark Schaefer
Fernando Schapachnik
Roberto Segala
Mihaela Sighireanu
Sunil Easaw Simon
Anatol Slissenko
Tayssir Touili
Éric Tournier
Helen Treharne
Enrico Tronci
Robert Valette
Pavel Vasilyev
Serghei Verlan
François Vernadat
Jim Woodcock
James Worrell

Sponsors

We would like to thank our sponsors:

AutoMathA project

LIAFA

Table of Contents

Invited Talks

Timed Alternating-Time Temporal Logic 1
Thomas A. Henzinger, Vinayak S. Prabhu

Concurrent Semantics Without the Notions of State or State Transitions 18
Edward A. Lee

Decidability and Expressive Power of Real Time Logics 32
Alexander Rabinovich

Contributed Papers

Extended Directed Search for Probabilistic Timed Reachability 33
Husain Aljazzar, Stefan Leue

Intersection of Regular Signal-Event (Timed) Languages 52
Béatrice Bérard, Paul Gastin, Antoine Petit

Refinements and Abstractions of Signal-Event (Timed) Languages 67
Béatrice Bérard, Paul Gastin, Antoine Petit

Bridging the Gap Between Timed Automata and Bounded Time Petri Nets 82
Bernard Berthomieu, Florent Peres, François Vernadat

Matching Scenarios with Timing Constraints 98
Prakash Chandrasekaran, Madhavan Mukund

Verification of the Generic Architecture of a Memory Circuit Using Parametric Timed Automata 113
Remy Chevallier, Emmanuelle Encrenaz-Tiphène, Laurent Fribourg, Weiwen Xu

Model Checking Timed Automata with Priorities Using DBM Subtraction 128
Alexandre David, John Håkansson, Kim G. Larsen, Paul Pettersson

Symbolic Robustness Analysis of Timed Automata 143
Conrado Daws, Piotr Kordy

Coping with the Parallelism of BitTorrent: Conversion of PEPA to ODEs in Dealing with State Space Explosion 156
Adam Duguid

Temporal Logic Verification Using Simulation 171
Georgios E. Fainekos, Antoine Girard, George J. Pappas

Undecidable Problems About Timed Automata 187
Olivier Finkel

On Timed Simulation Relations for Hybrid Systems and Compositionality 200
Goran Frehse

Integrating Discrete- and Continuous-Time Metric Temporal Logics Through Sampling 215
Carlo A. Furia, Matteo Rossi

On the Computational Power of Timed Differentiable Petri Nets 230
Serge Haddad, Laura Recalde, Manuel Silva

Model-Checking Timed ATL for Durational Concurrent Game Structures 245
François Laroussinie, Nicolas Markey, Ghassan Oreiby

A Dose of Timed Logic, in Guarded Measure 260
Kamal Lodaya, Paritosh K. Pandya

From MITL to Timed Automata 274
Oded Maler, Dejan Nickovic, Amir Pnueli

Adding Invariants to Event Zone Automata 290
Peter Niebert, Hongyang Qu

Static Analysis for State-Space Reduction of Polygonal Hybrid Systems 306
Gordon Pace, Gerardo Schneider

On the Expressiveness of MTL with Past Operators 322
Pavithra Prabhakar, Deepak D'Souza

Simulator for Real-Time Abstract State Machines 337
Pavel Vasilyev

A Characterization of Meaningful Schedulers for Continuous-Time Markov Decision Processes 352
Nicolás Wolovick, Sven Johr

Author Index 369

Timed Alternating-Time Temporal Logic*

Thomas A. Henzinger[1] and Vinayak S. Prabhu[2]

[1] Department of Computer and Communication Sciences, EPFL
tah@epfl.ch
[2] Department of Electrical Engineering and Computer Sciences, UC Berkeley
vinayak@eecs.berkeley.edu

Abstract. We add freeze quantifiers to the game logic ATL in order to specify real-time objectives for games played on timed structures. We define the semantics of the resulting logic TATL by restricting the players to physically meaningful strategies, which do not prevent time from diverging. We show that TATL can be model checked over timed automaton games. We also specify timed optimization problems for physically meaningful strategies, and we show that for timed automaton games, the optimal answers can be approximated to within any degree of precision.

1 Introduction

Timed games are a formal model for the synthesis of real-time systems [22,20]. While much research effort has been directed at algorithms for solving timed games [14,9,7,16,15,8,11], we find it useful to revisit the topic for two reasons. First, we wish to study a perfectly symmetric setup of the model, where all players (whether they represent a plant, a controller, a scheduler, etc.) are given equally powerful options for updating the state of the game, advancing time, or blocking time. Second, we wish to restrict all players to physically meaningful strategies, which do not allow a player to prevent time from diverging in order to achieve an objective. This restriction is often ensured by syntactic conditions on the cycles of timed automaton games [7,16,8,21] or by semantic conditions that discretize time; we find such conditions unsatisfactory and unnecessary: unsatisfactory, because they rule out perfectly meaningful strategies that suggest an arbitrary but finite number of transitions in a finite interval of time; unnecessary, because timed automaton games can be solved without such conditions.

We do not present a new model for timed games, but review the model of [13], which is symmetric for all players and handles the divergence of dense time without constraining the players. We consider the two-player case. Previous work on the existence of controllers [14,9,20,11] has in general required that time divergence be *ensured* by the controller — an unfair view in settings where the player for the environment can also block time. In our model, both players may block time, however, for a player to win for an objective, she must not be *responsible* for preventing time from diverging. To achieve this, we distinguish

* This research was supported in part by the NSF grants CCR-0208875, CCR-0225610, and CCR-0234690.

between *objectives* and *winning conditions.* An objective for a player is a set Φ of desired outcomes of the game. The winning condition WC maps the objective to another set of outcomes so that the player wins for $\mathsf{WC}(\Phi)$ using any strategy if and only if she wins for the original objective Φ using a physically meaningful strategy.

Let us be more precise. A timed game proceeds in an infinite sequence of turns. At each turn, both players propose a move: each move specifies an amount of time that the player is willing to let pass without action, possibly followed by an action that causes a discontinuous jump to a different state. The move with the *shorter* proposed time delay determines the next state of the game (if both players propose the same delays, then one of the corresponding actions is chosen nondeterministically). An outcome of the game is an infinite trajectory of continuous state segments (during which time passes) and discontinuous jumps. Let $\mathsf{Timediv}$ denote the outcomes for which time diverges (the other trajectories are often called "zeno" behaviors). Let $\mathsf{Blameless}_i$ denote the outcomes in which player $i \in \{1, 2\}$ proposes the shorter delay only finitely often. Clearly, player i is not responsible if time converges for an outcome in $\mathsf{Blameless}_i$. We therefore use the winning condition [13]

$$\mathsf{WC}_i(\Phi) \;=\; (\mathsf{Timediv} \cap \Phi) \;\cup\; (\mathsf{Blameless}_i \setminus \mathsf{Timediv}).$$

Informally, this condition states that if an outcome is time divergent, then it is a valid outcome, and hence must satisfy the objective Φ; and if it is not time divergent, then player i must not be responsible for the zeno behaviour. The winning conditions for both players are perfectly symmetric: since $\mathsf{WC}_1(\Phi) \cap \mathsf{WC}_2(\neg\Phi) = \emptyset$, at most one player can win.

In [6], several alternating-time temporal logics were introduced to specify properties of game structures, including the CTL-like logic ATL, and the CTL*-like logic ATL*. For example, the ATL formula $\langle\langle i\rangle\rangle\Diamond p$ is true at a state s iff player i can force the game from s into a state that satisfies the proposition p. We interpret these logics over *timed* game structures, and enrich them by adding *freeze* quantifiers [5] for specifying timing constraints. The resulting logics are called TATL and TATL*. The new logic TATL subsumes both the untimed game logic ATL, and the timed non-game logic TCTL [3]. For example, the TATL formula $\langle\langle i\rangle\rangle\Diamond_{\leq d}\, p$ is true at a state s iff player i can force the game from s into a p state in at most d time units. A version of TATL has recently been studied on durational concurrent structures in [19] .

The model checking of these logics requires the solution of timed games. Timed game structures are infinite-state. In order to consider algorithmic solutions, we restrict our attention to timed game structures that are generated by a finite syntax borrowed from timed automata [4]. By restricting the strategies of TATL games to physically meaningful strategies using WC, we obtain TATL* games. However, solving TATL* games is undecidable, because TATL* subsumes the linear-time logic TPTL [5], whose dense-time satisfiability problem is undecidable. We nontheless establish the decidability of TATL model checking, by carefully analyzing the fragment of TATL* we obtain through the WC translation.

We show that TATL model checking over timed automaton games is complete for EXPTIME; that is, no harder than the solution of timed automaton games with reachability objectives.

In timed games, as in optimal control, it is natural to study not only the decision problem if a player can force the game into a target state within d time units, but also the corresponding optimization problem: determine the *minimal* time d so that a player can force the game into a target state. Again we insist on the use of physically meaningful strategies. We show that for timed automaton games, the optimal answer can be computed to within any desired degree of precision. The exact optimization problem is still open; only special cases have been solved, such as the special case where every cycle of the timed automaton ensures syntactically that a positive amount of time passes [7], and the special case where the game is restricted to a finite number of moves [2]. The general case for *weighted* timed automaton games is known to be undecidable [10]. Average reward games in the framework of [13] are considered in [1], but with the time move durations restricted to either 0 or 1.

2 Timed Games

2.1 Timed Game Structures

We borrow our formalism from [13]. A *timed game structure* is a tuple $\mathcal{G} = \langle S, \Sigma, \sigma, A_1, A_2, \Gamma_1, \Gamma_2, \delta \rangle$ with the following components:

- S is a set of states.
- Σ is a finite set of propositions.
- $\sigma : S \mapsto 2^{\Sigma}$ is the observation map, which assigns to every state the set of propositions that are true in that state.
- A_1 and A_2 are two disjoint sets of actions for players 1 and 2, respectively. We assume that $\perp \notin A_i$, and write $A_i^{\perp}$ for $A_i \cup \{\perp\}$. The set of *moves* for player i is $M_i = \mathbb{R}_{\geq 0} \times A_i^{\perp}$. Intuitively, a move $\langle \Delta, a_i \rangle$ by player i indicates a waiting period of Δ time units followed by a discrete transition labeled with action a_i.
- $\Gamma_i : S \mapsto 2^{M_i} \setminus \emptyset$ are two move assignments. At every state s, the set $\Gamma_i(s)$ contains the moves that are available to player i. We require that $\langle 0, \perp \rangle \in \Gamma_i(s)$ for all states $s \in S$ and $i \in \{1, 2\}$. Intuitively, $\langle 0, \perp \rangle$ is a time-blocking stutter move.
- $\delta : S \times (M_1 \cup M_2) \mapsto S$ is the transition function. We require that for all time delays $\Delta, \Delta' \in \mathbb{R}_{\geq 0}$ with $\Delta' \leq \Delta$, and all actions $a_i \in A_i^{\perp}$, we have (1) $\langle \Delta, a_i \rangle \in \Gamma_i(s)$ iff both $\langle \Delta', \perp \rangle \in \Gamma_i(s)$ and $\langle \Delta - \Delta', a_i \rangle \in \Gamma_i(\delta(s, \langle \Delta', \perp \rangle))$; and (2) if $\delta(s, \langle \Delta', \perp \rangle) = s'$ and $\delta(s', \langle \Delta - \Delta', a_i \rangle) = s''$, then $\delta(s, \langle \Delta, a_i \rangle) = s''$.

The game proceeds as follows. If the current state of the game is s, then both players simultaneously propose moves $\langle \Delta_1, a_1 \rangle \in \Gamma_1(s)$ and $\langle \Delta_2, a_2 \rangle \in \Gamma_2(s)$. The move with the shorter duration "wins" in determining the next state of the game. If both moves have the same duration, then one of the two moves is chosen nondeterministically. Formally, we define the *joint destination function* $\delta_{\mathsf{jd}} : S \times M_1 \times M_2 \mapsto 2^S$ by

$$\delta_{\mathsf{jd}}(s, \langle \Delta_1, a_1\rangle, \langle \Delta_2, a_2\rangle) = \begin{cases} \{\delta(s, \langle \Delta_1, a_1\rangle)\} & \text{if } \Delta_1 < \Delta_2; \\ \{\delta(s, \langle \Delta_2, a_2\rangle)\} & \text{if } \Delta_2 < \Delta_1; \\ \{\delta(s, \langle \Delta_1, a_1\rangle), \delta(s, \langle \Delta_2, a_2\rangle)\} & \text{if } \Delta_1 = \Delta_2. \end{cases}$$

The time elapsed when the moves $m_1 = \langle \Delta_1, a_1\rangle$ and $m_2 = \langle \Delta_2, a_2\rangle$ are proposed is given by $\mathsf{delay}(m_1, m_2) = \min(\Delta_1, \Delta_2)$. The boolean predicate $\mathsf{blame}_i(s, m_1, m_2, s')$ indicates whether player i is "responsible" for the state change from s to s' when the moves m_1 and m_2 are proposed. Denoting the opponent of player $i \in \{1, 2\}$ by $\sim i = 3 - i$, we define

$$\mathsf{blame}_i(s, \langle \Delta_1, a_1\rangle, \langle \Delta_2, a_2\rangle, s') \;=\; \big(\Delta_i \le \Delta_{\sim i} \;\wedge\; \delta(s, \langle \Delta_i, a_i\rangle) = s'\big).$$

A *run* of the timed game structure $\mathcal{G}$ is an infinite sequence $r = s_0, \langle m_1^0, m_2^0\rangle, s_1, \langle m_1^1, m_2^1\rangle, \ldots$ such that $s_k \in S$ and $m_i^k \in \Gamma_i(s_k)$ and $s_{k+1} \in \delta_{\mathsf{jd}}(s_k, m_1^k, m_2^k)$ for all $k \ge 0$ and $i \in 1, 2$. For $k \ge 0$, let $\mathsf{time}(r, k)$ denote the "time" at position k of the run, namely, $\mathsf{time}(r, k) = \sum_{j=0}^{k-1} \mathsf{delay}(m_1^j, m_2^j)$ (we let $\mathsf{time}(r, 0) = 0$). By $r[k]$ we denote the $(k+1)$-th state s_k of r. The run prefix $r[0..k]$ is the finite prefix of the run r that ends in the state s_k; we write $\mathsf{last}(r[0..k])$ for the ending state s_k of the run prefix. Let Runs be the set of all runs of $\mathcal{G}$, and let $\mathsf{FinRuns}$ be the set of run prefixes.

A *strategy* π_i for player $i \in \{1, 2\}$ is a function $\pi_i : \mathsf{FinRuns} \mapsto M_i$ that assigns to every run prefix $r[0..k]$ a move to be proposed by player i at the state $\mathsf{last}(r[0..k])$ if the history of the game is $r[0..k]$. We require that $\pi_i(r[0..k]) \in \Gamma_i(\mathsf{last}(r[0..k]))$ for every run prefix $r[0..k]$, so that strategies propose only available moves. The results of this paper are equally valid if strategies do not depend on past moves chosen by the players, but only on the past sequence of states and time delays [13]. For $i \in \{1, 2\}$, let Π_i be the set of player-i strategies. Given two strategies $\pi_1 \in \Pi_1$ and $\pi_2 \in \Pi_2$, the set of possible *outcomes* of the game starting from a state $s \in S$ is denoted $\mathsf{Outcomes}(s, \pi_1, \pi_2)$: it contains all runs $r = s_0, \langle m_1^0, m_2^0\rangle, s_1, \langle m_1^1, m_2^1\rangle, \ldots$ such that $s_0 = s$ and for all $k \ge 0$ and $i \in \{1, 2\}$, we have $\pi_i(r[0..k]) = m_i^k$.

2.2 Timed Winning Conditions

An *objective* for the timed game structure $\mathcal{G}$ is a set $\Phi \subseteq \mathsf{Runs}$ of runs. The objective Φ is *untimed ω-regular* if there exists an ω-regular set $\Psi \subseteq (2^{\Sigma})^{\omega}$ of infinite sequences of sets of propositions such that a run $r = s_0, \langle m_1^0, m_2^0\rangle, s_1, \langle m_1^1, m_2^1\rangle, \ldots$ is in Φ iff the projection $\sigma(r) = \sigma(s_0), \sigma(s_1), \sigma(s_2), \ldots$ is in Ψ.

To win an objective Φ, a player must ensure that the possible outcomes of the game satisfy the *winning condition* $\mathsf{WC}(\Phi)$, a different subset of Runs. We distinguish between objectives and winning conditions, because players must win their objectives using only physically meaningful strategies; for example, a player should not satisfy the objective of staying in a safe set by blocking time forever. Formally, player $i \in \{1,2\}$ *wins* for the objective $\varPhi$ at a state $s \in S$ if there is a player-i strategy π_i such that for all opposing strategies $\pi_{\sim i}$, we have $\mathsf{Outcomes}(s, \pi_1, \pi_2) \subseteq \mathsf{WC}_i(\varPhi)$. In this case, we say that player i has the *winning strategy* π_i. The winning condition is formally defined as

$$\mathsf{WC}_i(\varPhi) \;=\; (\mathsf{Timediv} \cap \varPhi) \;\cup\; (\mathsf{Blameless}_i \setminus \mathsf{Timediv}),$$

which uses the following two sets of runs:

- $\mathsf{Timediv} \subseteq \mathsf{Runs}$ is the set of all time-divergent runs. A run r is *time-divergent* if $\lim_{k\to\infty} \mathsf{time}(r,k) = \infty$.
- $\mathsf{Blameless}_i \subseteq \mathsf{Runs}$ is the set of runs in which player i is responsible only for finitely many transitions. A run $s_0, \langle m_1^0, m_2^0\rangle, s_1, \langle m_1^1, m_2^1\rangle, \ldots$ belongs to the set $\mathsf{Blameless}_i$, for $i = \{1,2\}$, if there exists a $k \geq 0$ such that for all $j \geq k$, we have $\neg\, \mathsf{blame}_i(s_j, m_1^j, m_2^j, s_{j+1})$.

Thus a run r belongs to $\mathsf{WC}_i(\varPhi)$ if and only if the following conditions hold:

- if $r \in \mathsf{Timediv}$, then $r \in \varPhi$;
- if $r \notin \mathsf{Timediv}$, then $r \in \mathsf{Blameless}_i$.

Informally, if time diverges, then the outcome of the game is valid and the objective must be met, and if time does not diverge, then only the opponent should be responsible for preventing time from diverging.

A state $s \in S$ in a timed game structure $\mathcal{G}$ is *well-formed* if both players can win at s for the trivial objective Runs. The timed game structure $\mathcal{G}$ is *well-formed* if all states of $\mathcal{G}$ are well-formed. Structures that are not well-formed are not physically meaningful. We retrict out attention to well-formed timed game structures.

A strategy π_i for player $i \in \{1,2\}$ is *reasonable* if for all opposing strategies $\pi_{\sim i}$, all states $s \in S$, and all runs $r \in \mathsf{Outcomes}(s, \pi_1, \pi_2)$, either $r \in \mathsf{Timediv}$ or $r \in \mathsf{Blameless}_i$. Thus, no what matter what the opponent does, a reasonable player-i strategy should not be responsible for blocking time. Strategies that are not reasonable are not physically meaningful. A timed game structure is thus well-formed iff both players have reasonable strategies. We now show that we can restrict our attention to games which allow only reasonable strategies.

Proposition 1. *Let $s \in S$ be a state of a well-formed time game structure $\mathcal{G}$, and let $\varPhi \subseteq \mathsf{Runs}$ be an objective.*

1. *Player 1 wins for the objective $\varPhi$ at the state s iff there is a* reasonable *player-1 winning strategy π_1^*, that is, for all player-2 strategies π_2, we have $\mathsf{Outcomes}(s, \pi_1^*, \pi_2) \subseteq \mathsf{WC}(\varPhi)$.*
2. *Player 1 does not win for the objective $\varPhi$ at s using only reasonable strategies iff there is a reasonable player-2 spoiling strategy π_2^*. Formally, for every reasonable player-1 strategy π_1^*, there is a player-2 strategy π_2 such that $\mathsf{Outcomes}(s, \pi_1^*, \pi_2) \not\subseteq \mathsf{WC}(\varPhi)$ iff there is a* reasonable *player-2 strategy π_2^* such that $\mathsf{Outcomes}(s, \pi_1^*, \pi_2^*) \not\subseteq \mathsf{WC}(\varPhi)$.*

The symmetric claims with players 1 and 2 interchanged also hold.

Proof. (1) Let π_1 be the winning strategy for player 1 for objective Φ at state s. Let π_1 be not reasonable. Then, by definition, there exists an opposing strategy π_2 such that for some run $r \in \mathsf{Outcomes}(s, \pi_1, \pi_2)$, we have both $r \notin \mathsf{Timediv}$ and $r \notin \mathsf{Blameless}_1$. This contradicts the fact that π_1 was a winning strategy.
(2) Let π_1^* be any player-1 reasonable strategy. Player 1 loses for the objective Φ from state s, thus there exists a player 2 spoiling strategy π_2 such that $\mathsf{Outcomes}(s, \pi_1^*, \pi_2) \not\subseteq \mathsf{WC}(\Phi)$. This requires that for some run $r \in \mathsf{Outcomes}(s, \pi_1^*, \pi_2)$, we have either 1) $(r \in \mathsf{Timediv}) \wedge (r \notin \Phi)$ or 2) $(r \notin \mathsf{Timediv}) \wedge (r \notin \mathsf{Blameless}_1)$. We cannot have the second case, for π_1^* is a reasonable strategy, thus, the first case must hold. By definition, for every state s' in a well-formed time game structure, there exists a player-2 reasonable strategy $\pi_2^{s'}$. Now, let π_2^* be such that its acts like π_2 on the particular run r, and is like π_2^s otherwise, that is $\pi_2^*(r_f) = \pi_2(r_f)$, for all run prefixes r_f of r, and $\pi_2^*(r_f) = \pi_2^s(r_f)$ otherwise. The strategy π_2^* is reasonable, as for all strategies π_1', and for every run $r' \in \mathsf{Outcomes}(s, \pi_1', \pi_2^*)$, we have $(r' \in \mathsf{Timediv}) \vee (r' \in \mathsf{Blameless}_2)$. Since π_2^* acts like π_2 on the particular run r, it is also spoiling for the player-1 strategy π_1^*. □

Corollary 1. *For $i = \{1, 2\}$, let $\mathsf{Win}_i(\Phi)$ be the states of a well-formed timed game structure $\mathcal{G}$ at which player i can win for the objective Φ. Let $\mathsf{Win}_i^*(\Phi)$ be the states at which player i can win for the objective Φ when both players are restricted to use reasonable strategies. Then, $\mathsf{Win}_i(\Phi) = \mathsf{Win}_i^*(\Phi)$.*

Note that if π_1^* and π_2^* are player-1 and player-2 reasonable strategies, then for every state s and every run $r \in \mathsf{Outcomes}(s, \pi_1, \pi_2)$, we have r to be non-zeno. Thus, if we restrict our attention to plays in which both players use only reasonable strategies, then for any objective Φ, we have that player i wins for the winning condition $\mathsf{WC}(\Phi)$ if and only if he wins for the winning condition Φ. We can hence talk *semantically* about games restricted to reasonable player strategies in well-formed timed game structures, without differentiating between objectives and winning conditions. From a *computational* perspective, we allow all strategies, taking care to distinguish between objectives and winning conditions. Proposition 1 indicates both approaches to be equivalent.

2.3 Timed Automaton Games

Timed automata [4] suggest a finite syntax for specifying infinite-state timed game structures. A *timed automaton game* is a tuple $\mathcal{T} = \langle L, \Sigma, \sigma, C, A_1, A_2, E, \gamma \rangle$ with the following components:

- L is a finite set of locations.
- Σ is a finite set of propositions.
- $\sigma : L \mapsto 2^{\Sigma}$ assigns to every location a set of propositions.
- C is a finite set of clocks. We assume that $z \in C$ for the unresettable clock z, which is used to measure the time elapsed since the start of the game.
- A_1 and A_2 are two disjoint sets of actions for players 1 and 2, respectively.

– $E \subseteq L \times (A_1 \cup A_2) \times \mathsf{Constr}(C) \times L \times 2^{C \setminus \{z\}}$ is the edge relation, where the set $\mathsf{Constr}(C)$ of *clock constraints* is generated by the grammar

$$\theta ::= x \leq d \mid d \leq x \mid \neg\theta \mid \theta_1 \wedge \theta_2$$

for clock variables $x \in C$ and nonnegative integer constants d. For an edge $e = \langle l, a_i, \theta, l', \lambda \rangle$, the clock constraint θ acts as a guard on the clock values which specifies when the edge e can be taken, and by taking the edge e, the clocks in the set $\lambda \subseteq C \setminus \{z\}$ are reset to 0. We require that for all edges $\langle l, a_i, \theta', l', \lambda' \rangle, \langle l, a_i, \theta'', l'', \lambda'' \rangle \in E$ with $l' \neq l''$, the conjunction $\theta' \wedge \theta''$ is unsatisfiable. This requirement ensures that a state and a move together uniquely determine a successor state.

– $\gamma : L \mapsto \mathsf{Constr}(C)$ is a function that assigns to every location an invariant for both players. All clocks increase uniformly at the same rate. When at location l, each player i must propose a move out of l before the invariant $\gamma(l)$ expires. Thus, the game can stay at a location only as long as the invariant is satisfied by the clock values.

A *clock valuation* is a function $\kappa : C \mapsto \mathbb{R}_{\geq 0}$ that maps every clock to a nonnegative real. The set of all clock valuations for C is denoted by $K(C)$. Given a clock valuation $\kappa \in K(C)$ and a time delay $\Delta \in \mathbb{R}_{\geq 0}$, we write $\kappa + \Delta$ for the clock valuation in $K(C)$ defined by $(\kappa + \Delta)(x) = \kappa(x) + \Delta$ for all clocks $x \in C$. For a subset $\lambda \subseteq C$ of the clocks, we write $\kappa[\lambda := 0]$ for the clock valuation in $K(C)$ defined by $(\kappa[\lambda := 0])(x) = 0$ if $x \in \lambda$, and $(\kappa[\lambda := 0])(x) = \kappa(x)$ if $x \notin \lambda$. A clock valuation $\kappa \in K(C)$ *satisfies* the clock constraint $\theta \in \mathsf{Constr}(C)$, written $\kappa \models \theta$, if the condition θ holds when all clocks in C take on the values specified by κ.

A *state* $s = \langle l, \kappa \rangle$ of the timed automaton game $\mathcal{T}$ is a location $l \in L$ together with a clock valuation $\kappa \in K(C)$ such that the invariant at the location is satisfied, that is, $\kappa \models \gamma(l)$. Let S be the set of all states of $\mathcal{T}$. In a state, each player i proposes a time delay allowed by the invariant map γ, together either with the action $\perp$, or with an action $a_i \in A_i$ such that an edge labeled a_i is enabled after the proposed time delay. We require that for $i \in \{1, 2\}$ and for all states $s = \langle l, \kappa \rangle$, if $\kappa \models \gamma(l)$, either $\kappa + \Delta \models \gamma(l)$ for all $\Delta \in \mathbb{R}_{\geq 0}$, or there exist a time delay $\Delta \in \mathbb{R}_{\geq 0}$ and an edge $\langle l, a_i, \theta, l', \lambda \rangle \in E$ such that (1) $a_i \in A_i$ and (2) $\kappa + \Delta \models \theta$ and for all $0 \leq \Delta' \leq \Delta$, we have $\kappa + \Delta' \models \gamma(l)$, and (3) $(\kappa + \Delta)[\lambda := 0] \models \gamma(l')$. This requirement is necessary (but not sufficient) for well-formedness of the game.

The timed automaton game $\mathcal{T}$ defines the following timed game structure $[\![\mathcal{T}]\!] = \langle S, \Sigma, \sigma^*, A_1, A_2, \Gamma_1, \Gamma_2, \delta \rangle$:

– S is defined above.
– $\sigma^*(\langle l, \kappa \rangle) = \sigma(l)$.
– For $i \in \{1, 2\}$, the set $\Gamma_i(\langle l, \kappa \rangle)$ contains the following elements:
  1. $\langle \Delta, \perp \rangle$ if for all $0 \leq \Delta' \leq \Delta$, we have $\kappa + \Delta' \models \gamma(l)$.
  2. $\langle \Delta, a_i \rangle$ if for all $0 \leq \Delta' \leq \Delta$, we have $\kappa + \Delta' \models \gamma(l)$, $a_i \in A_i$ and there exists an edge $\langle l, a_i, \theta, l', \lambda \rangle \in E$ such that $\kappa + \Delta \models \theta$.

– $\delta(\langle l, \kappa\rangle, \langle \Delta, \bot\rangle) = \langle l, \kappa + \Delta\rangle$, and $\delta(\langle l, \kappa\rangle, \langle \Delta, a_i\rangle) = \langle l', (\kappa + \Delta)[\lambda := 0]\rangle$ for the unique edge $\langle l, a_i, \theta, l', \lambda\rangle \in E$ with $\kappa + \Delta \models \theta$.

The timed game structure $[\![\mathcal{T}]\!]$ is not necessarily well-formed, because it may contain cycles along which time cannot diverge. We will see below how we can check well-formedness for timed automaton games.

2.4 Clock Regions

Timed automaton games can be solved using a region construction from the theory of timed automata [4]. For a real $t \geq 0$, let $\mathsf{frac}(t) = t - \lfloor t \rfloor$ denote the fractional part of t. Given a timed automaton game $\mathcal{T}$, for each clock $x \in C$, let c_x denote the largest integer constant that appears in any clock constaint involving x in $\mathcal{T}$ (let $c_x = 0$ if there is no clock constraint invloving x). Two clock valuations $\kappa_1, \kappa_2 \in K(C)$ are *clock-region equivalent*, denoted $\kappa_1 \cong \kappa_2$, if the following three conditions hold:

1. For all $x \in C$, either $\lfloor \kappa_1(x) \rfloor = \lfloor \kappa_2(x) \rfloor$, or both $\lfloor \kappa_1(x) \rfloor > c_x$, $\lfloor \kappa_2(x) \rfloor > c_x$.
2. For all $x, y \in C$ with $\kappa_1(x) \leq c_x$ and $\kappa_1(y) \leq c_y$, we have $\mathsf{frac}(\kappa_1(x)) \leq \mathsf{frac}(\kappa_1(y))$ iff $\mathsf{frac}(\kappa_2(x)) \leq \mathsf{frac}(\kappa_2(y))$.
3. For all $x \in C$ with $\kappa_1(x) \leq c_x$, we have $\mathsf{frac}(\kappa_1(x)) = 0$ iff $\mathsf{frac}(\kappa_2(x)) = 0$.

Two states $\langle l_1, \kappa_1\rangle, \langle l_2, \kappa_2\rangle \in S$ are *clock-region equivalent*, denoted $\langle l_1, \kappa_1\rangle \cong \langle l_2, \kappa_2\rangle$, iff $l_1 = l_2$ and $\kappa_1 \cong \kappa_2$. It is not difficult to see that $\cong$ is an equivalence relation on S. A *clock region* is an equivalence class with respect to $\cong$. There are finitely many clock regions; more precisely, the number of clock regions is bounded by $|L| \cdot \prod_{x \in C}(c_x + 1) \cdot |C|! \cdot 2^{|C|}$. For a state $s \in S$, we write $[s] \subseteq S$ for the clock region containing s.

Timed automaton games can be solved for untimed ω-regular objectives by considering the finite quotient game structure obtained from the clock-region equivalence [13]. There, also a variation of timed games is considered, where each move is either a time delay or an action (rather than a pair of both). The results of this paper carry over also to that model (which is strictly weaker, in that a player may be able to achieve fewer objectives).

3 TATL

The alternating-time temporal logics ATL and ATL* were introduced in [6] for specifying properties of untimed game structures. These logics are natural specification languages for multi-component systems, where properties need to be guarenteed by subsets of the components irrespective of the behavior of the other components. Each component represents a player in the game, and sets of players may form teams. We restrict our attention here to the two-player case (e.g., system vs. environment; or plant vs. controller), but all results can be extended to the multi-player case. For example, letting the system be player 1, and the environment player 2, the ATL formula $\langle\!\langle 1 \rangle\!\rangle \Box p$ specifies the property that the system will always remain in a safe set of p states, no matter what the environment does.

For timed systems, we need the players to use only reasonable strategies when achieving their objectives. We show that this requirement can be encoded within ATL* (but not within ATL). We also permit timing constraints within objectives. For example, the timed formula $\langle\langle 1\rangle\rangle\Diamond_{\leq d}\, p$ says that player 1 can reach a target set of p states within d time units, no matter what player 2 does (and again, player 1 must use only reasonable strategies to attain this goal). The resulting timed logics are called TATL and TATL*. While the model-checking problem of TATL* is undecidable over timed automaton games, we show that it is decidable for the fragment of TATL* that is obtained by adding to TATL the restriction to reasonable strategies.

3.1 Syntax and Semantics

Consider a fixed timed game structure $\mathcal{G} = \langle S, \Sigma, \sigma, A_1, A_2, \Gamma_1, \Gamma_2, \delta\rangle$. The temporal logic TATL (Timed Alternating-Time Temporal Logic) is interpreted over the states of $\mathcal{G}$. We use the syntax of *freeze quantification* [5] for specifying timing constraints within the logic. The freeze quantifier "$x\cdot$" binds the value of the clock variable x in a formula $\varphi(x)$ to the current time $t \in \mathbb{R}_{\geq 0}$; that is, the constraint $x\cdot\varphi(x)$ holds at time t iff $\varphi(t)$ does. For example, the property that "every p state is followed by a q state within d time units" can be written as: $\forall\Box x\cdot(p \rightarrow \Diamond y\cdot(q \wedge y \leq x + d))$. This formula says that "in every state with time x, if p holds, then there is a later state with time y such that both q and $y \leq x + d$ hold." Formally, given a set D of clock variables, a TATL formula is one of the following:

- $\textsc{true} \mid p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2$, where $p \in \Sigma$ is a proposition, and φ_1, φ_2 are TATL formulae.
- $x + d_1 \leq y + d_2 \mid x\cdot\varphi$, where $x, y \in D$ are clock variables and d_1, d_2 are nonnegative integer constants, and φ is a TATL formula. We refer to the clocks in D as *formula clocks.*
- $\langle\langle\mathfrak{P}\rangle\rangle\Box\varphi \mid \langle\langle\mathfrak{P}\rangle\rangle\varphi_1\mathcal{U}\varphi_2$, where $\mathfrak{P} \subseteq \{1, 2\}$ is a set of players, and $\varphi, \varphi_1, \varphi_2$ are TATL formulae.

We omit the next operator of ATL, which has no meaning in timed systems. The freeze quantifier $x\cdot\varphi$ binds all free occurrences of the formula clock variable x in the formula φ. A TATL formula is *closed* if it contains no free occurrences of formula clock variables. Without loss of generality, we assume that for every quantified formula $x\cdot\varphi$, if $y\cdot\varphi'$ is a subformula of φ, then x and y are different; that is, there is no nested reuse of formula clocks. When interpreted over the states of a timed automaton game $\mathcal{T}$, a TATL formula may also contain free (unquantified) occurrences of clock variables from $\mathcal{T}$.

There are four possible sets of players (so-called *teams*), which may collaborate to achieve a common goal: we write $\langle\langle\,\rangle\rangle$ for $\langle\langle\emptyset\rangle\rangle$; we write $\langle\langle i\rangle\rangle$ for $\langle\langle\{i\}\rangle\rangle$ with $i \in \{1, 2\}$; and we write $\langle\langle 1, 2\rangle\rangle$ for $\langle\langle\{1, 2\}\rangle\rangle$. Roughly speaking, a state s satisfies the TATL formula $\langle\langle i\rangle\rangle\varphi$ iff player i can win the game at s for an objective derived from φ. The state s satisfies the formula $\langle\langle\,\rangle\rangle\varphi$ (resp., $\langle\langle 1, 2\rangle\rangle\varphi$) iff every run (resp., some run) from s is contained in the objective derived from φ. Thus, the team $\emptyset$ corresponds to both players playing adversially, and the team $\{1,2\}$ corresponds to both players collaborating to achieve a goal. We therefore write $\forall$ short for $\langle\!\langle\, \rangle\!\rangle$, and $\exists$ short for $\langle\!\langle 1,2 \rangle\!\rangle$, as in ATL.

We assign the responsibilities for time divergence to teams as follows: let $\mathsf{Blameless}_{\emptyset} = \mathsf{Runs}$, let $\mathsf{Blameless}_{\{1,2\}} = \emptyset$, and let $\mathsf{Blameless}_{\{i\}} = \mathsf{Blameless}_i$ for $i \in \{1,2\}$. A strategy $\pi_{\mathfrak{P}}$ for the team $\mathfrak{P}$ consists of a strategy for each player in $\mathfrak{P}$. We denote the "opposing" team by $\sim\!\mathfrak{P} = \{1,2\} \setminus \mathfrak{P}$. Given a state $s \in S$, a team-$\mathfrak{P}$ strategy $\pi_{\mathfrak{P}}$, and a team-$\sim\!\mathfrak{P}$ strategy $\pi_{\sim\mathfrak{P}}$, we define $\mathsf{Outcomes}(s, \pi_{\mathfrak{P}} \cup \pi_{\sim\mathfrak{P}}) = \mathsf{Outcomes}(s, \pi_1, \pi_2)$ for the player-1 strategy π_1 and the player-2 strategy π_2 in the set $\pi_{\mathfrak{P}} \cup \pi_{\sim\mathfrak{P}}$ of strategies. Given a team-$\mathfrak{P}$ strategy $\pi_{\mathfrak{P}}$, we define the set of possible outcomes from state s by $\mathsf{Outcomes}(s, \pi_{\mathfrak{P}}) = \cup_{\pi_{\sim\mathfrak{P}}} \mathsf{Outcomes}(s, \pi_{\mathfrak{P}} \cup \pi_{\sim\mathfrak{P}})$, where the union is taken over all team-$\sim\!\mathfrak{P}$ strategies $\pi_{\sim\mathfrak{P}}$.

To define the semantics of TATL, we need to distinguish between *physical time* and *game time*. We allow moves with zero time delay, thus a physical time $t \in \mathbb{R}_{\geq 0}$ may correspond to several linearly ordered states, to which we assign the game times $\langle t, 0\rangle, \langle t, 1\rangle, \langle t, 2\rangle, \ldots$ For a run $r \in \mathsf{Runs}$, we define the set of game times as

$$\mathsf{GameTimes}(r) \;=\; \begin{array}{l}\{\langle t,k\rangle \in \mathbb{R}_{\geq 0} \times \mathbb{N} \mid 0 \leq k < |\{j \geq 0 \mid \mathsf{time}(r,j) = t\}|\} \;\cup \\ \{\langle t, 0\rangle \mid \mathsf{time}(r,j) \geq t \text{ for some } j \geq 0\}.\end{array}$$

The state of the run r at a game time $\langle t,k\rangle \in \mathsf{GameTimes}(r)$ is defined as

$$\mathsf{state}(r, \langle t,k\rangle) \;=\; \begin{cases} r[j+k] & \text{if } \mathsf{time}(r,j) = t \text{ and for all } j' < j, \mathsf{time}(r,j') < t; \\ \delta(r[j], \langle t - \mathsf{time}(r,j), \bot\rangle) & \text{if } \mathsf{time}(r,j) < t < \mathsf{time}(r, j+1). \end{cases}$$

Note that if r is a run of the timed game structure $\mathcal{G}$, and $\mathsf{time}(r,j) < t < \mathsf{time}(r,j+1)$, then $\delta(r[j], \langle t - \mathsf{time}(r,j), \bot\rangle)$ is a state in S, namely, the state that results from $r[j]$ by letting time $t - \mathsf{time}(r,j)$ pass. We say that the run r *visits* a proposition $p \in \Sigma$ if there is a $\tau \in \mathsf{GameTimes}(r)$ such that $p \in \sigma(\mathsf{state}(r,\tau))$. We order the game times of a run lexicographically: for all $\langle t,k\rangle, \langle t',k'\rangle \in \mathsf{GameTimes}(r)$, we have $\langle t,k\rangle < \langle t',k'\rangle$ iff either $t < t'$, or $t = t'$ and $k < k'$. For two game times τ and τ', we write $\tau \leq \tau'$ iff either $\tau = \tau'$ or $\tau < \tau'$.

An *environment* $\mathcal{E} : D \mapsto \mathbb{R}_{\geq 0}$ maps every formula clock in D to a nonnegative real. Let $\mathcal{E}[x := t]$ be the environment such that $(\mathcal{E}[x := t])(y) = \mathcal{E}(y)$ if $y \neq x$, and $(\mathcal{E}[x := t])(y) = t$ if $y = x$. For a state $s \in S$, a time $t \in \mathbb{R}_{\geq 0}$, an environment $\mathcal{E}$, and a TATL formula φ, the satisfaction relation $(s,t,\mathcal{E}) \models_{\mathrm{td}} \varphi$ is defined inductively as follows (the subscript td indicates that players may win in only a physically meaningful way):

- $(s,t,\mathcal{E}) \models_{\mathrm{td}} \text{TRUE}$; $\quad (s,t,\mathcal{E}) \models_{\mathrm{td}} p$, for a proposition p, iff $p \in \sigma(s)$.
- $(s,t,\mathcal{E}) \models_{\mathrm{td}} \neg\varphi$ iff $(s,t,\mathcal{E}) \not\models_{\mathrm{td}} \varphi$.
- $(s,t,\mathcal{E}) \models_{\mathrm{td}} \varphi_1 \wedge \varphi_2$ iff $(s,t,\mathcal{E}) \models_{\mathrm{td}} \varphi_1$ and $(s,t,\mathcal{E}) \models_{\mathrm{td}} \varphi_2$.
- $(s,t,\mathcal{E}) \models_{\mathrm{td}} x + d_1 \leq y + d_2$ iff $\mathcal{E}(x) + d_1 \leq \mathcal{E}(y) + d_2$.
- $(s,t,\mathcal{E}) \models_{\mathrm{td}} x \cdot \varphi$ iff $(s,t,\mathcal{E}[x := t]) \models_{\mathrm{td}} \varphi$.

– $(s, t, \mathcal{E}) \models_{\text{td}} \langle\langle \mathfrak{P} \rangle\rangle \Box \varphi$ iff there is a team-$\mathfrak{P}$ strategy $\pi_{\mathfrak{P}}$ such that for all runs $r \in \mathsf{Outcomes}(s, \pi_{\mathfrak{P}})$, the following conditions hold:
  If $r \in \mathsf{Timediv}$, then for all $\langle u, k \rangle \in \mathsf{GameTimes}(r)$, we have $(\mathsf{state}(r, \langle u, k \rangle), t + u, \mathcal{E}) \models_{\text{td}} \varphi$. If $r \notin \mathsf{Timediv}$, then $r \in \mathsf{Blameless}_{\mathfrak{P}}$.
– $(s, t, \mathcal{E}) \models_{\text{td}} \langle\langle \mathfrak{P} \rangle\rangle \varphi_1 \mathcal{U} \varphi_2$ iff there is a team-$\mathfrak{P}$ strategy $\pi_{\mathfrak{P}}$ such that for all runs $r \in \mathsf{Outcomes}(s, \pi_{\mathfrak{P}})$, the following conditions hold:
  If $r \in \mathsf{Timediv}$, then there is a $\langle u, k \rangle \in \mathsf{GameTimes}(r)$ such that $(\mathsf{state}(r, \langle u, k \rangle), t + u, \mathcal{E}) \models_{\text{td}} \varphi_2$, and for all $\langle u', k' \rangle \in \mathsf{GameTimes}(r)$ with $\langle u', k' \rangle < \langle u, k \rangle$, we have $(\mathsf{state}(r, \langle u', k' \rangle), t + u', \mathcal{E}) \models_{\text{td}} \varphi_1$. If $r \notin \mathsf{Timediv}$, then $r \in \mathsf{Blameless}_{\mathfrak{P}}$.

Note that for an $\exists$ formula to hold, we require time divergence (as $\mathsf{Blameless}_{\{1,2\}} = \emptyset$). Also note that for a closed formula, the value of the environment is irrelevant in the satisfaction relation. A state s of the timed game structure $\mathcal{G}$ *satisfies* a closed formula φ of TATL, denoted $s \models_{\text{td}} \varphi$, if $(s, 0, \mathcal{E}) \models_{\text{td}} \varphi$ for any environment $\mathcal{E}$.

We use the following abbreviations. We write $\langle\langle \mathfrak{P} \rangle\rangle \varphi_1 \mathcal{U}_{\sim d} \varphi_2$ for $x \cdot \langle\langle \mathfrak{P} \rangle\rangle \varphi_1 \mathcal{U}\, y \cdot (\varphi_2 \wedge y \sim x + d)$, where $\sim$ is one of $<$, $\leq$, $=$, $\geq$, or $>$. Interval constraints can also be encoded in TATL; for example, $\langle\langle \mathfrak{P} \rangle\rangle \varphi_1 \mathcal{U}_{(d_1, d_2]} \varphi_2$ stands for $x \cdot \langle\langle \mathfrak{P} \rangle\rangle \varphi_1 \mathcal{U}\, y \cdot (\varphi_2 \wedge y > x + d_1 \wedge y \leq x + d_2)$. We write $\Diamond \varphi$ for $\textsc{true}\, \mathcal{U} \varphi$ as usual, and therefore $\langle\langle \mathfrak{P} \rangle\rangle \Diamond_{\sim d} \varphi$ stands for $x \cdot \langle\langle \mathfrak{P} \rangle\rangle \Diamond y \cdot (\varphi \wedge y \sim x + d)$.

3.2 TATL*

TATL is a fragment of the more expressive logic called TATL*. There are two types of formulae in TATL*: *state formulae*, whose satisfaction is related to a particular state, and *path formulae*, whose satisfaction is related to a specific run. Formally, a TATL* state formula is one of the following:

(S1) $\textsc{true}$ or p for propositions $p \in \Sigma$.
(S2) $\neg \varphi$ or $\varphi_1 \wedge \varphi_2$ for TATL* state formulae φ, φ_1, and φ_2.
(S3) $x + d_1 \leq y + d_2$ for clocks $x, y \in D$ and nonnegative integer constants d_1, d_2.
(S4) $\langle\langle \mathfrak{P} \rangle\rangle \psi$ for $\mathfrak{P} \subseteq \{1, 2\}$ and TATL* path formulae ψ.

A TATL* path formula is one of the following:

(P1) A TATL* state formula.
(P2) $\neg \psi$ or $\psi_1 \wedge \psi_2$ for TATL* path formulae ψ, ψ_1, and ψ_2.
(P3) $x \cdot \psi$ for formula clocks $x \in D$ and TATL* path formulae ψ.
(P4) $\psi_1 \mathcal{U} \psi_2$ for TATL* path formulae ψ_1, ψ_2.

The logic TATL* consists of the formulae generated by the rules S1–S4. As in TATL, we assume that there is no nested reuse of formula clocks. Additional temporal operators are defined as usual; for example, $\Diamond \varphi$ stands for $\textsc{true}\, \mathcal{U} \varphi$, and $\Box \varphi$ stands for $\neg \Diamond \neg \varphi$. The logic TATL can be viewed as a fragment of TATL* consisting of formuale in which every $\mathcal{U}$ operator is immediately preceeded by a $\langle\langle \mathfrak{P} \rangle\rangle$ operator, possibly with an intermittent negation symbol [6].

The semantics of TATL* formulae are defined with respect to an environment $\mathcal{E} : D \mapsto \mathbb{R}_{\geq 0}$. We write $(s,t,\mathcal{E}) \models \varphi$ to indicate that the state s of the timed game structure $\mathcal{G}$ satisfies the TATL* state formula φ at time $t \in \mathbb{R}_{\geq 0}$; and $(r,\tau,t,\mathcal{E}) \models \psi$ to indicate that the suffix of the run r of $\mathcal{G}$ which starts from game time $\tau \in \mathsf{GameTimes}(r)$ satisfies the TATL* path formula ψ, provided the time at the initial state of r is t. Unlike TATL, we allow all strategies for both players (including unreasonable strategies), because we will see that the use of reasonable strategies can be enforced within TATL* by certain path formulae. Formally, the satisfaction relation $\models$ is defined inductively as follows. For state formulae φ,

- $(s,t,\mathcal{E}) \models \text{TRUE}$; $(s,t,\mathcal{E}) \models p$, for a proposition p, iff $p \in \sigma(s)$.
- $(s,t,\mathcal{E}) \models \neg\varphi$ iff $(s,t,\mathcal{E}) \not\models \varphi$.
- $(s,t,\mathcal{E}) \models \varphi_1 \wedge \varphi_2$ iff $(s,t,\mathcal{E}) \models \varphi_1$ and $(s,t,\mathcal{E}) \models \varphi_2$.
- $(s,t,\mathcal{E}) \models x + d_1 \leq y + d_2$ iff $\mathcal{E}(x) + d_1 \leq \mathcal{E}(y) + d_2$.
- $(s,t,\mathcal{E}) \models \langle\langle \mathfrak{P} \rangle\rangle \psi$ iff there is a team-$\mathfrak{P}$ strategy $\pi_{\mathfrak{P}}$ such that for all runs $r \in \mathsf{Outcomes}(s, \pi_{\mathfrak{P}})$, we have $(r, \langle 0,0 \rangle, t, \mathcal{E}) \models \psi$.

For path formulae ψ,

- $(r, \langle u,k \rangle, t, \mathcal{E}) \models \varphi$, for a state formula φ, iff $(\mathsf{state}(r, \langle u,k \rangle), t+u, \mathcal{E}) \models \varphi$.
- $(r,\tau,t,\mathcal{E}) \models \neg\psi$ iff $(r,\tau,t,\mathcal{E}) \not\models \psi$.
- $(r,\tau,t,\mathcal{E}) \models \psi_1 \wedge \psi_2$ iff $(r,\tau,t,\mathcal{E}) \models \psi_1$ and $(r,\tau,t,\mathcal{E}) \models \psi_2$.
- $(r, \langle u,k \rangle, t, \mathcal{E}) \models x{\cdot}\psi$ iff $(r, \langle u,k \rangle, t, \mathcal{E}[x := t+u]) \models \psi$.
- $(r,\tau,t,\mathcal{E}) \models \psi_1 \mathcal{U} \psi_2$ iff there is a $\tau' \in \mathsf{GameTimes}(r)$ such that $\tau \leq \tau'$ and $(r,\tau',t,\mathcal{E}) \models \psi_2$, and for all $\tau'' \in \mathsf{GameTimes}(r)$ with $\tau \leq \tau'' < \tau'$, we have $(r,\tau'',t,\mathcal{E}) \models \psi_1$.

A state s of the timed game structure $\mathcal{G}$ *satisfies* a closed formula φ of TATL*, denoted $s \models \varphi$, if $(s,0,\mathcal{E}) \models \varphi$ for any environment $\mathcal{E}$.

3.3 Model Checking TATL

We restrict our attention to timed automaton games. Given a closed TATL (resp. TATL*) formula φ, a timed automaton game $\mathcal{T}$, and a state s of the timed game structure $[\![\mathcal{T}]\!]$, the *model-checking problem* is to determine whether $s \models_{\mathrm{td}} \varphi$ (resp., $s \models \varphi$). The alternating-time logic TATL* subsumes the linear-time logic TPTL [5]. Thus the model-checking problem for TATL* is undecidable. On the other hand, we now solve the model-checking problem for TATL by reducing it to a special kind of TATL* problem, which turns out to be decidable.

Given a TATL formula φ over the set D of formula clocks, and a timed automaton game $\mathcal{T}$, we look at the timed automaton game $\mathcal{T}_\varphi$ with the set $C_\varphi = C \uplus D$ of clocks (we assume $C \cap D = \emptyset$). Let c_x be the largest constant to which the formula variable x is compared in φ. We pick an invariant $\gamma(l)$ in $\mathcal{T}$ and modify it to $\gamma(l)' = \gamma(l) \wedge (x \leq c_x \vee x \geq c_x)$ in $\mathcal{T}_\varphi$ for every formula clock $x \in D$ (this is to inject the proper constants in the region equivalence relation). Thus, $\mathcal{T}_\varphi$ acts exactly like $\mathcal{T}$ except that it contains some extra clocks which are never used. As in [13], we represent the sets $\mathsf{Timediv}$

and $\mathsf{Blameless}_i$ using ω-regular conditions. We look at the enlarged automaton game structure $\widehat{[\![\mathfrak{T}_\varphi]\!]}$ with the state space $\widehat{S} = S_\varphi \times \{T, F\}^3$, and an augmented transition relation $\widehat{\delta}_{\mathsf{jd}} : \widehat{S} \times M_1 \times M_2 \mapsto 2^{\widehat{S}}$. In an augmented state $\langle s, \mathit{tick}, \mathit{bl}_1, \mathit{bl}_2\rangle \in \widehat{S}$, the component $s \in S_\varphi$ is a state of the original game structure $[\![\mathfrak{T}_\varphi]\!]$, tick is true if the global clock z has crossed an integer boundary in the last transition, and bl_i is true if player i is to blame for the last transition. Formally, $\langle\langle l', \kappa'\rangle, \mathit{tick}', \mathit{bl}_1', \mathit{bl}_2'\rangle \in \widehat{\delta}_{\mathsf{jd}}(\langle\langle l, \kappa\rangle, \mathit{tick}, \mathit{bl}_1, \mathit{bl}_2\rangle, m_1, m_2)$ iff (1) $\langle l', \kappa'\rangle \in \delta_{\mathsf{jd}}(\langle l, \kappa\rangle, m_1, m_2)$; (2) $\mathit{tick}' = \text{TRUE}$ if $\kappa'(z) - \kappa(z) \geq 1$, and FALSE otherwise; and (3) $\mathit{bl}_i' = \mathsf{blame}_i(\langle l, \kappa\rangle, m_1, m_2, \langle l', \kappa'\rangle)$. It can be seen that a run is in $\mathsf{Timediv}$ iff tick is true infinitely often, and that the set $\mathsf{Blameless}_i$ corresponds to runs along which bl_i is true only finitely often. We extend the clock equivalence relation to these expanded states: $\langle\langle l, \kappa\rangle, \mathit{tick}, \mathit{bl}_1, \mathit{bl}_2\rangle \cong \langle\langle l', \kappa'\rangle, \mathit{tick}', \mathit{bl}_1', \mathit{bl}_2'\rangle$ iff $l = l', \mathit{tick} = \mathit{tick}', \mathit{bl}_1 = \mathit{bl}_1', \mathit{bl}_2 = \mathit{bl}_2'$ and $\kappa \cong \kappa'$. Finally, we extend bl to teams: $\mathit{bl}_\emptyset = \text{FALSE}, \mathit{bl}_{\{1,2\}} = \text{TRUE}, \mathit{bl}_{\{i\}} = \mathit{bl}_i$.

We will use the algorithm of [13] which computes winning sets for timed automaton games with untimed ω-regular objectives. The algorithm uses the *controllable predecessor operator*, $\mathsf{CPre}_1 : 2^{\widehat{S}} \mapsto 2^{\widehat{S}}$ in its fixpoint computation, defined formally by $\mathsf{CPre}_1(\widehat{X}) \equiv \{\widehat{s} \mid \exists m_1 \in \Gamma_1(\widehat{s})\ \forall m_2 \in \Gamma_2(\widehat{s})(\widehat{\delta}_{\mathsf{jd}}(\widehat{s}, m_1, m_2) \subseteq \widehat{X}\})$. Intuitively, $\widehat{s} \in \mathsf{CPre}_1(\widehat{X})$ iff player 1 can force the augmented game into $\widehat{X}$ from $\widehat{s}$ in one move. The CPre_1 operator is invariant over states of a region, that is, for $\widehat{X}$ a union of regions, and $\widehat{s} \cong \widehat{s}'$, we have $\widehat{s} \in \mathsf{CPre}_1(\widehat{X})$ iff $\widehat{s}' \in \mathsf{CPre}_1(\widehat{X})$. This invariance follows from the fact that if player 1 can force the game into a region $\widehat{R}$ from $\widehat{s}$, then he can do so from any other state $\widehat{s}' \cong \widehat{s}$. The region invariance of CPre_1 allows the us to work on the region game graph. So long as we work with state sets that correspond to unions of regions, we get winning sets that are also unions of regions. We show that we can maintain this invariant when model checking TATL.

We first consider the subset of TATL in which formulae are clock variable free. Using the encoding for time divergence and blame predicates, we can embed the notion of reasonable winning strategies into TATL* formulae.

Lemma 1. *A state s in a timed game structure $[\![\mathfrak{T}_\varphi]\!]$ satisfies a formula clock variable free* TATL *formula φ in a meaningful way, denoted $s \models_{\mathrm{td}} \varphi$, iff the state $\widehat{s} = \langle s, \text{FALSE}, \text{FALSE}, \text{FALSE}\rangle$ in the expanded game structure $\widehat{[\![\mathfrak{T}_\varphi]\!]}$ satisfies the* TATL* *formula $\mathsf{atlstar}(\varphi)$, that is, iff $\widehat{s} \models \mathsf{atlstar}(\varphi)$ where $\mathsf{atlstar}$ is a partial mapping from* TATL *to* TATL*, *defined inductively as follows:*

$$\mathsf{atlstar}(\text{TRUE}) = \text{TRUE}; \qquad \mathsf{atlstar}(p) = p$$

$$\mathsf{atlstar}(\neg\varphi) = \neg\, \mathsf{atlstar}(\varphi); \qquad \mathsf{atlstar}(\varphi_1 \wedge \varphi_2) = \mathsf{atlstar}(\varphi_1) \wedge \mathsf{atlstar}(\varphi_2)$$

$$\mathsf{atlstar}(\langle\!\langle \mathfrak{P}\rangle\!\rangle \Box\varphi) = \langle\!\langle \mathfrak{P}\rangle\!\rangle \left((\Box\Diamond\, \mathit{tick} \rightarrow \Box\, \mathsf{atlstar}(\varphi)) \wedge (\Diamond\Box\neg\, \mathit{tick} \rightarrow \Diamond\Box\neg\, \mathit{bl}_{\mathfrak{P}})\right)$$

$$\mathsf{atlstar}(\langle\!\langle \mathfrak{P}\rangle\!\rangle \varphi_1\, \mathcal{U} \varphi_2) = \langle\!\langle \mathfrak{P}\rangle\!\rangle \begin{pmatrix} (\Box\Diamond\, \mathit{tick} \rightarrow \mathsf{atlstar}(\varphi_1)\, \mathcal{U}\, \mathsf{atlstar}(\varphi_2)) \wedge \\ (\Diamond\Box\neg\, \mathit{tick} \rightarrow \Diamond\Box\neg\, \mathit{bl}_{\mathfrak{P}}) \end{pmatrix}$$

Now, for φ a clock variable free TATL formula, $\mathsf{atlstar}(\varphi)$ is actually an ATL* formula. Thus, the untimed ω-regular model checking algorithm of [13] can be used to (recursively) model check $\mathsf{atlstar}(\varphi)$. As we are working in the continuous domain, we need to ensure that for an until formula $\langle\langle\mathfrak{P}\rangle\rangle\varphi_1\,\mathcal{U}\varphi_2$, team $\mathfrak{P}$ does not "jump" over a time at which $\neg(\varphi_1 \vee \varphi_2)$ holds. This can be handled by introducing another player in the opposing team $\sim\!\mathfrak{P}$, the *observer*, who can only take pure time moves. The observer entails the opposing team to observe *all* time points. The observer is necessary only when $\mathfrak{P} = \{1, 2\}$. We omit the details.

A naive extension of the above approach to full TATL does not immediately work, for then we get TATL* formulae which are not in ATL* (model checking for TATL* is undecidable). We do the following: for each formula clock constraint $x + d_1 \leq y + d_2$ appearing in the formula φ, let there be a new proposition p_α for $\alpha = x + d_1 \leq y + d_2$. We denote the set of all such formula clock constraint propositions by Λ. A state $\langle l, \kappa\rangle$ in the timed automaton game $\mathcal{T}_\varphi$ satisfies p_α for $\alpha = x + d_1 \leq y + d_2$ iff $\kappa(x) + d_1 \leq \kappa(y) + d_2$. The propositions p_α are invariant over regions, maintaining the region-invariance of sets in the fixpoint algorithm of [13], thus allowing us to work over the region game graph.

Lemma 2. *For a* TATL *formula φ, let φ^Λ be obtained from φ by replacing all formula variable constraints $x + d_1 \leq y + d_2$ with equivalent propositions $p_\alpha \in \Lambda$. Let $[\![\mathcal{T}_\varphi]\!]^\Lambda$ denote the timed game structure $[\![\mathcal{T}_\varphi]\!]$ together with the propositions from Λ. Then,*

1. *We have $s \models_{\mathrm{td}} \varphi$ for a state s in the timed game structure $[\![\mathcal{T}_\varphi]\!]$ iff the state $s \models_{\mathrm{td}} \varphi^\Lambda$ in $[\![\mathcal{T}_\varphi]\!]^\Lambda$.*
2. *Let $\varphi^\Lambda = w{\cdot}\psi^\Lambda$. Then, in the structure $[\![\mathcal{T}_\varphi]\!]^\Lambda$ the state $s \models_{\mathrm{td}} \varphi^\Lambda$ iff $s[w := 0] \models_{\mathrm{td}} \psi^\Lambda$.*
3. *Let $\varphi = \langle\langle\mathfrak{P}\rangle\rangle\Box p \mid \langle\langle\mathfrak{P}\rangle\rangle p_1\,\mathcal{U} p_2$, where p, p_1, p_2 are propositions that are invariant over states of regions in $\mathcal{T}_\varphi$. Then for $s \cong s'$ in $\mathcal{T}_\varphi$, we have $s \models_{\mathrm{td}} \varphi$ iff $s' \models_{\mathrm{td}} \varphi$.*

Lemmas 1 and 2 together with the EXPTIME algorithm for timed automaton games with untimed ω-regular region objectives give us a recursive model-checking algorithm for TATL.

Theorem 1. *The model-checking problem for* TATL *(over timed automaton games) is* EXPTIME*-complete.*

EXPTIME-hardness follows from the EXPTIME-hardness of alternating reachability on timed automata [17].

Model checking of TATL allows us to check the well-formedness of a timed automaton game $\mathcal{T}$: a state s of the timed game structure $[\![\mathcal{T}]\!]$ is well-formed iff $s \models_{\mathrm{td}} (\langle\langle 1\rangle\rangle\Box\text{TRUE}) \wedge (\langle\langle 2\rangle\rangle\Box\text{TRUE})$ This well-formedness check is the generalization to the game setting of the non-zenoness check for timed automata, which computes the states s such that $s \models_{\mathrm{td}} \exists\Box\text{TRUE}$ [18]. If not all states of $[\![\mathcal{T}]\!]$ are well-formed, then the location invariants of $\mathcal{T}$ can be strengthened to characterize well-formed states (note that the set of well-formed states consists of a union of regions).

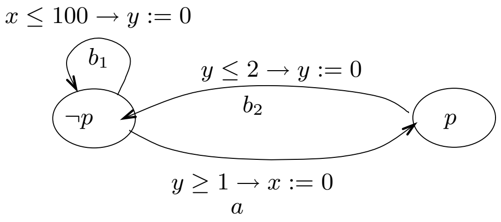

$x \leq 100 \rightarrow y := 0$
b_1
$y \leq 2 \rightarrow y := 0$
b_2
$\neg p$
p
$y \geq 1 \rightarrow x := 0$
a

Fig. 1. A timed automaton game

4 The Minimum-Time Problem

The *minimum-time problem* is to determine the minimal time in which a player can force the game into a set of target states, using only reasonable strategies. This game problem is the generalisation of the non-game version of [12]. Formally, given a player $i \in \{1,2\}$, a timed game structure $\mathcal{G}$, a target proposition $p \in \Sigma$, and a run r of $\mathcal{G}$, let

$$T^i_{\text{visit}}(r,p) = \begin{cases} \infty \ \text{ if } r \notin \mathsf{Timediv} \text{ and } r \notin \mathsf{Blameless}_i; \\ \infty \ \text{ if } r \in \mathsf{Timediv} \text{ and } r \text{ does not visit } p; \\ 0 \ \text{ if } r \notin \mathsf{Timediv} \text{ and } r \in \mathsf{Blameless}_i; \\ \inf\{t \in \mathbb{R}_{\geq 0} \mid p \in \sigma(\mathsf{state}(r, \langle t,k \rangle)) \text{ for some } k\} \ \text{ otherwise.} \end{cases}$$

Then, the *minimal time* for player 1 to force the game from a start state $s \in S$ to a p state is defined as

$$T^1_{\min}(s,p) = \inf_{\pi_1 \in \Pi_1} \sup_{\pi_2 \in \Pi_2} \{T^1_{\text{visit}}(r,p) \mid r \in \mathsf{Outcomes}(s,\pi_1,\pi_2)\}.$$

The minimal time $T^2_{\min}(s,p)$ for player 2 is defined symmetrically. As in Corollary 1, the minimal time remains the same when both players are restricted to use reasonable strategies.

Proposition 2. *Given a state s, a proposition p of a well-formed timed game structure $\mathcal{G}$, and $i \in \{1,2\}$, let $T^{i,*}_{\min}(s,p)$ be the minimal time for player i to force the game from s to a p state with both players restricted to use only reasonable strategies. Then, $T^{i,*}_{\min}(s,p) = T^i_{\min}(s,p)$.*

As an example consider the timed automaton game in Figure 1 with initial state $s_0 = \langle \neg p, (x=0, y=0) \rangle$. The action a belongs to player 1 and $b_j, j \in \{1,2\}$ to player 2. Not all runs in the game graph are non-zeno, in particular player 2 can keep take b_1 and keep player 1 from reaching p from s_0. However, it can be easily seen that physically, player 1 will be able to have p satisfied by time 101.

To solve the minimum-time problem, we consider a well-formed timed automaton game $\mathcal{T} = \langle L, \Sigma, \sigma, C, A_1, A_2, E, \gamma \rangle$.

Lemma 3. *For a state s and a target proposition $p \in \Sigma$ of a well-formed timed game automaton $\mathcal{T}$,*

1. *Let $s \models_{\mathrm{td}} \langle\langle i \rangle\rangle \Diamond p$. Then there exists $d < \infty$ such that $s \models_{\mathrm{td}} \langle\langle i \rangle\rangle \Diamond_{\leq d} p$.*
2. *Let $s \models_{\mathrm{td}} \langle\langle i \rangle\rangle \Diamond_{\leq d} p$. Then the minimal time for player i to reach p from state s is less than or equal to d, that is, $T^i_{\min}(s, p) \leq d$.*
3. *Let $s \not\models_{\mathrm{td}} \langle\langle i \rangle\rangle \Diamond_{\leq d} p$. Then the minimal time for player i to reach p from state s is not less than d, that is, $T^i_{\min}(s, p) \geq d$.*

Proof. We prove the first claim for $i = 1$. We have that $s \models \langle\langle 1 \rangle\rangle \Diamond p$, thus there is a player-1 strategy π_1 such that for all opposing strategies π_2 of player 2, and for all runs $r \in \mathsf{Outcomes}(s, \pi_1, \pi_2)$ we have that, 1) if time diverges in run r then r contains a state satisfying p, and 2) if time does not diverge in r, then player 1 is blameless. Suppose that for all $d > 0$ we have $s \not\models_{\mathrm{td}} \langle\langle 1 \rangle\rangle \Diamond_{\leq d} p$. We have that player 1 cannot win for his objective of $\Diamond_{\leq d} p$, in particular, π_1 is not a winning strategy for this new objective. Hence, there is a player-2 strategy π_2^d such that for some run $r_d \in \mathsf{Outcomes}(s, \pi_1, \pi_2^d)$ either 1) time converges and player 1 is to blame or 2) time diverges in run r_d and r_d contains a location satisfying p, but not before time d. Player 1 does not have anything to gain by blocking time, so assume time diverges in run r_d (or equivalently, assume π_1 to be a reasonable strategy by Proposition 1). The only way strategies π_2^d and runs r_d can exist for every $d > 0$ is if player 2 can *force* the game (while avoiding p) so that a portion of the run lies in a region cycle $R_{k_1}, \ldots R_{k_m}$, with *tick* being true in one of the regions of the cycle (note that a system may stay in a region for at most one time unit). Now, if a player can control the game from state s so that the next state lies in region R, then he can do the same from any state s' such that $s' \cong s$. Thus, it must be that player 2 has a strategy π_2^* such that a run in $\mathsf{Outcomes}(s, \pi_1, \pi_2^*)$ corresponds to the region sequence $R_0, \ldots, R_k, (R_{k_1}, \ldots R_{k_m})^\omega$, with none of the regions satisfying p. Time diverges in this run as *tick* is infinitely often true due to the repeating region cycle. This contradicts the fact the π_1 was a winning strategy for player 1 for $\langle\langle 1 \rangle\rangle \Diamond p$. $\square$

Lemma 3 suggests the following algorithm for computing the minimal time to reach p to within a precision of one: first confirm that $\langle\langle 1 \rangle\rangle \Diamond p$ holds, then iteratively check whether $\langle\langle 1 \rangle\rangle \Diamond_{\leq k} p$ holds for $k = 0, 1, 2, \ldots$. Any desired degree of precision can be acheived by the standard trick of "blowing" up the timed automaton. The algorithm is exponential both in the size of the timed automaton game and in the number of bits used to encode the desired precision.

Theorem 2. *Given a timed automaton game $\mathcal{T}$, a state s, and a target proposition p, the minimal time $T^i_{\min}(s, p)$ for player $i \in \{1, 2\}$ to force the game from s to a p state can be computed to within any desired degree of precision.*

The dual *maximal time problem* asks what is the maximal time for which player i can ensure that the system stays within p states. Corresponding results hold for the maximum time problem — it can be computed it to within any desired degree of precision in timed automaton games in EXPTIME.

The problems of computing the exact minimal and maximal times for timed automaton games are open.

References

1. B. Adler, L. de Alfaro, and M. Faella. Average reward timed games. In *FORMATS 05*, LNCS 3829, pages 65–80. Springer, 2005.
2. R. Alur, M. Bernadsky, and P. Madhusudan. Optimal reachability for weighted timed games. In *ICALP 04*, LNCS 3142, pages 122–133. Springer, 2004.
3. R. Alur, C. Courcoubetis, and D.L. Dill. Model-checking in dense real-time. *Inf. and Comp.*, 104(1):2–34, 1993.
4. R. Alur and D.L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.
5. R. Alur and T.A. Henzinger. A really temporal logic. *Journal of the ACM*, 41:181–204, 1994.
6. R. Alur, T.A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *Journal of the ACM*, 49:672–713, 2002.
7. E. Asarin and O. Maler. As soon as possible: Time optimal control for timed automata. In *HSCC 99*, LNCS 1569, pages 19–30. Springer, 1999.
8. P. Bouyer, F. Cassez, E. Fleury, and K.G. Larsen. Optimal strategies in priced timed game automata. In *FSTTCS 04*, LNCS 3328, pages 148–160. Springer, 2004.
9. P. Bouyer, D. D'Souza, P. Madhusudan, and A. Petit. Timed control with partial observability. In *CAV 03*, LNCS 2725, pages 180–192. Springer, 2003.
10. T. Brihaye, V. Bruyère, and J.F. Raskin. On optimal timed strategies. In *FORMATS 05*, LNCS 3829, pages 49–64. Springer, 2005.
11. F. Cassez, A. David, E. Fleury, K.G. Larsen, and D. Lime. Efficient on-the-fly algorithms for the analysis of timed games. In *CONCUR 05*, LNCS 3653, pages 66–80. Springer, 2005.
12. C. Courcoubetis and M. Yannakakis. Minimum and maximum delay problems in real-time systems. *Formal Methods in System Design*, 1(4):385–415, 1992.
13. L. de Alfaro, M. Faella, T.A. Henzinger, R. Majumdar, and M. Stoelinga. The element of surprise in timed games. In *CONCUR 03*, LNCS 2761, pages 144–158. Springer, 2003.
14. D. D'Souza and P. Madhusudan. Timed control synthesis for external specifications. In *STACS 02*, LNCS 2285, pages 571–582. Springer, 2002.
15. M. Faella, S. La Torre, and A. Murano. Automata-theoretic decision of timed games. In *VMCAI 02*, LNCS 2294, pages 94–108. Springer, 2002.
16. M. Faella, S. La Torre, and A. Murano. Dense real-time games. In *LICS 02*, pages 167–176. IEEE Computer Society, 2002.
17. T.A. Henzinger and P.W. Kopke. Discrete-time control for rectangular hybrid automata. *Theoretical Computer Science*, 221:369–392, 1999.
18. T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111:193–244, 1994.
19. F. Laroussinie, N. Markey, and G. Oreiby. Model checking timed ATL for durational concurrent game structures. In *FORMATS 06*, LNCS. Springer, 2006.
20. O. Maler, A. Pnueli, and J. Sifakis. On the synthesis of discrete controllers for timed systems (an extended abstract). In *STACS 95*, pages 229–242, 1995.
21. A. Pnueli, E. Asarin, O. Maler, and J. Sifakis. Controller synthesis for timed automata. In *Proc. System Structure and Control*. Elsevier, 1998.
22. H. Wong-Toi and G. Hoffmann. The control of dense real-time discrete event systems. In *Proc. of 30th Conf. Decision and Control*, pages 1527–1528, 1991.

Concurrent Semantics Without the Notions of State or State Transitions

Edward A. Lee

University of California, Berkeley
eal@eecs.berkeley.edu

Abstract. This paper argues that basing the semantics of concurrent systems on the notions of state and state transitions is neither advisable nor necessary. The tendency to do this is deeply rooted in our notions of computation, but these roots have proved problematic in concurrent software in general, where they have led to such poor programming practice as threads. I review approaches (some of which have been around for some time) to the semantics of concurrent programs that rely on neither state nor state transitions. Specifically, these approaches rely on a broadened notion of computation consisting of interacting components. The semantics of a concurrent compositions of such components generally reduces to a fixed point problem. Two families of fixed point problems have emerged, one based on metric spaces and their generalizations, and the other based on domain theories. The purpose of this paper is to argue for these approaches over those based on transition systems, which require the notion of state.

1 Introduction

In this paper, I argue that basing the semantics of concurrent systems on the notion of state and state transitions is problematic. The resulting models often have unnecessary nondeterminism in the sense that the nondeterminism is an accident of the choice of modeling technique, rather than an intrinsic or interesting property of the system under study. This complicates analysis and impedes understanding. Moreover, such models fail to be compositional, in the sense that deterministic transition systems, when composed, often become nondeterministic, and that this nondeterminism reveals few if any insights about the system. The result is poor descriptions of the composite behavior.

Introducing time into concurrent models can help, but it either reduces the problem to sequential computation or it relies on a fictional abstraction, Newtonian universal time. In Newtonian universal time, every component in a distributed system shares a common notion of time. Networked computational systems have no mechanism for establishing this common notion of time. Considerable effort is required to establish even an approximate common notion of time [26], and since it is necessarily approximate, the resulting models will be either inaccurate or unnecessarily nondeterministic. When a strongly common notion of time is assumed in the semantics, as in for example discrete-event systems, then distributed or parallel execution becomes a major challenge [19,57].

There is a strong draw, however, towards using the notions of state and state transitions in semantics. These notions are deeply rooted in our understanding of computation. Programming languages are based on these concepts (and as a consequence, adapt poorly to concurrent computation [32]). Even our foundational notions of semantics are wedded to state. In [55], for example, Winskel explains denotational semantics to be a description of commands as functions mapping a syntax into a function that maps state into state. Winskel observes, in fact, that this notation appears to not be powerful enough for parallelism and fairness (presumably because of its focus on a single global state). When it comes to modeling parallelism, in [55] Winskel focuses on Hoare's CSP [25] and Milner's CCS [42]. He gives both in terms of labeled transition systems, where the labels can include input/output operations. But labeled transition systems are intrinsically based on the notion of state. Coupling two deterministic non-interacting processes (a trivial composition) under either CSP or CCS semantics will yield a nondeterministic semantic model. This nondeterminism contributes nothing to the understanding of the system. It reflects the uninteresting and inconsequential multiplicity of possible interleavings of independent actions.

The focus on transition systems follows naturally from our core imperative notions of computation. There has been, of course, considerable exploration of concurrent alternatives to imperative models of computation. For example, in [20], Goldin et al. describe a persistent Turing machine (PTM), which has three tapes: input, working, and output. It continually processes data from the input, producing outputs. Networks of these can interact. A "universal PTM" can simulate the actions of any PTM. Goldin et al. argue that any "sequential interactive computation" (which has only a causality restriction) can be represented by a PTM. PTM's have stream and actor semantics, and are intrinsically concurrent. But they are not compositional. Consider again two non-interacting PTSs. Without synchronization between these, the resulting composition is not usefully modeled as a PTM. Similarly, in [22], Gössler and Sifakis argue eloquently for a separation of behavior models from interaction models. But "behavior" is again given as transition systems. Once again, without imposing strong synchronization, a composition of behaviors is not usefully modeled as a behavior.

Of course, one can introduce synchronization to alleviate these problems. The synchronous languages [8] take a particularly strong stance on this, where concurrent computations are simultaneous and instantaneous, and at each "tick" of a global "clock" variables have a value given denotationally as a fixed point. This model is clean and compositional, but has proved notoriously difficult to implement in parallel or distributed systems. This has led to a focus on "global asynchronous, locally synchronous" (GALS) models [9]. These are, by definition, not compositional, since compositions of asynchronously interacting components cannot be again made synchronous without introducing unnecessary nondeterminism.

Even our notions of equivalence between systems are deeply connected to the notion of state. In [43], Milner originated the idea of observational equivalence as mutual simulation (and bisimulation). Simulation and bisimulation are relations on states, and hence become much less useful when system state is not a well-defined concept. To be sure, the formalisms apply, but only at the expense of the introduction of nondeterminism that is likely not intrinsic to the system being modeled.

To get a sense of just how deeply rooted the concept of state and state transitions are, consider next the very notion of computation.

2 Computation as Transformation of State

Let $\mathbb{N} = \{0, 1, 2, \cdots\}$ represent the natural numbers. Let $B = \{0, 1\}$ be the set of binary digits. Let B^* be the set of all finite sequences of bits, and

$$B^\omega = (\mathbb{N} \to B)$$

be the set of all infinite sequences of bits (each of which is a function that maps $\mathbb{N}$ into B). Following [15], let $B^{**} = B^* \cup B^\omega$. We will use B^{**} to represent the state of a computing machine, its (potentially infinite) inputs, and its (potentially infinite) outputs. Let

$$Q = (B^{**} \rightharpoonup B^{**})$$

denote the set of all partial functions with domain and codomain B^{**}.

An *imperative machine* (A, c) is a finite set $A \subset Q$ of *atomic actions* and a *control function* $c\colon B^{**} \to \mathbb{N}$. The set A represents the atomic actions (typically instructions) of the machine and the function c represents how instructions are sequenced. We assume that A contains one *halt* instruction $h \in A$ with the property that

$$\forall\, b \in B^{**}, \quad h(b) = b.$$

That is, the halt instruction leaves the state unchanged.

A *sequential program* of length $m \in \mathbb{N}$ is a function

$$p\colon \mathbb{N} \to A$$

where

$$\forall\, n \geq m, \quad p(n) = h.$$

That is, a sequential program is a finite sequence of instructions tailed by an infinite sequence of halt instructions. Note that the set of all sequential programs, which we denote P, is a countably infinite set.

An execution of this program is a *thread*. It begins with an initial $b_0 \in B^{**}$, which represents the initial state of the machine and the (potentially infinite) input, and for all $n \in \mathbb{N}$,

$$b_{n+1} = p(c(b_n))(b_n). \tag{1}$$

Here, $c(b_n)$ provides the index into the program p for the next instruction $p(c(b_n))$. That instruction is applied to the state b_n to get the next state b_{n+1}. If for any $n \in \mathbb{N} \quad c(b_n) \geq m$, then $p(c(b_n)) = h$ and the program halts in state b_n (that is, the state henceforth never changes). If for all initial states $b_0 \in B$ a program p halts, then p defines a total function in Q. If a program p halts for some $b_0 \in B$, then it defines a partial function in Q.[1]

We now get to the core appeal that sequential programs have. Given a program and an initial state, the sequence given by (1) is defined. If the sequence halts, then the function computed by the program is defined. Any two programs p and p' can be compared. They are equivalent if they compute the same partial function. That is, they are equivalent if they halt for the same initial states, and for such initial states, their final state is the same.[2] Such a theory of equivalence is essential for any useful formalism. But in extending this theory of equivalence to concurrent systems, we are tempted to expose the internal sequence of state transformations, which proves to be an enormous mistake. This mistake underlies the technology of multithreaded programming, on which most concurrent software is built.

The essential and appealing properties of programs are lost when multiple threads are composed. Consider two programs p_1 and p_2 that execute concurrently in a multithreaded fashion. What we mean by this is that (1) is replaced by

$$b_{n+1} = p_i(c(b_n))(b_n) \quad i \in \{1, 2\}. \tag{2}$$

At each step n, either program may provide the next (atomic) action. Consider now whether we have a useful theory of equivalence. That is, given a pair of multithreaded programs (p_1, p_2) and another pair (p_1', p_2'), when are these two pairs equivalent? A reasonable extension of the basic theory defines them to be equivalent if *all interleavings* halt for the same initial state and yield the same final state. The enormous number of possible interleavings makes it extremely difficult to reason about such equivalence except in trivial cases (where, for example, the state B^{**} is partitioned so that the two programs are unaffected by each others' partition). Even in such trivial cases, we need to extend the semantics in some way to explicitly talk about isolation of state.

Even worse, given two programs p and p' that are equivalent when executed according to (1), if they are executed in a multithreaded environment, we can no longer conclude that they are equivalent. In fact, we have to know about all other threads that might execute (something that may not itself be well defined), and we would have to analyze all possible interleavings. We conclude that with threads, there is no useful theory of equivalence.

[1] Note that a classic result in computing is now evident. It is easy to show that Q is not a countable set. (Even the subset of Q of constant functions is not countable, since B^{**} itself is not countable. This can be easily demonstrated using Cantor's diagonal argument.) Since the set of all finite programs P is countable, we can conclude that not all functions in Q can be given by finite programs. That is, any sequential machine has limited expressiveness. Turing and Church [54] demonstrated that many choices of sequential machines (A, c) result in programs P that can give exactly the same subset of Q. This subset is called the *effectively computable functions*.

[2] In this classical theory, programs that do not halt are all equivalent. This creates serious problems when applying the theory of computation to embedded software, where useful programs do not halt [30].

This problem shows up in practical situations. Building non-trivial multithreaded program with predictable behavior is notoriously difficult [32]. Moreover, implementing a multithreaded model of computation is extremely difficult. Witness, for example, the deep subtleties with the Java memory model (see for example [47] and [21]), where even astonishingly trivial programs produce considerable debate about their possible behaviors.

The core abstraction of computation given by (1), on which all widely-used programming languages are built, emphasizes deterministic composition of deterministic components. The actions are deterministic and their sequential composition is deterministic. Sequential execution is, semantically, function composition, a neat, simple model where deterministic components compose into deterministic results.

Threads, on the other hand, are wildly nondeterministic. The job of the programmer is to prune away that nondeterminism. We have, of course, developed tools to assist in the pruning. Semaphores, monitors, and more modern overlays on threads (see [32] for a discussion of these) offer the programmer ever more effective pruning.

A model based on nondeterministic interleavings of state transformations is not a useful model. We should build neither programming techniques nor semantics on it. In [32], I have discussed alternative programming techniques. Here I discuss alternative approaches to semantics. As with with the programming techniques, these alternative approaches are not new (mostly). The purpose of this paper is to argue their superiority, not to introduce new techniques.

3 Tagged Signal Model

Instead of functions of the form

$$f\colon B^{**} \to B^{**}$$

concurrent computation can be given in terms of functions of the form

$$f\colon (\mathcal{T} \to B^{**}) \to (\mathcal{T} \to B^{**}), \tag{3}$$

where $(\mathcal{T} \to B^{**})$ is the set of functions with domain $\mathcal{T}$ and codomain B^{**}. In the above, $\mathcal{T}$ is a partially or totally ordered set of *tags*, where the ordering can represent time, causality, or more abstract dependency relations. A computation viewed in this way maps an evolving bit pattern into an evolving bit pattern. This formulation is based on the "tagged signal model" [35], and it has been shown adaptable to many concurrent models of computation [9,12,37]. The mathematical structure of $\mathcal{T}$ can be highly variable, depending on the concurrency model, and can model tight synchronization (as in the synchronous languages) and loose synchronization (as in stream processing).

The tagged signal model is similar in objectives to the coalgebraic formalism of abstract behavior types in [4], interaction categories [1], and interaction semantics [51]. As with all three of these, the tagged signal model seeks to model a variety of interaction styles between concurrent components, without focusing on their state. Components may have state, and may execute through state transformations, but this is an irrelevant implementation detail. The notion of state is not exposed at the interface, and consequently the same model can be used for compositions of components where the state of the composition is not well-defined.

When computational components are given in the form of (3), we call the components *actors* [31]. Their environment (which can include other actors) provides them with data, and they react and possibly provide the environment with additional data. As suggested by the name, the classical "actor model" [3,24] is actor-oriented in our sense. In the actor model, components have their own thread of control and interact via message passing. We are using the term "actors" more broadly, inspired the analogy with the physical world, where actors control their own actions. In this sense, the synchronous languages [8], for example, are also actor-oriented. Asynchronous dataflow models are also actor-oriented in our sense, including both Kahn-MacQueen process networks [28], where each component has its own thread of control, and Dennis-style dataflow [17], where components (also called "actors" in the original literature) "fire" in response to the availability of input data. In our conception, however, compositions of actors are also actors. It does not matter whether the composition has a single thread of control or a well-defined "firing."

A number of component architectures that are not commonly considered in software engineering also have an actor-oriented nature and are starting to be used as source languages for embedded software [33,30]. Discrete-event (DE) systems, for example, are commonly used in circuit design and in modeling and design of communication networks [13,5]. In DE, components interact via events, which carry data and a time stamp, and reactions are chronologically ordered by time stamp. In continuous-time (CT) models, such as those specified in Simulink (from The MathWorks) and Modelica [53], components interact via (semantically) continuous-time signals, and execution engines approximate the continuous-time semantics with discrete traces.

When computational components are given in the form of (3), concurrent composition can take the form of ordinary function composition. By contrast, under the imperative model, the clean and simple mechanism of function composition is only applicable to sequential composition.

For components of form of (3), the domain and range of the function f are themselves functions, with form $x \colon \mathcal{T} \to B^{**}$. We call these functions "signals." They represent a potentially infinite evolving bit pattern. To build a useful semantic model for general concurrent systems, however, we need to broaden this simplistic model. For one, the model is much more usable if components have multiple inputs and outputs. This is easy to accomplish.

First, we generalize the notion of a signal. An *event* is a pair (t, v), where $t \in \mathcal{T}$ and $v \in \mathcal{V}$, a set of values. The set of events is $\mathcal{E} = \mathcal{T} \times \mathcal{V}$. A *signal* s is a subset of $\mathcal{E}$. So the set of all signals is $\mathcal{P}(\mathcal{E})$, the power set. A *functional signal* s is a partial function from $\mathcal{T}$ to $\mathcal{V}$, meaning that if $(t, v_1) \in s$ and $(t, v_2) \in s$, then $v_1 = v_2$. We denote the set of all functional signals by $S = (\mathcal{T} \rightharpoonup \mathcal{V})$. We will only consider functional signals here, so when we say "signal" we mean "functional signal."

Next we generalize the notion of an actor. An actor is associated with a set of *ports*. Actors receive and produce events on ports. Thus, a port is associated with a signal, which is a set of events. Given a set P of ports, a *behavior* is a function

$$\sigma \colon P \to S.$$

That is, a behavior for a set of ports assigns to each port $p \in P$ a signal $\sigma(p) \in S$.

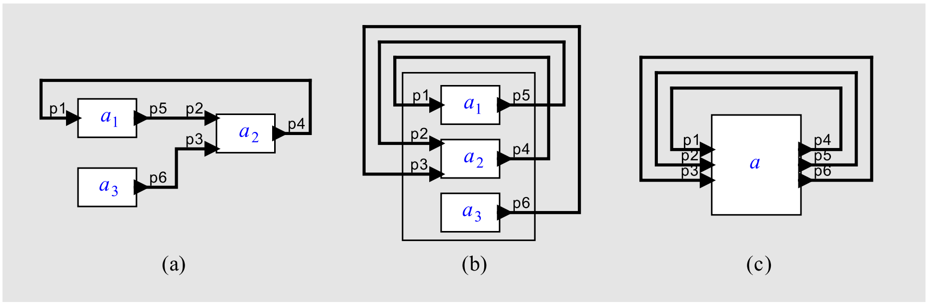


Fig. 1. A composition of three actors and its interpretation as a feedback system

Three actors with ports are depicted graphically in figure 1(a). The actors are represented by rectangular boxes and the ports by small black triangles. At each port, there is a signal. Note that nothing in our formalism so far constrains the set $\mathcal{V}$ of values. In particular, there is nothing to keep us from including in $\mathcal{V}$ representations of actors themselves, which would yield a higher-order formalism. In particular, since a signal represents an evolving set of values, such a higher-order formalism supports evolving actor composition structures.

An *actor* a with ports P_a is a set of behaviors,

$$a \subset (P_a \to S).$$

That is, an actor can be viewed as constraints on the signals at its ports. A signal $s \in S$ at port p is said to satisfy an actor a if there is a behavior $\sigma \in a$ such that $s = \sigma(p)$.

A *connector* c between ports P_c is also a set of behaviors,

$$c \subset (P_c \to S),$$

but with the constraint that for each behavior $\sigma \in c$, there is a signal $s \in S$ such that

$$\forall\, p \in P_c, \quad \sigma(p) = s.$$

That is, a connector asserts that the signals at a set of ports are identical. In figure 1(a), the connectors are represented as wires between ports.

Given two sets of behaviors, a with ports P_a and b with ports P_b, the *composition behavior set* is the intersection, defined as

$$a \wedge b \subset ((P_a \cup P_b) \to S),$$

where

$$a \wedge b = \{\sigma \mid \sigma \downarrow_{P_a} \in a \text{ and } \sigma \downarrow_{P_b} \in b\},$$

where $\sigma \downarrow_P$ denotes the restriction of σ to the subset P of ports.

A set A of actors (each of which is a set of behaviors) and a set C of connectors (each of which is also a set of behaviors) defines a *composite actor*. The composite actor is defined to be the composition behavior set of the actors A and connectors C. Figure 1(a) is such a composite actor.

In many such concurrent formalisms, ports are either inputs or outputs to an actor but not both. Consider an actor a with ports $P_a = P_i \cup P_o$, where P_i are the input ports and P_o are the output ports. In figure 1(a), triangles pointing into the actor rectangles represent input ports, and triangles pointing out from the rectangles represent output ports. The actor is said to be *functional* if

$$\forall\ \sigma_1, \sigma_2 \in a, \quad (\sigma_1 \downarrow_{P_i} = \sigma_2 \downarrow_{P_i}) \Rightarrow (\sigma_1 \downarrow_{P_o} = \sigma_2 \downarrow_{P_o}).$$

Such an actor can be viewed as a function from input signals to output signals. Specifically, given a functional actor a with input ports P_i and output ports P_o, we can define a function

$$F_a \colon (P_i \to S) \rightharpoonup (P_o \to S). \tag{4}$$

This function is total if any signal at an input port satisfies the actor. Otherwise it is partial. If the function is total, the actor is said to be *receptive.* A connector, of course, is functional and receptive.

An actor with no input ports (only output ports) is functional if and only if its behavior set is a singleton set. That is, it has only one behavior. An actor with no output ports (only input ports) is always functional.

A composition of actors and connectors is itself an actor. The input ports of such a composition can include any input port of a component actor that does not share a connection with an output port of a component actor. If the composition has no input ports, it is said to be *closed.* Figure 1(a) is a closed composition.

A composition is *determinate* if it is functional. Note that now a composition is determinate if and only if its observable behavior is determinate. There is no accidental nondeterminism due to the choice of modeling technique.

A key question in many such concurrent formalisms is, given a set of total functional actors and connectors, is the composition functional and total? This translates into the question of existence and uniqueness of behaviors of compositions. It determines whether a composition is determinate and whether it is receptive. This is a question of semantics.

4 Semantics

First, we observe that the semantics of any network of functional actors is the signal values. This reduces to a fixed point problem. In particular, any composition of functional actors can be restructured as a single actor with feedback connections. The composition in figure 1(a) can be redrawn as shown in figure 1(b), which suggests the abstraction shown in figure 1(c). It is easy to see that any diagram of this type can be redrawn in this way and abstracted to a single actor with the same number of input and output ports, with each output port connected back to a corresponding input port.

It is also easy to see that if actors a_1, a_2, and a_3 in figure 1(b) are functional, then the composite actor a in figure 1(c) is functional. Let F_a denote the function of the form (4) giving the behaviors of a. Then the behavior of the feedback composition in figure 1(c) is a function

$$f\colon \{p1, p2, p3\} \to S$$

that is a fixed point of F_a. That is,

$$F_a(f) = f.$$

A key question, of course, is whether such a fixed point exists (does the composition have a behavior?) and whether it is unique (is the composition determinate?).

For some models of computation, a unique semantics is assigned even when there are multiple fixed points by associating a partial order with the set S of signals and choosing the least or greatest fixed point. For dataflow models [27,11,34], a prefix order on the signals turns the set of signals into a complete partial order (CPO). Given such a CPO, we define the semantics of the diagram to be the least fixed point. The least fixed point is assured of existing if a is monotonic, and a constructive procedure exists for finding that least fixed point if a is also continuous (in the prefix order) [27]. It is easy to show that if a_1, a_2, and a_3 in figure 1(b) are continuous, then so is a in figure 1(c). Hence, continuity is a property that composes easily.

This approach builds on domain theory [2], developed for the denotational semantics of programming languages [55,50]. But unlike many semantics efforts that focus on system state and transformation of that state, it focuses on concurrent interactions, and does not even assume that there is a well-defined notion of "system state."

Note that even when a unique fixed point exists and can be found, the result may not be desirable. Suppose for example that in figure 1(c) F_a is the identity function. This function is continuous, so under the prefix order, the least fixed point exists and can be found constructively. In fact, the least fixed point assigns to each port the empty signal. We interpret this result as deadlock, because an execution of the program cannot proceed beyond the empty signals. Whether such a deadlock condition exists is much harder to determine than whether the composition yields a continuous function. In fact, it can be shown that in general this question is undecidable for dataflow models [34]. An approach that analyzes such networks for such difficulties is given in [36] and [58]. This approach defines *causality interfaces* for actors and gives an algebra for composing these interfaces.

In synchronous languages, the problem of existence and uniqueness reduces to determining existence and uniqueness at each tick of the global clock, rather than over the entire execution. In this case, we can use a flat CPO (rather than one based on a prefix order) and similarly assign a least fixed point semantics [49,10,18]. In this CPO, all monotonic functions are continuous. As in the dataflow case, continuity composes easily, but does not tell the whole story. In particular, the least fixed point may include the bottom $\perp$ of the CPO, which represents an "unknown" value. When this occurs, the program is said to have a *causality loop*. Whether a program has a causality loop can be difficult to determine in general, but one can define a conservative "constructive semantics" that enables a finite static analysis of programs to determine whether a program has a causality loop [10]. One can further define a language that needs to know very little about the actors to determine whether such a causality loop exists [18]. The causality interfaces of [36] and [58] can again be used to analyze these models for causality loops.

When $\mathcal{T}$ represents time, it is customary to define semantics using metric space approaches [6,48,56,7,16,29]. In such formulations, causality plays a central role. Causality intuitively defines the dependence that outputs from a concurrent component have on inputs to that component. In metric-space formulations, causal components are modeled as contracting functions in the metric space, conveniently enabling us to leverage powerful fixed-point theorems.

In [39], Liu, Matsikoudis, and myself recently showed that the standard metric-space formulation excessively restricts the models of time that can be used. In particular, it cannot handle super-dense time [40,41], used in hardware description languages, hybrid systems modeling, and distributed discrete-event models. Super-dense time is essential to cleanly model simultaneous events without unnecessary nondeterminism. Moreover, the metric-space approaches do not handle well finite time lines, and time with no origin. Moreover, if we admit continuous-time and mixed signals (essential for hybrid systems modeling) or certain Zeno signals, then causality is no longer equivalent to its formalization in terms of contracting functions. In [39], Liu et al. give an alternative semantic framework using a generalized ultrametric [45] that overcomes these limitations. The existence and uniqueness of behaviors for such systems comes from the fixed-point theorem of [46], but this theorem gives no constructive method to compute the fixed point. In [14] we go a step further, and for the particular case of super-dense time, we define *petrics*, a generalization of metrics, which we use to generalize the Banach fixed-point theorem to provide a constructive fixed-point theorem.

Domain-theoretic approaches, it turns out, can also be applied to timed systems, obviating the need for a metric space. The following approach is described in [38], which is based on [37]. This approach constrains the tagged signal model described above in a subtle but important way. Specifically, it assumes that the tag set is a poset $(\mathcal{T}, \leq)$, and that a signal is a partial function defined on a down set of $\mathcal{T}$ (a similar restriction is made in [44]). A subset $\mathcal{T}'$ of $\mathcal{T}$ is a down set if for all $t' \in \mathcal{T}'$ and $t \in \mathcal{T}$, $t \leq t'$ implies $t \in \mathcal{T}'$. Down sets are also called initial segments in the literature [23]. Under this approach, a signal $s : \mathcal{T} \rightharpoonup V$ is a partial function from $\mathcal{T}$ to V such that $\text{dom}(s)$ is a down set of $\mathcal{T}$, where $\text{dom}(s)$ is the subset of $\mathcal{T}$ on which s is defined.

This constraint leads to a natural prefix order on signals. For any $s_1, s_2 \in \mathcal{S}$, s_1 is a prefix of s_2, denoted by $s_1 \sqsubseteq s_2$, if and only if $\text{dom}(s_1) \subseteq \text{dom}(s_2)$, and $s_1(t) = s_2(t)$, $\forall t \in \text{dom}(s_1)$. That is, a signal s_1 is a prefix of another signal s_2 if the graph of the function s_1 is a subset of the graph of the function s_2. The prefix order on signals is a natural generalization of the prefix order on strings or sequences, and the extension order on partial functions [52]. It is shown in [38] that existence and uniqueness of behaviors are ensured by continuity with respect to this prefix order. Causality conditions are also defined that ensure liveness and freedom from Zeno conditions. In this formulation, causality does not require a metric and can embrace a wide variety of models of time in timed concurrent systems.

In summary, approaches to semantics based on the tagged signal model, rather than on the notion of state and state transitions, appear to accomplish the key objectives of studies semantics. Since they do not explicitly depend on the notion of state, they do not introduce unnecessary and uninteresting nondeterminism due to irrelevant interleavings of state transitions.

5 Conclusion

I have argued that basing the semantics of concurrent systems on the notions of state and state transitions is neither advisable nor necessary. The tendency to do this is deeply rooted in our notions of computation, but these roots have also proved problematic in concurrent programming, where they have led to such poor programming foundations as threads. I have outlined some approaches to the semantics of concurrent programs that rely on neither state nor state transitions.

Acknowledgements

This paper describes work that is part of the Ptolemy project, and reflects the ideas of many contributors. Most notably, I would like to acknowledge Adam Cataldo, Elaine Cheong, Thomas Huining Feng, Xiaojun Liu, Elefterios Matsikoudis, Yang Zhao, Haiyang Zheng, Gang Zhou, and Rachel Zhou. The Ptolemy project is supported by the National Science Foundation (NSF award number CCR-00225610), and Chess (the Center for Hybrid and Embedded Software Systems), which receives support from NSF, the State of California Micro Program, and the following companies: Agilent, DGIST, General Motors, Hewlett Packard, Microsoft, and Toyota.

References

1. S. Abramsky, S. J. Gay, and R. Nagarajan. Interaction categories and the foundations of typed concurrent programming. In M. Broy, editor, *Deductive Program Design: Proceedings of the 1994 Marktoberdorf Summer School*, NATO ASI Series F. Springer-Verlag, 1995.
2. S. Abramsky and A. Jung. Domain theory. In *Handbook of logic in computer science (vol. 3): semantic structures*, pages 1–168. Oxford University Press, Oxford, UK, 1995.
3. G. Agha. Concurrent object-oriented programming. *Communications of the ACM*, 33(9):125–140, 1990.
4. F. Arbab. Abstract behavior types : A foundation model for components and their composition. *Science of Computer Programming*, 55:3–52, 2005.
5. J. R. Armstrong and F. G. Gray. *VHDL Design Representation and Synthesis.* Prentice-Hall, second edition, 2000.
6. A. Arnold and M. Nivat. Metric interpretations of infinite trees and semantics of non deterministic recursive programs. *Fundamenta Informaticae*, 11(2):181–205, 1980.
7. C. Baier and M. E. Majster-Cederbaum. Denotational semantics in the cpo and metric approach. *Theoretical Computer Science*, 135(2):171–220, 1994.
8. A. Benveniste and G. Berry. The synchronous approach to reactive and real-time systems. *Proceedings of the IEEE*, 79(9):1270–1282, 1991.
9. A. Benveniste, L. Carloni, P. Caspi, and A. Sangiovanni-Vincentelli. Heterogeneous reactive systems modeling and correct-by-construction deployment. In *EMSOFT.* Springer, 2003.
10. G. Berry. *The Constructive Semantics of Pure Esterel.* Book Draft, 1996.
11. M. Broy and G. Stefanescu. The algebra of stream processing functions. *Theoretical Computer Science*, 258:99–129, 2001.
12. J. R. Burch, R. Passerone, and A. L. Sangiovanni-Vincentelli. Notes on agent algebras. Technical Report UCB/ERL M03/38, University of California, November 2003.
13. C. G. Cassandras. *Discrete Event Systems, Modeling and Performance Analysis.* Irwin, 1993.
14. A. Cataldo, E. A. Lee, X. Liu, E. Matsikoudis, and H. Zheng. A constructive fixed-point theorem and the feedback semantics of timed systems. In *Workshop on Discrete Event Systems (WODES)*, Ann Arbor, Michigan, 2006.
15. B. A. Davey and H. A. Priestly. *Introduction to Lattices and Order.* Cambridge University Press, 1990.
16. J. W. de Bakker and E. P. de Vink. Denotational models for programming languages: Applications of banachs fixed point theorem. *Topology and its Applications*, 85:35–52, 1998.
17. J. B. Dennis. First version data flow procedure language. Technical Report MAC TM61, MIT Laboratory for Computer Science, 1974.
18. S. A. Edwards and E. A. Lee. The semantics and execution of a synchronous block-diagram language. *Science of Computer Programming*, 48(1), 2003.
19. G. S. Fishman. *Discrete-Event Simulation: Modeling, Programming, and Analysis.* Springer-Verlag, 2001.
20. D. Goldin, S. Smolka, P. Attie, and E. Sonderegger. Turing machines, transition systems, and interaction. *Information and Computation*, 194(2):101–128, 2004.

21. A. Gontmakher and A. Schuster. Java consistency: nonoperational characterizations for Java memory behavior. *ACM Trans. Comput. Syst.*, 18(4):333–386, 2000.
22. G. Gssler and J. Sifakis. Composition for component-based modeling. *Science of Computer Programming*, 55, 2005.
23. Y. Gurevich. Evolving algebras 1993: Lipari Guide. In E. Börger, editor, *Specification and Validation Methods*, pages 9–37. 1994.
24. C. Hewitt. Viewing control structures as patterns of passing messages. *Journal of Artifical Intelligence*, 8(3):323363, 1977.
25. C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8), 1978.
26. S. Johannessen. Time synchronization in a local area network. *IEEE Control Systems Magazine*, pages 61–69, 2004.
27. G. Kahn. The semantics of a simple language for parallel programming. In *Proc. of the IFIP Congress 74*. North-Holland Publishing Co., 1974.
28. G. Kahn and D. B. MacQueen. Coroutines and networks of parallel processes. In B. Gilchrist, editor, *Information Processing*, pages 993–998. North-Holland Publishing Co., 1977.
29. E. A. Lee. Modeling concurrent real-time processes using discrete events. *Annals of Software Engineering*, 7:25–45, 1999.
30. E. A. Lee. Embedded software. In M. Zelkowitz, editor, *Advances in Computers*, volume 56. Academic Press, 2002.
31. E. A. Lee. Model-driven development - from object-oriented design to actor-oriented design. In *Workshop on Software Engineering for Embedded Systems: From Requirements to Implementation (a.k.a. The Monterey Workshop)*, Chicago, 2003.
32. E. A. Lee. The problem with threads. *Computer*, 39(5):33–42, 2006.
33. E. A. Lee, S. Neuendorffer, and M. J. Wirthlin. Actor-oriented design of embedded hardware and software systems. *Journal of Circuits, Systems, and Computers*, 12(3):231–260, 2003.
34. E. A. Lee and T. M. Parks. Dataflow process networks. *Proceedings of the IEEE*, 83(5):773–801, 1995.
35. E. A. Lee and A. Sangiovanni-Vincentelli. A framework for comparing models of computation. *IEEE Transactions on Computer-Aided Design of Circuits and Systems*, 17(12):1217–1229, 1998.
36. E. A. Lee, H. Zheng, and Y. Zhou. Causality interfaces and compositional causality analysis. In *Foundations of Interface Technologies (FIT), Satellite to CONCUR*, San Francisco, CA, 2005.
37. X. Liu. Semantic foundation of the tagged signal model. Phd thesis, EECS Department, University of California, December 20 2005.
38. X. Liu and E. A. Lee. CPO semantics of timed interactive actor networks. Technical Report EECS-2006-67, UC Berkeley, May 18 2006.
39. X. Liu, E. Matsikoudis, and E. A. Lee. Modeling timed concurrent systems. In *CONCUR*, Bonn, Germany, 2006. LNCS.
40. O. Maler, Z. Manna, and A. Pnueli. From timed to hybrid systems. In *Real-Time: Theory and Practice, REX Workshop*, pages 447–484. Springer-Verlag, 1992.
41. Z. Manna and A. Pnueli. Verifying hybrid systems. *Hybrid Systems*, pages 4–35, 1992.
42. R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer, 1980.
43. R. Milner. Elements of interaction. *Communications of the ACM*, 36:78–89, 1993.

44. H. Naundorf. Strictly causal functions have a unique fixed point. *Theoretical Computer Science*, 238(1-2):483–488, 2000.
45. S. Priess-Crampe and P. Ribenboim. Generalized ultrametric spaces I. *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, 66:55–73, 1996.
46. S. Priess-Crampe and P. Ribenboim. Fixed point and attractor theorems for ultrametric spaces. *Forum Mathematicum*, 12:53–64, 2000.
47. W. Pugh. Fixing the Java memory model. In *Proceedings of the ACM 1999 conference on Java Grande*, pages 89–98, San Francisco, California, United States, 1999. ACM Press.
48. G. M. Reed and A. W. Roscoe. Metric spaces as models for real-time concurrency. In *3rd Workshop on Mathematical Foundations of Programming Language Semantics*, pages 331–343, London, UK, 1988.
49. K. Schneider, J. Brandt, and T. Schuele. Causality analysis of synchronous programs with delayed actions. In *International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, Washington DC, USA, 2004.
50. J. E. Stoy. *Denotational Semantics.* MIT Press, Cambridge, MA, 1977.
51. C. L. Talcott. Interaction semantics for components of distributed systems. In *Formal Methods for Open Object-Based Distributed Systems (FMOODS)*, 1996.
52. P. Taylor. *Practical Foundations of Mathematics.* Cambridge University Press, 1999.
53. M. M. Tiller. *Introduction to Physical Modeling with Modelica.* Kluwer Academic Publishers, 2001.
54. A. M. Turing. Computability and λ-definability. *Journal of Symbolic Logic*, 2:153–163, 1937.
55. G. Winskel. *The Formal Semantics of Programming Languages.* MIT Press, Cambridge, MA, USA, 1993.
56. R. K. Yates. Networks of real-time processes. In E. Best, editor, *Proc. of the 4th Int. Conf. on Concurrency Theory (CONCUR)*, volume LNCS 715. Springer-Verlag, 1993.
57. B. P. Zeigler, H. Praehofer, and T. G. Kim. *Theory of Modeling and Simulation.* Academic Press, 2nd edition, 2000.
58. Y. Zhou and E. A. Lee. A causality interface for deadlock analysis in dataflow. Technical Report UCB/EECS-2006-51, EECS Department, UC Berkeley, May 12 2006.

Decidability and Expressive Power of Real Time Logics

Alexander Rabinovich

Dept. of CS, Sackler Faculty of Exact Sciences,
Tel Aviv Univversity
rabinoa@post.tau.ac.il
http://www.cs.tau.ac.il/∼rabinoa

Temporal Logic based on the two modalities "Since" and "Until" (*TL*) is popular among computer scientists as the framework for reasoning about a system evolving in time. This logic is usually referred to as the standard linear time temporal logic. The "linear time" appears in the name of this logic probably because when it was introduced by Pnueli its intended models were linear orders; more precisely, ω-models. The adjective "standard" is presumable used due to the Kamp theorem which states that it is expressively equivalent (over the ω-models) to first order monadic logic of order - a very fundamental logic.

The two logics are expressively equivalent whether the system evolves in discrete steps or in continuous time; however, for continuous time, both logics can not express properties like: "X will happen exactly after one unit of time." Unfortunately, the extension of *TL* by this modality is undecidable.

Two of the most important characteristics of specification formalisms are expressive power and the decidability/complexity of the validity and model checking problems. Over the years, different extensions of *TL* which can specify metric properties of real time were suggested.

The goal of this talk is to survey expressiveness and decidability results for temporal and predicate logics for the specification of metric properties of real time.

Extended Directed Search for Probabilistic Timed Reachability

Husain Aljazzar and Stefan Leue

Department of Computer and Information Science
University of Konstanz, Germany
{Husain.Aljazzar, Stefan.Leue}@uni-konstanz.de

Abstract. Current numerical model checkers for stochastic systems can efficiently analyse stochastic models. However, the fact that they are unable to provide debugging information constrains their practical use. In precursory work we proposed a method to select diagnostic traces, in the parlance of functional model checking commonly referred to as failure traces or counterexamples, for probabilistic timed reachability properties on discrete-time and continuous-time Markov chains. We applied directed explicit-state search algorithms, like Z*, to determine a diagnostic trace which carries large amount of probability. In this paper we extend this approach to determining sets of traces that carry large probability mass, since properties of stochastic systems are typically not violated by single traces, but by collections of those. To this end we extend existing heuristics guided search algorithms so that they select sets of traces. The result is provided in the form of a Markov chain. Such diagnostic Markov chains are not just essential tools for diagnostics and debugging but, they also allow the solution of timed reachability probability to be approximated from below. In particular cases, they also provide real counterexamples which can be used to show the violation of the given property. Our algorithms have been implemented in the stochastic model checker PRISM. We illustrate the applicability of our approach using a number of case studies.

1 Introduction

Motivation. Software debugging is an important task in the design, implementation and integration of systems. In particular Model Checking techniques have recently been used extensively to aid the developer in fixing errors. To this end it is necessary that Model Checkers provide meaningful debugging information. In Model Checking of functional properties such information can be made available without additional computational cost. In the case of a safety property violation, Model Checkers like SPIN [1] deliver a single linear failure trace from the initial state to a property violating state that may later be used in locating the cause of a property violation. In model checking parlance such a failure trace is called a *counterexample* to the desired safety property. To obtain short and therefore easy to comprehend counterexamples, search techniques such as *Breadth-First*

Search (BFS) or *Directed Model Checking* (DMC) [2], which relies on heuristics guided state space search, can be employed.

Performance and dependability models are usually represented as stochastic models describing how the system changes from state to state as time passes. In the presence of stochastic models we are not just interested in detecting functional failure behavior of the system but in the quantitative analysis of its dependability and performance. We use the terms *target state* for states which we are interested in, i.e. states satisfying a given state proposition, and *diagnostic traces* for traces leading to target states. As in the functional setting, DMC algorithms can be employed in the Model Checking of safety properties to select diagnostic traces that are meaningful in the fault localization process. However, contrary to the functional setting, in the stochastic context we are faced with two main challenges. First, indicative of the quality of a diagnostic trace is not its length, but its probability mass. Hence, in order to use heuristics guided search techniques it is necessary to find a new quality measure based on the probability mass of traces as well as heuristics functions based on this measure that steer the search along traces with high probability mass. We first addressed this problem in [3]. Second, one diagnostic trace is in general not enough to provide meaningful error information for explaining why some probabilistic safety property is satisfied or not since all diagnostic traces contribute jointly to the probability of the property. Thus, the developer needs to consider a reasonably large set of diagnostic traces in order to debug the model. Obviously, the more probability this set carries, the more expressive it is. In this paper we address this challenge using advanced heuristic guided algorithms which make it possible to incrementally select a set of diagnostic traces with a high probability mass. This set forms a Markov chain which emulates the original model with respect to the given property.

In our approach, the set of diagnostic traces is incrementally selected. Its probability mass gradually grows during the search process with every iteration. However, it can always be ensured to be a lower bound of the total probability of the given property. In other words, the total probability of the given property to be satisfied is approximated from below. In particular cases, our method can be used to generate a counterexample which shows the violation of the given property. In this case a set of traces is computed whose probability is not smaller than the given probability upper bound. In order to repair the model, the developer has to consider the computed set. That is because, it is not possible to decrease the total probability to be under the given upper bound without applying changes to this part of the model.

Related Work and State of the Art. Many heuristic strategies and algorithms have been introduced to solve problems of, amongst others, graph search and optimization. In [4], Pearl has given a widespread overview of a set of general-purpose problem solving strategies, e.g. *Best First (BF)* and *Generalized Best First (GBF)*. Also a variety of specialized directed search algorithms, e.g. A* and Z*, have been proposed. An approach how to apply heuristics guided directed search algorithms to functional explicit state Model Checking, especially for the generation of counterexamples, has been presented in [2]. Discrete- and continuous-time Markovian models, e.g. Markov chains and Markov decision processes, build a very important and widely used class of stochastic models [5,6,7]. Prevalent stochastic Model Checkers, like PRISM [8], ETMCC [9] and its successor MRMC [10], apply efficient numerical methods to analyze Markovian models with respect to performance and dependability properties expressed in a stochastic temporal logic, like the *Probabilistic CTL* (PCTL) [11] or the *Continuous Stochastic Logic* (CSL) introduced in [12] and extended in [13]. These numerical approaches reach a high degree of numerical result accuracy. However, they are memory intensive because they keep the whole state space of the model in memory. Another disadvantage of these approaches is their inability to deliver debugging information, in particular diagnostic traces. Some approaches attempt to reduce the memory consumption of stochastic Model Checking using Monte-Carlo sampling methods [14,15,16]. Our method and these approaches have the generation of explicit paths through the stochastic model in common. However, while the goal of these approaches is to perform the stochastic model checking using Monte-Carlo sampling, ours is to dissect a meaningful portion of the model. In our own precursory work we proposed an approach based on Directed Model Checking to select a diagnostic trace which carries a high probability for a given probabilistic safety property specified on a *discrete-time Markov chain* [3]. We have also proposed an approximation based on uniformization to deal with *continuous-time Markov chains.* Our approach is memory saving because it is performed on the fly.

Structure of the Paper. In Section 2 we introduce the stochastic models which we use as well as related notations and further preliminaries. Section 3 gives an overview on directed stochastic algorithms. In Section 4 we introduce our new algorithms and discuss their properties. Section 5 contains case studies and experimental evaluation of the approach. We conclude the paper in Section 6.

2 Stochastic Systems

2.1 Markov Chains

A *discrete-time Markov chains (DTMC)* is a probabilistic transition system consisting of states and transitions between them. Each transition is labeled with a numerical value called transition probability. It indicates the probability for firing this transition as the next step of the system if the system is in the origin state of the transition. Formally, we define a DTMC as follows:

Definition 1. *A labeled discrete-time Markov chain (DTMC) $\mathcal{D}$ is a quadruple (S, s_{init}, P, L), where*

- *S is a finite set of states*
- *$s_{init} \in S$ is an initial state*
- *$P : S \times S \longrightarrow [0, 1]$ is a probability matrix, satisfying that for each state s, $\sum\limits_{s' \in S} P(s, s') = 1$.*

– *$L : S \longrightarrow 2^{AP}$ is a labeling function, which assigns each state a subset of the set of atomic propositions AP. We interpret this to define the set of valid propositions in the state.*

For each pair of states s and s', $P(s, s')$ gives the probability to move from s to s'. For a pair of states s and s', a transition from s to s' is possible if and only if $P(s, s') > 0$ holds. A state s is called absorbing if $P(s, s) = 1$ and consequently, $P(s, s') = 0$ for all other states $s' \neq s$.

Example 1. Figure 1 illustrates a simple DTMC $\mathcal{D} = (S, s_0, P, L)$ with $S = \{s_0, s_1, s_2\}$. The probability matrix P is given on the left hand side of the figure. On the right hand side of the figure, the DTMC is illustrated as a state transition graph. The labeling function L can be, for example, defined as:

$$L = \{(s_0, \{idle\}), (s_1, \{active\}), (s_2, \{broken\})\},$$

where $AP = \{idle, active, broken\}$. The state s_2 is obviously absorbing.

$$P = \begin{pmatrix} 0.0 & 1.0 & 0.0 \\ 0.9 & 0.0 & 0.1 \\ 0.0 & 0.0 & 1.0 \end{pmatrix}$$

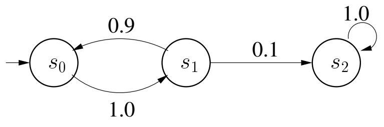


Fig. 1. A simple DTMC

Because of their simplicity, DTMCs are widely used in the modeling and analysis of stochastic systems. However, as the name already indicates, time is assumed to be discrete. If a more realistic modeling is required, then *continuous-time Markov chains (CTMCs)* are used. While each transition of a DTMC corresponds to a discrete time-step, in a CTMC transitions occur in real time. Instead of transition probability, each transition is labeled by a rate defining the delay which occurs before it is taken. The probability of a transition from s to some state s' being taken within t time units is described by a random variable which follows a negative exponential distribution with the transition rate as a parameter. To simplify matters, we illustrate our approach on DTMCs. However, we note that our approach can deal with CTMCs due to a uniformization based approximation presented in [3].

2.2 Paths and Traces

We now define the notions of *paths* and *traces* frequently used in this paper when talking about the probability mass of system execution. In this section, let $\mathcal{D} = (S, s_{init}, P, L)$ be a DTMC.

Definition 2. *An infinite path through $\mathcal{D}$ is a sequence $s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \ldots$ with, for $i \in \mathbb{N}$, $s_i \in S$ and $P(s_i, s_{i+1}) > 0$. A finite path is a finite sequence $s_0 \rightarrow s_1 \rightarrow \ldots s_{l-1} \rightarrow s_l$ with, for all $i \in \{0, \ldots, l\}$, $s_i \in S$, $P(s_i, s_{i+1}) > 0$ for all $i < l$ and s_l is absorbing.*

$Paths^{\mathcal{D}}$ denotes the set of all (finite and infinite) paths through $\mathcal{D}$. For some state s, $Paths^{\mathcal{D}}(s)$ denotes the set of all paths starting in s. For some path σ, define $states(\sigma)$ as the set of all states and $trans(\sigma)$ as the set of all transitions appearing in σ. $length(\sigma)$ is the number of transitions, i.e. $length(\sigma) = |trans(\sigma)|$. For a number i, with $0 \leq i \leq length(\sigma)$, $\sigma[i]$ is the i-th state of σ and $\sigma \uparrow i$ refers to the finite prefix of σ of the length i. The notations given in the paragraph are defined on finite prefixes as on paths.

Example 2. For the path $\sigma = s_0 \rightarrow s_1 \rightarrow s_0 \rightarrow s_1 \rightarrow s_2$ through the DTMC from Example 1, we get $states(\sigma) = \{s_0, s_1, s_2\}$ and $trans(\sigma) = \{(s_0, s_1), (s_1, s_0), (s_1, s_2))\}$. $length(\sigma)$ is 4. $\sigma[0]$, as well as $\sigma[2]$, is s_0, and $\sigma[4]$ is s_2. The 0th finite prefix of σ is s_0 and the 3rd is $s_0 \rightarrow s_1 \rightarrow s_0 \rightarrow s_1$.

The probability measure Pr of paths is defined in a standard manner on the smallest σ-algebra on $Paths^{\mathcal{D}}(s_{init})$ generated by sets of paths with a common finite prefix as follows:

$$Pr(\{\sigma \in Paths^{\mathcal{D}}(s_{init}) \mid \sigma \uparrow n = s_0 \rightarrow s_1 \rightarrow \ldots \rightarrow s_n\}) = P(s_0, s_1) \cdot \ldots \cdot P(s_{n-1}, s_n),$$

where s_0 is just another reference to the initial state, i.e. $s_0 = s_{init}$. Obviously, the set $\{\sigma \in Paths^{\mathcal{D}}(s_{init}) \mid \sigma \uparrow n = s_0 \rightarrow s_1 \rightarrow \ldots \rightarrow s_n\}$ is completely characterized by the prefix $s_0 \rightarrow s_1 \rightarrow \ldots \rightarrow s_n$. Thus, we rewrite the equation above replacing the set by the prefix itself: $Pr(s_0 \rightarrow s_1 \rightarrow \ldots \rightarrow s_n) = P(s_0, s_1) \cdot \ldots \cdot P(s_{n-1}, s_n)$.

Definition 3. *A trace is a finite sequence of states $< s_0, s_1, \ldots, s_n >$, including the self loops of all states $s_0, \ldots, s_n$, where, for all $i \in \{0, \ldots, n-1\}$, it holds that $s_i \neq s_{i+1}$ and $P(s_i, s_{i+1}) > 0$.*

For a given trace $R =< s_0, s_1, \ldots, s_n >$, we define the following notation: $first(R) = s_0, last(R) = s_n, states(R) = \{s_0, s_1, \ldots, s_n\}, trans(R) = \{(s_i, s_{i+1}) \mid 0 \leq i < n\} \cup \{(s_i, s_i) \mid 0 \leq i \leq n \wedge P(s_i, s_i) > 0\}$ and $length(R) = |states(R)| - 1$. We refer to the set of all traces through the DTMC $\mathcal{D}$ as $Traces^{\mathcal{D}}$. For a pair of states s and s', $Traces^{\mathcal{D}}(s, s')$ refers to the set containing all traces through $\mathcal{D}$ from s to s', i.e. $Traces^{\mathcal{D}}(s, s') = \{R \in Traces^{\mathcal{D}} \mid first(R) = s \wedge last(R) = s'\}$. If $\mathcal{D}$ is clear from the context we omit the respective superscript.

We point out that, using the notation of traces, we abstract from the number of repetitions of a cycle. Repeating the cycles in a concrete way results in a concrete finite prefix which induces a set of paths. Thus, each trace R can be considered as a compact representation of a set of paths $Paths(R)$ which we formally define as follows:

$$\begin{aligned} Paths(R) := \{ \sigma \in Paths(first(R)) \mid\ & \exists n \in \mathbb{N}: last(\sigma \uparrow n) = last(R) \\ & \wedge states(\sigma \uparrow n) = states(R) \\ & \wedge trans(\sigma \uparrow n) \subseteq trans(R) \}. \end{aligned} \tag{1}$$

This means that each path σ from $Paths(R)$ starts with a finite prefix from $first(R)$ to $last(R)$ which hits each state from R using exclusively transitions from R. Note that not all transitions of R have to appear in the path σ. Consequently, σ does not have to contain all self loops of R. We are usually interested in paths from $Paths(R)$ which reach $last(R)$ before a given time bound T. We call such paths *time bounded* and refer to this subset of $Paths(R)$ as $Paths(R,T)$. Formally, $Paths(R,T)$ is defined as follows: $Paths(R,T) := \{ \sigma \in Paths(R) \mid \exists n \leq T : \ last(\sigma \uparrow n) = last(R) \}$. Note that $Paths(R)$ is equal to $Paths(R, \infty)$.

Example 3. $R_1 =< s_0, s_1, s_2 >$ and $R_2 =< s_0, s_1, s_0, s_1, s_2 >$ are examples for traces in the DTMC $\mathcal{D}$ described in Example 1. The sets of paths induced by R_1 and R_2 for the time bound 4 are:

$$\begin{aligned} Paths(R_1,4) = &\quad \{\sigma \in Paths(s_0) \mid \sigma \uparrow 2 = s_0 \rightarrow s_1 \rightarrow s_2\} \\ Paths(R_2,4) = &\quad \{\sigma \in Paths(s_0) \mid \sigma \uparrow 2 = s_0 \rightarrow s_1 \rightarrow s_2\} \\ &\cup \{\sigma \in Paths(s_0) \mid \sigma \uparrow 4 = s_0 \rightarrow s_1 \rightarrow s_0 \rightarrow s_1 \rightarrow s_2\} \end{aligned}$$

Note that, $Paths(R_1, 4)$ is a subset of $Paths(R_2, 4)$.

Traces are relevant because search algorithms act on the state transition graph of the model and deliver, as will be shown later, traces as a result. Thus, it is important to define a mass to measure the stochastic quality of traces. For this purpose, we consider each trace as a set of paths. Accordingly, for a trace R and time bound T, we define a function ψ as follows:

$$\psi(R,T) = Pr(Paths(R,T)). \tag{2}$$

This definition presumes that $first(R)$ is a start state of the DTMC. Intuitively, ψ is the probability mass of the set of time bounded paths induced by the trace R.

Example 4. For the trace R_1 and R_2 from Example 3, we compute $\psi(R_1, 4)$ and $\psi(R_2, 4)$ as follows:

$$\begin{aligned} \psi(R_1,4) &= Pr(Paths(R_1,4)) = Pr(s_0 \rightarrow s_1 \rightarrow s_2) = 1.0 \cdot 0.1 = 0.1 \\ \psi(R_2,4) &= Pr(Paths(R_2,4)) = Pr(s_0 \rightarrow s_1 \rightarrow s_2) + Pr(s_0 \rightarrow s_1 \rightarrow s_0 \rightarrow s_1 \rightarrow s_2) \\ &= 1.0 \cdot 0.1 + 1.0 \cdot 0.9 \cdot 1.0 \cdot 0.1 = 0.19 \end{aligned}$$

Using the function ψ, we are now able to compare traces with respect to their stochastic quality. For two traces, the one with the higher ψ value is considered to be superior to the one with the lower value. Accordingly, we define the optimality of traces as follows.

Definition 4. *Let M be a set of traces. We call a trace $R \in M$ optimal over M, iff for any other trace $R' \in M$ the following inequality holds: $\psi(R,T) \geq \psi(R',T)$.*

We call traces, which do not contain cycles except for self loops, *forward traces*. More precisely, a trace $R =< s_0, s_1, \ldots, s_n >$ is a forward trace iff $\forall i,j \in \{0, 1, \ldots, n\} : s_i = s_j \Rightarrow i = j$.

3 Directed Algorithms for Probabilistic Timed Reachability

3.1 Probabilistic Timed Reachability Properties

In our approach we address an important class of properties for stochastic systems namely *probabilistic timed reachability* (PTR) properties. Such properties express a constraint on the probability of reaching some state satisfying a given state proposition φ within a given time period T. We restrict ourselves to φ being state propositions and call states satisfying φ *target states*, and we write $s \models \varphi$ for any target state s. PTR properties are safety properties. They are widely used in specifying dependability and performance properties of systems. PTR properties can be cast as instances of the following pattern: *"What is the probability to reach a state satisfying a state proposition φ within time period T?* In stochastic temporal logics like PCTL [11] or CSL [13], such properties are formulated as follows: $\mathcal{P}(\Diamond^{\leq T}\ \varphi)$. We note that after a slight technical modification our method can easily be applied to time-bounded *Until* formulas of the form $\mathcal{P}(\varphi_1\ U_{\leq\ T}\ \varphi_2)$.

For a given DTMC $\mathcal{D} = (S, s_{init}, P, L)$, $\mathcal{P}(\Diamond^{\leq T}\ \varphi)$ computes the probability mass of the set of all paths through $\mathcal{D}$ which lead to any target state within the time period T. Technically, $\mathcal{P}(\Diamond^{\leq T}\ \varphi)$ is determined based on the computation of the corresponding *transient probability*. For a given state $s \in S$ and a time point t, the transient probability $\pi(s,t)$ is the probability to be in s exactly at the time point t. Formally, $\pi(s,t) := \Pr\{\sigma \in \mathit{Paths}^{\mathcal{D}}(s_{init}) \mid \sigma@t = s\}$. The procedure for the computation of $\mathcal{P}(\Diamond^{\leq T}\ \varphi)$ consists of two steps. First, the stochastic model checker renders all target states absorbing, i.e. for each target state s all outgoing transitions are ignored and a self loop with transition probability 1 is added to s. Second, the model checker computes the transient probability of all target states at the time point T as follows:

$$\mathit{Prob}(\Diamond^{\leq T}\ \varphi) := \sum_{s \models \varphi} \pi(s,T) \tag{3}$$

Note that in the modified model the system can never leave a reached target state because all target states have been made absorbing. We refer to the probability defined in Equation 3, i.e. the probability of satisfying the PTR property, as the *timed reachability probability*. Occasionally it is required that the timed reachability probability is bounded by an upper or a lower bound. In these cases a bounded version of the operator $\mathcal{P}$ is used, e.g. $\mathcal{P}_{\leq p}$ or $\mathcal{P}_{\geq p}$. We use the terms *upwards* and *downwards bounded* PTR property to refer to a PTR property with a bounded $\mathcal{P}$ operator.

Diagnostics for PTR Properties. The verification of a given PTR property relies on the analysis of paths leading to target states within the given time period, as Equation 3 and the definition of π given above highlight. Thus, the developer would surely be interested in such paths when she or he is debugging the model with respect to a given PTR property. We call such paths *diagnostic* *paths*. However, a single path has a very low probability mass compared to the whole probability of the property. In the case of a CTMC, the probability of a single path is even zero. For this reason, we consider *diagnostic traces* instead of *diagnostic paths*.

Definition 5. *For a PTR property $\mathcal{P}(\Diamond^{\leq T}\ \varphi)$ specified on a DTMC $\mathcal{D} = (S, s_{init}, P, L)$, a diagnostic trace R is an element from the set $\bigcup_{s \models \varphi} \{R \in Traces^{\mathcal{D}}(s_{init}, s) \mid length(R) \leq T\}$.*

Intuitively, a diagnostic trace R is a trace from the initial state to any target state with the length of at most $T^{\star}$. It can be easily shown that $Paths(R, T)$ contains only diagnostic paths.

3.2 Best-First Search (BF)

Most of the prevalent path directed search algorithms are based on a common optimizing search strategy called Best-First (BF) [4]. This strategy uses a hash table called *CLOSED* to collect information about all explored states which have been expanded, i.e. their successors have been generated. We call such states *closed* states. Additionally, it uses a priority queue called *OPEN* to store all explored states which have not been expanded yet. We call such states *open* states. The queue *OPEN* is organized as a priority queue which is sorted according the "quality" of states. This quality is estimated numerically by an *evaluation function f*. Amongst others, f usually depends on local state information, e.g. values of the state variables, and information gathered by the search up to that point. f may also depend on the specification of the target, in our case the given PTR property, as well as further knowledge about the problem domain that may be available, in which case we call the resulting algorithm *informed* or *directed*. This knowledge is mostly encoded in a heuristic function h which is used by the evaluation function f. In each iteration BF expands the optimal open state s, i.e. the state from *OPEN* with the best f value, moves it from *OPEN* to *CLOSED*. Each successor state s' is checked on being a target state. If s' is a target state, the algorithm terminates with the solution. Otherwise, s is put into *OPEN*. Once *OPEN* is empty, the algorithm terminates without a solution. The strategy BF* is derived from BF strategy by a slight modification called *termination delay*. The termination delay means that the termination of the algorithm is delayed until a target state is selected for expansion. In both strategies BF and BF*, the evaluation function f remains arbitrary. Both strategies do not specify how f is computed. This is a major issue which has a significant impact on the search effort and the solution's nature. BF and BF* are instantiated to concrete algorithms by specifying a concrete evaluation function f or requiring f to satisfy particular conditions. For instance, the prominent A* algorithm is obtained from BF* if an additive computation of the path length is used as an evaluation function f [4]. If the evaluation function f is computed recursively,

* If $length(R)$ is greater that T, then $Paths(R, T)$ is empty. Consequently, $\psi(R, T)$ is equal to zero.

then BF becomes Z and BF* becomes Z* [4]. More precisely, in Z or Z*, when a state s is expanded, for a successor state s', $f(s')$ is computed by an arbitrary combination of the form $f(s') = F[\chi(s), f(s), h(s')]$, where $\chi(s)$ is a set of local parameters characterizing s, e.g. the weight of the transition (s, s').

3.3 Stochastic Directed Search Algorithms

In [3] we have shown how to explore the state space of a given Markov chain using (directed) search algorithms in order to generate a diagnostic trace for a given PTR property. To this end, we proposed a stochastic evaluation function for DTMCs based on the stochastic quality of traces. To be able to deal with CTMCs we proposed an approximation based on uniformization (see Section 2.1). To facilitate understanding the algorithms used in this approach we briefly recall, in this section, the algorithms which we used in [3].

Stochastic Evaluation Function. The search algorithm spans a tree called search tree. For each explored state s there exists exactly one trace R leading from the initial state to s. We define a new function γ as follows:

$$\gamma(s) := \psi(R, T), \tag{4}$$

where T is the time bound in the considered PTR property and ψ is as defined in Equation 2. Additionally, we expect a heuristic function h which estimates the stochastic quality of the optimal diagnostic trace starting in the current state, i.e. the sate we are computing the f value for. More precisely, for a given state s, let R^* be an optimal diagnostic trace starting in s, i.e. R^* is optimal over the set of all diagnostic traces starting in s (see Definition 4). $h(s)$ estimates the stochastic quality of R^*, i.e. the value $\psi(R^*, T)$, which means that we act as if s were the initial state. Note that $h(s)$ can only give a heuristic estimate based on the description of s, information gathered by the prior search and general knowledge about the problem. Although we can not give a general definition of h because it is application dependent, we proposed in [3] a method to obtain heuristic functions starting from given estimation for atomic state propositions.

The algorithms Z and Z* used in our approach use the product of γ and h as an evaluation function f. Formally, f is defined as follows:

$$f(s) := \gamma(s) \cdot h(s). \tag{5}$$

Equation 4 might induce the impression that one has to traverse the trace from a given state up to the initial state in order to compute γ for the given state. This is surely not the case. The computation is performed using information which we have gathered during the prior search and attached to the predecessor of the state which we are computing the γ value for. More concretely, we mark each open state s with a vector $\pi'(s)$ which gives the transient probabilities of s restricted on the trace from the initial state to s for the time from 0 to T. When s is expanded, for any successor state s', $\gamma(s')$ is computed according the following expression: $\gamma(s') = P(s,s') \cdot \sum_{k=0}^{t-1} \pi'(s,k)$. Altogether, f is computed recursively as follows (c.f. Section 3.2):

$$\begin{aligned} f(s') = F[\chi(s), f(s), h(s')] = F[\chi(s), h(s')] &= F[\{\pi'(s), P(s,s')\}, h(s')] \\ &= \gamma(s') \cdot h(s'). \end{aligned}$$

In order to illustrate during our later experimental evaluation the advantage of using informed search approaches we introduce two auxiliary algorithms. They are derived from Z and Z* by omitting heuristic estimate function h in the evaluation function f, i.e. $f := \gamma$. As we can observe from Equation 4, γ depends only on local state information and information gathered by the prior search. We call the resulting algorithms undirected Z (UZ) and undirected Z* (UZ*). If we use BF with just h, as described above, as an evaluation function, we get a greedy algorithm. The optimality of this algorithm is not guaranteed in general. However, it has usually a very good performance in terms of runtime as well as memory consumption. In the remainder of the paper, we refer to this algorithm simply as Greedy.

4 Extended Directed Search Algorithms

In this section we present a search strategy which extends the algorithms presented in our precursory work [3] and and which we discussed in Section 3. For a given PTR property, the new strategy makes it possible to generate not only one diagnostic trace but a set of diagnostic traces.

4.1 Extended Best-First Search (XBF)

We extend the Best-First strategy (BF) to a new strategy which we call *Extended Best-First* (XBF). The primary aim of this extension is to select a reasonable set of diagnostic traces which approximates the relevant behavior of the model, with respect to a given PTR property, from below. This set is helpful in diagnostics and can be used as a counterexample in the case of upwards bounded PTR properties. Algorithms based on BF explore the state transition graph of the model spanning a tree called traversal or search tree. Consequently, in the explored part of the state space, each state has exactly one parent. This excludes the possibility to accommodate cycles in the solution. Hence, the solution space is restricted to forward diagnostic traces. The idea is to develop a directed search strategy which explores the state transition graph of the model using a subgraph. We thus allow an explored state to have more than one parent. Additionally, XBF is designed so that it is able to select more than one target state. XBF is mainly obtained from BF by three modifications.

1. For each state we record all parent states which we find during the search. Therefore, we replace the single parent reference used in BF by a list *PARENTS* containing all parents detected by the search.

2. Additionally to *OPEN* and *CLOSED*, XBF maintains a list *TARGETS* which contains a reference for each target state encountered during the search.
3. XBF does not terminate when it finds the first solution. It continues the search for further solutions until the whole state space is processed or termination is explicitly requested.

The pseudo code of XBF is given in Algorithm 1. In the body of the **while**-loop (starting at code lines 3), there is no statement which directly causes a termination of the loop. Thus, the loop will run until the condition "*OPEN* is empty or termination is requested" is fulfilled. *OPEN* becomes empty when the state space is completely explored. The termination can also be explicitly requested using an arbitrary external condition which we refer to as the *external termination condition.* An example for that is that the number of the found diagnostic traces or the probability mass of the found solution exceeds a given bound. The **if**-statement starting at line 7 is used to detect new diagnostic traces. If the considered state s' is a target state or it is known, from prior search, that a target state is reachable from s', then we know that a new diagnostic trace is found. In this case, all known ancestor states of s' are marked as solution states. The code line 10 is useful to gather any information needed for the external termination condition, for example the number of found diagnostic traces is increased or/and the probability mass of the solution can be updated. Additionally, if s' is a target state, then it is also added to the *TARGETS* list. For each explored state s, a list *PARENTS* is used to keep pointers to all known parents of s (c.f. code lines 13, 14 and 16). Consequently, we are able to record all found traces leading to s including cycles.

Note that the stochastic evaluation function f from Equation 5 is defined based on the assumption that the explored part of the state transition graph is a tree, c.f. the definition of γ in Equation 4. This is not the case for XBF. Strictly speaking, we should redefine the function f taking into account that the explored state space is not a tree but a subgraph of the whole state space. Consequently, for a given state s, when a new trace leading to s is found, we should recompute $f(s)$ and correct the transient probability vector $\pi'(s)$ taking this trace into account. To do this, the old vector $\pi'(s)$ is needed. Therefore, we would have to store the vector π' not only for open but also for closed states. Additionally, $\pi'(s)$ was used to compute the transient probabilities of the successors of s. Thus, the whole explored part of the state space rooted at s would have to be re-explored in order to recompute π' for all states in that part. This would drastically decrease the performance of the algorithm in terms of both memory consumption and runtime. Therefore, we keep the definition of f based on the tree formed by the optimal traces. For each explored state s, let $Traces_{expl}(s_{init}, s)$ be the set of explored traces from s_{init} to s. We slightly modify the definition of γ as $\gamma(s) := \psi(R, T)$, where $R \in Traces_{expl}(s_{init}, s)$ is an optimal trace over $Traces_{expl}(s_{init}, s)$. Consequently, we reopen a state only if a better trace leading to it is found.

When the search algorithm terminates the solution is constructed using the **if**-statement at line 20. In our application, this step includes back tracking the

```
   Data: Safety property $\varphi$, the initial state $s_{init}$ and a state transition relationship
   Result: A solution if any target state is reachable
1  Initializations: OPEN ← an empty priority queue, CLOSED ← an empty hash
   table, TARGETS ← $\emptyset$ ;
2  Insert $s_{init}$ into OPEN ;
3  while OPEN is not empty and termination is not requested do
4      Remove from OPEN and place on CLOSED the state $s$ for which $f$ is
       optimal ;
5      Expand $s$ generating all its successors ;
6      foreach $s'$ successor of $s$ do
7          if $s'$ is a target state or $s'$ is marked as a solution state then
8              if $s'$ is a target state then Insert $s'$ into TARGETS ;
9              Back track all pointers from $s'$ up to $s_{init}$ marking each touched
               states as a solution state ;
10             Signal a new diagnostic trace ;
           end
11         Compute $f(s')$ ;
12         if $s'$ is not already in OPEN or CLOSED then
13             Attach to $s'$ a new list PARENTS ;
14             Add a pointer to $s$ into PARENTS of $s'$ ;
15             Insert $s'$ into OPEN ;
           end
           else
16             Add a pointer to $s$ into PARENTS of $s'$ ;
17             if the newly computed $f(s')$ is better than the old value then
18                 Replace the old value of $f(s')$ by the new one ;
19                 if $s'$ is in CLOSED then Reopen $s'$ (move it to OPEN) ;
               end
           end
       end
   end
20 if TARGETS is not empty then
21     Construct the solution
   end
22 Exit without a solution.
```

Algorithm 1. Pseudo code of Extended Best First (XBF)

pointers from all target states up to the initial state. Additionally, an extra absorbing state *sink* is added to substitute the remaining part of the model. Furthermore, for each state s contained in the solution, all outgoing transitions of s, which are not contained in the solution, are redirected to the *sink*. It is easy to show that the final result is a Markov chain. We call it the *diagnostic Markov chain* (DiagMC).

As mentioned before, the step of signaling a new trace at line 10 is used to gather information which is needed for the external termination condition. Hence, we can cause a termination delay, if this step is delayed until a target or a solution state is chosen to be expanded. Similar to BF*, we call the derived strategy XBF*. Similar to Greedy, Z, Z*, UZ and UZ*, we obtain XGreedy, XZ, XZ*, XUZ, XUZ* by modifying the evalution function f as descriped in Section 3.3. Note that XUZ and XUZ* are undirected algorithms. We consider them in this paper only to illustrate the advantage of using a heuristic function.

Example 5. Figure 2 shows a simple transition system. We can view it as the transition system of a DTMC. We assume that the model contains only one target state which we labeled by $\times$. The figure illustrates the incremental growth of the selected solution. At some point of the search the algorithm will find a diagnostic trace, for instance the one highlighted by bold lines in Figure 2 (a). If the algorithm is not explicitly stopped, it will continue to find more diagnostic traces. After some iterations the solution will grow into the subgraph highlighted in Figure 2 (b). The largest solution which can be found is given in Figure 2 (c). At the end of the algorithm, the solution is transformed into the diagnostic Markov chain illustrated in Figure 3. We refer to the DiagMC obtained from the largest solution as the *complete* DiagMC.

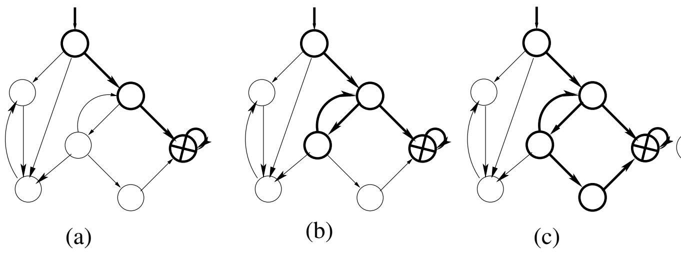


Fig. 2. Incrementally selected solution

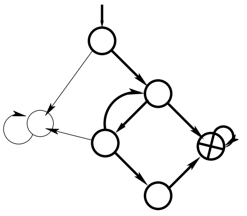


Fig. 3. The DiagMC resulting from Fig. 2 (c)

4.2 Impact of Cycles

To illustrate the effect of cycles on the timed reachability probability, we consider, again, the traces R_1 and R_2 from Example 3. Obviously, R_2, including the cycle $s_0 \rightarrow s_1 \rightarrow s_0$, has a higher probability mass than R_1. Thus, we would expect our algorithm to deliver R_2 as a solution. However, this is very difficult to realize. One reason for that is that prevalent search algorithms based on BF traverse the graph in the form of a traversal tree, i.e., each state has exactly one parent. As a consequence, cycles can not be included in the solution. If we enable the algorithm to accommodate cycles we have to allow states to have more than one parent. However, the more involved reason is the evolving complexity of the computation of f, as mentioned in Section 4.1. We try to make this point clear using the following example.

Example 6. Again, we consider the DTMC $\mathcal{D}$ given in Example 1. After the expansion of s_0, the following transient probability vectors are assigned to the states s_0 and s_1: $\pi'(s_0) = (1, 0, 0, \ldots, 0)$ and $\pi'(s_1) = (0, 1, 0, \ldots, 0)$. By the expansion of s_1, s_0 and s_2 are generated. On the one hand, $\pi'(s_1)$ is used to compute $\pi'(s_2) = (0, 0, 0.1, \ldots, 0)$ and the cycle $s_0 \rightarrow s_1 \rightarrow s_0$ is detected. We know now that a part of the probability is circulated back to s_0. So, the vector $\pi'(s_0)$ has to be corrected to $(1, 0, 0.9, 0, \ldots, 0)$. However, the vector $\pi'(s_0)$ was used in the computation of $\pi'(s_1)$. Hence, we have to reopen the state s_0 in order to correct the vectors $\pi'(s_1)$. Then, s_1 also has to be reopened in order to correct $\pi'(s_2)$. We have to repeat this procedure until the vectors $\pi'(s_0)$, $\pi'(s_1)$ and $\pi'(s_2)$ are not changed any more.

As Example 6 illustrates, we might have to repeat the search linearly in the number of cycles and the time bound T. This would drastically decrease the performance of the algorithm in terms of both memory consumption and runtime. In order to accommodate cycles while avoiding these excessive computational costs, XBF uses the following strategy. For a state s, if a new parent of s is detected, XBF records this information adding a reference of the new parent into the *PARENT* list of s. However, s is only reopened, if the newly detected trace carries a higher probability than the old one. In Example 6, $f(s_0)$ computed regarding the newly detected trace $< s_0, s_1, s_0 >$ is $\psi(< s_0, s_1, s_0 >, 4) \cdot h(s_0) = 0.9 \cdot h(s_0)$ is less than the old value $\psi(< s_0 >) = 1.0 \cdot h(s_0)$. Thus, the vector $\pi'(s_0)$ will not be corrected and s_0 will not be reopened, although the cycle $s_0 \rightarrow s_1 \rightarrow s_0$ is included.

4.3 Under-Approximation of Timed Reachability Probability

An important contribution of this paper is that the timed reachability probability is approximated from below when using our extended algorithms. As mentioned in Section 4.1, the probability mass of the selected solution incrementally grows during the search process (see Example 5). Certainly, the probability mass of the solution highlighted in Figure 2 (c) is not smaller than that of the solution given in Figure 2 (b). To show this fact we reason as follows: The probability mass of the solution is the probability mass of the set of diagnostic paths induced by all traces contained in the solution. All diagnostic paths which are possible in (b) are also possible in (c). Hence, the set of diagnostic paths induced by the solution from (b) is contained in the set of diagnostic paths induced by the solution from (c). Similar reasoning can be used for the solution from (a) and (b).

The delivered DiagMC under-approximates the relevant behavior of the original model. The *complete* DiagMC, i.e. the DiagMC obtained from the largest solution, e.g. the DiagMC given in Figure 3, contains all diagnostic paths of the original model. As a consequence, checking the property on the complete DiagMC will deliver the same result as checking it on the original model. Normally the complete DiagMC consists of only a small portion of the complete model. Thus, checking the property on the complete DiagMC can be performed much faster than checking it on the original model. In many cases it is even not required to determine the complete DiagMC for the purposes of debugging or generating counterexamples.

Counterexamples. If the PTR property which we are interested in is upwards bounded, then our method can be used to generate a real counterexample. In this case a DiagMC is computed whose probability is not smaller than the given upper probability bound. For instance, in Example 5, if the probability mass of the solution from Fig. 2(b) is higher than the bound given in the property, then the computed DiagMC suffices to show the violation of the property. It can also be used as a counterexample. Hence, the search algorithm can be stopped at this point. In order to repair the model, the developer has to consider the computed DiagMC. Note that it is not possible to decrease the total probability to be under the given upper bound without applying changes to this part of the model. Note that during the search we are unable to compute the accurate probability mass of the currently selected DiagMC. We can only compute an approximated value which is not higher than the accurate probability mass. The reason is that we avoid refreshing the transient probability vectors when a cycle is detected (see Section 4.2. In the case of a CTMC, the approximation using uniformization is another reason which prevents us from computing the accurate probability mass of the selected DiagMC. As a consequence, sometimes the algorithm runs longer than required for under-approximating an upwards bounded PTR.

4.4 Diagnostics of PTR Properties

As we mentioned in Section 3.1, our main goal is to obtain diagnostic tracesin order to facilitate diagnostics and debugging of PTR properties. Each diagnostic trace represents a potentially large set of diagnostic paths and it has a meaningful probability mass which is not equal to zero. A diagnostic trace with a higher probability mass is more essential in the debugging process than the one with a lower probability mass. In our precursory work we presented a method to select one forward diagnostic trace R which carries a large amount of probability. We suggested be used in debugging the system and called it a counterexample to the considered PTR property. Obviously, one trace might be insufficient to be a real counterexample to a PTR property. Even for diagnostics the developer should consider more than one critical diagnostic trace. However, in [3] we have developed the basic techniques for the exploration of state spaces of stochastic models using directed search strategies. The advanced algorithms presented in this paper make it possible to deliver more than one critical diagnostic trace represented in form of a diagnostic Markov chain.

5 Case Studies and Experimental Results

We have implemented our algorithms in Java 5.0 based on the *Data Structures Library* (JDSL) [17,18]. Our algorithms uses the PRISM Model Checker [8] which is designed to analyze stochastic models. The DTMCs and CTMCs that we use in our experiments are modeled in the PRISM modeling language. We use the PRISM Simulation Engine in order to generate the model state spaces on-the-fly. Our search algorithms work on the thus generated state spaces. Whenever precise numerical stochastic model checking is required, for instance in order to compute total model probabilities, it is performed by the PRISM Model Checker. We next present two case studies which we used to experimentally evaluate our method. Space limitations do not permit us to present the experimental results in full. More detail is included in [19].

Case Study 1: A Query Processing System. In this section we consider a very simple model for a query processing system given as a CTMC in PRISM. The system receives queries from clients and puts them into a queue with a maximal capacity C queries where they await processing. The system works in two different modes. In the secure mode, the processing of the queries is safer but much slower than it in the normal mode. Hence, the secure mode is switched on only if it is necessary. The system is illustrated in Figure 4 in the form of a stochastic Petri net. The initial marking of this Petri net is as follows: C tokens in *Free-Slots*, one token in *Intact* and one token in *Ready*. For the maximal queue capacity $C = 500$, the CTMC consists of about 2500 states and 5000 transitions. We assume the following transition rates: $\lambda = 0.1$ is the rate to receive a new query, $\mu_1 = 3.0$ and $\mu_2 = 1.0$ are the rates for processing a query in normal and secure mode, respectively. $\kappa = 1.0 \cdot 10^{-5}$ is the rate for an attack to be successful, and $\nu = 9.0 \cdot 10^{-5}$ is the rate for an attack to be detected which causes the secure mode to be enabled. $\xi = 5.0$ is the rate to pick a query from the queue for processing.

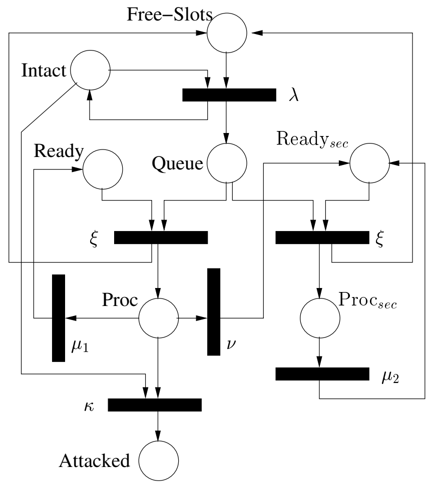


Fig. 4. SPN of the Query Processing System

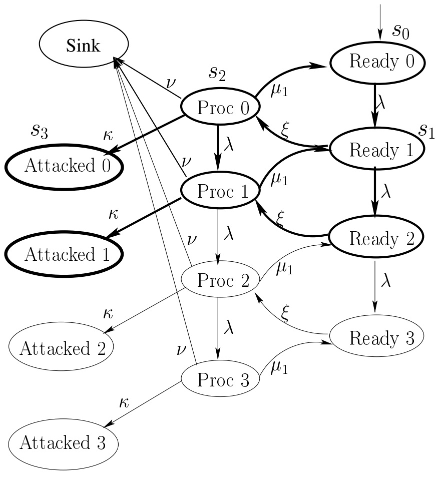


Fig. 5. The complete DiagMC in the Query Processing System

An example of an interesting PTR property is *QPS-Attacked* which expresses the interest in the probability of the system to be conquered by an attack within a particular time period. Figure 5 shows the complete DiagMC for this property in the case that $C = 3$. The numbers by which the states are labeled indicate the number of requests in the queue.

Our experimental results showed that while Z* delivers only one diagnostic trace $< s_0, s_1, s_2, s_3 >$, which is represented in the figure by the state labels s_0 to s_3, XZ* delivers a set of diagnostic traces. For instance, if we first restrict the search to select only three diagnostic traces, XZ* delivers the DiagMC indicated by bold lines in Figure 5. If we allow the algorithm to run to the end it will select the complete DiagMC given in Figure 5 which reflects the complete failure behavior of the model. It is easy to see from this DiagMC that we have to increase the rate ν if we want to reduce the likelihood of an attack to be successful. This means we have to speed up the activation of the security component. Another approach is to perform a security check on the queue to detect attacks before they reach the processor. This results in an extra transition leading from *Ready* state to $Ready_{sec}$ in Figure 4. It is mapped to a transition from *Ready* to *Sink* in the DiagMC in Figure 5.

When comparing the different algorithms in terms of solution quality, we observed that the solutions produced by the extended algorithms, especially XZ* and XUZ*, have a very high probability mass compared to the solutions of the basic algorithms. In some instances their probability masses were very close to the total probability measured for the model by precise model checking. This highlights the the effect of the guided under-approximation that we perform. We also observed that the non-optimal (respectively non-admissible) greedy algorithms Greedy and XGreedy delivered solutions with very low probabilities. However, these algorithms had the best computational performance in terms of both memory consumption and runtime measured in the number of search iterations. Generally, we observed that the directed algorithms Greedy, Z, Z*, XGreedy, XZ and XZ* explored much less states and made much less search iterations than the undirected ones UZ, UZ*, XUZ and XUZ*. In the case of unreachability of target states, the directed algorithms detected that much earlier than the undirected ones.

Case Study 2: A Workstation Cluster. The second comprehensive case study that we conducted was a dependable cluster of workstations as first presented in [20]. It represents a system consisting of two sub-clusters connected via a backbone. Each sub-cluster consists of N workstations with a central switch that provides the interface to the backbone. Each of the components of the system (workstations, switches, and backbone) can break down randomly. In order to provide minimum quality of service (QoS), at least k $(< N)$ workstations have to be operational connected to each other via operational switches. The system is modeled as a CTMC. For the maximal number of workstations per cluster (=256) the CTMC consists of about 2.3 million states. We are interested in the likelihood that the quality of service drops below the minimum within a particular time period. We ran experiments on models for different N values and restricted the search to arbitrarily chosen number of diagnostic traces.

The DiagMCs that our analysis computes indicate the most critical portion of the failure behavior of the system. Their probability masses come very close to the full probability mass measured on the complete model. In most cases, the directed search algorithms outperformed the undirected algorithms in terms of computational cost and memory consumption. However, we observed that the performance of the undirected algorithm XUZ is often similar to the performance of XZ*. Also, the quality of solution delivered by XUZ is in some cases higher than that of the solutions delivered by XZ*. The use of the greedy algorithms further reduces the computational costs, at the expense of a loss of solution quality in the order of multiple orders of magnitude.

6 Conclusion

In this paper we have presented a heuristics guided method to generate diagnostic information for the debugging of probabilistic timed reachability properties on stochastic models. For this purpose we have developed an advanced heuristic search strategy called XBF which extends the framework presented in [3]. XBF is instantiated to concrete algorithms, namely XGreedy, XZ and XZ* as well as the undirected variants XUZ and XUZ*. We have evaluated our method using a number of experiments on two case studies. Overall, the experiments showed that 1) the DiagMCs that have been computed are meaningful and useful as diagnostic information in the analysis of the model, 2) the solution delivered by the algorithms XUZ, XZ* and XUZ* have high stochastic quality, i.e. they have high probability masses, and 3) in almost all situations the directed algorithms outperformed the undirected ones in terms of computational cost.

Currently, we are investigating how the probability vectors can be interpolated so that we can avoid to store the complete transient probability vectors during the search in order to reduce overall memory consumption of the method. We are also studying visualization techniques for state spaces in order to facilitate comprehension of the diagnostic Markov chains that are provided to the user. As future work includes the application of our method to Markov decision processes which include the concept of non-determinism that is essential in analyzing concurrent stochastic models.

Acknowledgments. We thank Marta Kwiatkowska and Holger Hermanns for enlightening discussions on the subject of this paper. We thank also Dave Parker for his advice on choosing appropriate case studies. This work was carried out in the course of the DFG funded research project DiRePro.

References

1. Holzmann, G.J.: The Spin Model Checker: Primer and Reference Manual. Addision–Wesley (2003)
2. Edelkamp, S., Leue, S., Lluch-Lafuente, A.: Directed explicit-state model checking in the validation of communication protocols. International Journal on Software Tools for Technology Transfer STTT **5** (2004) 247–267

3. Aljazzar, H., Hermanns, H., Leue, S.: Counterexamples for timed probabilistic reachability. In Pettersson, P., Yi, W., eds.: FORMATS. Volume 3829 of Lecture Notes in Computer Science., Springer (2005) 177–195
4. Pearl, J.: Heuristics – Intelligent Search Strategies for Computer Problem Solving. Addision–Wesley (1986)
5. Feller, W.: An Introduction to Probability Theory and Its Applications. John Wiley & Sons (1968)
6. Stewart, W.J.: Introduction to the Numerical Solution of Markov Chains. Princeton University Press, New Jersey, USA (1994)
7. Kulkarni, V.G.: Modeling and analysis of stochastic systems. Chapman & Hall, Ltd., London, UK (1995)
8. Hinton, A., Kwiatkowska, M., Norman, G., Parker, D.: PRISM: A tool for automatic verification of probabilistic systems. In Hermanns, H., Palsberg, J., eds.: Proc. 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'06). Volume 3920 of LNCS., Springer (2006) 441–444
9. Hermanns, H., Katoen, J.P., Meyer-Kayser, J., Siegle, M.: A markov chain model checker. In: Tools and Algorithms for Construction and Analysis of Systems. (2000) 347–362
10. Katoen, J.P., Khattri, M., Zapreev, I.S.: A markov reward model checker. qest **0** (2005) 243–244
11. Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. Formal Asp. Comput. **6** (1994) 512–535
12. Aziz, A., Sanwal, K., Singhal, V., Brayton, R.: Model-checking continuous-time markov chains. ACM Trans. Comput. Logic **1** (2000) 162–170
13. Baier, C., Haverkort, B., Hermanns, H., Katoen, J.P.: Model-checking algorithms for continuous-time Markov chains. IEEE Transions on Software Engineering **29** (2003)
14. Grosu, R., Smolka, S.A.: Monte carlo model checking. In Halbwachs, N., Zuck, L.D., eds.: TACAS. Volume 3440 of Lecture Notes in Computer Science., Springer (2005) 271–286
15. Sen, K., Viswanathan, M., Agha, G.: On statistical model checking of stochastic systems. In Etessami, K., Rajamani, S.K., eds.: CAV. Volume 3576 of Lecture Notes in Computer Science., Springer (2005) 266–280
16. Younes, H.L.S., Simmons, R.G.: Probabilistic verification of discrete event systems using acceptance sampling. In Brinksma, E., Larsen, K.G., eds.: CAV. Volume 2404 of Lecture Notes in Computer Science., Springer (2002) 223–235
17. Goodrich, M., Tamassia, R.: Data Structures and Algorithms in Java (2nd edition). John Wiley & Sons, Inc., New York, NY, USA (2000)
18. JDSL Web Page: (http://www.cs.brown.edu/cgc/jdsl/)
19. Aljazzar, H., Leue, S.: Extended directed search for probabilistic timed reachability. Technical Report soft-06-03, Chair for Software Engineering, University of Konstanz, Gemany (2006) URL: http://www.inf.uni-konstanz.de/soft/research/publications/pdf/soft-06-03.pdf.
20. Haverkort, B.R., Hermanns, H., Katoen, J.P.: On the use of model checking techniques for dependability evaluation. In: SRDS. (2000) 228–237

Intersection of Regular Signal-Event (Timed) Languages

Béatrice Bérard[1], Paul Gastin[2], and Antoine Petit[2]

[1] LAMSADE, Université Paris Dauphine & CNRS,
Place du Maréchal de Lattre de Tassigny, F-75775 Paris Cedex 16, France
[2] LSV, ENS de Cachan & CNRS,
61 av. du Président Wilson, F-94235 Cachan Cedex, France

Abstract. We propose in this paper a construction for a "well known" result: regular signal-event languages are closed by intersection. In fact, while this result is indeed trivial for languages defined by Alur and Dill's timed automata (the proof is an immediate extension of the one in the untimed case), it turns out that the construction is much more tricky when considering the most involved model of signal-event automata. While several constructions have been proposed in particular cases, it is the first time, up to our knowledge, that a construction working on finite and infinite signal-event words and taking into account signal stuttering, unobservability of zero-duration τ-signals and Zeno runs is proposed.

1 Introduction

In order to accept timed words, the model of timed automata was first proposed in [1,2]. It has been widely studied for the last fifteen years and successfully applied to industrial cases. For this model, an observation, called a time-event word, may be viewed as an alternating sequence of waiting times and instantaneous actions. A timed automaton is a finite automaton extended with variables called clocks, designed to recognize time-event words: time elapses while the control stays in a given node and an event is observed when a discrete transition occurs.

Another model was introduced by [3], and further studied in [10,4,11] with the aim of describing hardware systems. In this case, an observation is a signal word, i.e. a sequence of factors a^d, where a is a signal and d is its duration. The original model of timed automata was then modified to fit this setting: a signal is emitted while the automaton stays in some state and no event is produced when a discrete transition is fired. In this framework, when a transition occurs between two states with the same signal a, we obtain a^{d_1} followed by a^{d_2}, which are merged in $a^{d_1+d_2}$. This phenomenon is called stuttering.

It was noticed in [4] that both approaches are complementary and can be combined in an algebraic formalism to obtain the so-called signal-event monoid. Timed automata can be easily adapted to take both signals and events into account, thus yielding signal-event automata: states emit signals and transitions produce events.

We consider in this paper both finite and infinite behaviors of signal-event automata and we also include unobservable events (ε-transitions) and hidden signals (τ-labeled states). These features can be very useful and even necessary, for instance for handling abstractions [6]. They also allow us to get as special cases the initial models of timed automata and signal automata.

We study in this paper a construction for the intersection of languages accepted by signal-event automata. Surprisingly, it turns out that this closure property is rather difficult to obtain. Usually, the construction for intersection relies on a basic synchronized product. In [1], which deals with infinite time-event words only (no signal is involved), a Büchi-like product is performed. The situation is more complex for signal words due to stuttering of signals and unobservability of zero-duration signals. In [3], a construction is given for the intersection of signal automata, but neither signal stuttering nor unobservability of zero-duration signal is taken into account, and only finite runs are considered. Note that the full version [4] of [3] deals with the intersection of usual timed automata only. In [10], in order to obtain a determinization result, a construction is proposed to remove stuttering and zero-duration signals on signal automata using a single clock but intersection is not considered directly. In [11], stuttering is handled but intersection is done for signal automata acting on finite sequences only and without zero-duration signals. To cope with stuttering, intermediate states and ε-transitions are added to the automaton, thus introducing all possible ways of splitting some signal a^d into a finite concatenation $a^{d_1} \dots a^{d_n}$. When dealing with ω-sequences, this approach would produce additional Zeno runs leading to another difficulty arising with the possibility to accept a finite signal-event word of finite duration with either a finite run or an infinite Zeno run.

We provide a general construction for the intersection of signal-event timed automata working on finite and infinite signal-event words. We solve the main difficulties of signal stuttering, unobservability of zero-duration τ-signals and Zeno runs. Note that, although Zeno behaviours have been studied (see for instance [12,9]), it has been sometimes argued that excluding Zeno runs directly from the semantics of timed automata is a realistic assumption, since they do not appear in "real" systems. However, the semantics of timed automata used by model-checking tools like UPPAAL do include Zeno runs while performing forward reachability analysis (this can be easily checked with an example). Hence, we think that the general theory should include Zeno runs as well.

We first give in Section 2 precise definitions of finite and infinite signal-event languages, with the corresponding notion of signal-event automata. Section 3 establishes a normal form for signal-event automata so that no infinite run accepts a finite word with finite duration and no finite run accepts a word with infinite duration. This normal form is useful for the general construction of intersection of signal-event automata dealing both with signal stuttering and with finite and infinite sequences proposed in Section 4.

For lack of space, this paper does not contain the proofs of the correctness of the different automata constructions we propose. These proofs are available in the technical report [7].

2 Signal-Event Words and Signal-Event Automata

Let Z be any set. We write Z^* (respectively Z^ω) the set of finite (respectively infinite) sequences of elements in Z, with ε for the empty sequence, and $Z^\infty = Z^* \cup Z^\omega$ the set of all sequences of elements in Z. The set Z^∞ is equipped with the usual partial concatenation defined from $Z^* \times Z^\infty$ to Z^∞.

Throughout this paper, we consider a time domain $\mathbb{T}$ which can be either the set $\mathbb{N}$ of natural numbers, the set $\mathbb{Q}_+$ of non-negative rational numbers or the set $\mathbb{R}_+$ of non-negative real numbers and we set $\overline{\mathbb{T}} = \mathbb{T} \cup \{\infty\}$.

2.1 Signal-Event Words

We now describe the most general class of systems where both piecewise-constant signals and discrete events can occur, based on the signal-event monoid defined in [4]. We consider two finite alphabets Σ_e and Σ_s, with $\Sigma = \Sigma_e \cup (\Sigma_s \times \overline{\mathbb{T}})$: an element in Σ_e is the label of an instantaneous event, while a pair $(a, d) \in \Sigma_s \times \mathbb{T}$, written a^d, associates a duration d with a signal a. Moreover, Σ_s includes the special symbol τ for an internal (or hidden) signal, the purpose of which is to represent a situation where no signal can be observed.

Intuitively, *signal-event words* (SE-words for short and sometimes called *timed words*) correspond to sequences obtained from Σ^∞ by merging consecutive identical signals and removing internal τ-signals with duration 0. But note that signals different from τ may have a null duration.

Formally, the partial monoid of signal-event words is the quotient $\Sigma^\infty / \approx$ where $\approx$ is the congruence (with respect to the partial concatenation on Σ^∞) generated by

$$\begin{cases} \tau^0 \approx \varepsilon & \text{and} \\ \prod_{i\in I} a^{d_i} \approx \prod_{j\in J} a^{d'_j} & \text{if } \sum_{i\in I} d_i = \sum_{j\in J} d'_j \end{cases}$$

where the index sets I and J above may be infinite. The partial monoid $\Sigma^\infty / \approx$ will be denoted $SE(\Sigma, \mathbb{T})$ or simply $SE(\Sigma)$ or SE when there is no ambiguity. We write a^∞ for the equivalence class of any sequence of the form $\prod_{i\geq 1} a^{d_i}$, where $\sum_{i\geq 1} d_i = \infty$. Note that for two words of the forms ua^d and $a^{d'}v$ with $d < \infty$, the concatenation is $ua^{d+d'}v$.

A finite or infinite sequence in $\Sigma^\infty \cup \Sigma^* \cdot (\Sigma_s \times \{\infty\})$ which does not contain τ^0 and such that two consecutive signals are distinct is said to be in *normal form* (NF). SE-words are often identified with sequences in normal form. A SE-word is *finite* if its normal form is a finite sequence (even if it ends with a^∞).

A duration can be associated with each element of Σ by: $\|a\| = 0$ if $a \in \Sigma_e$ and $\|a^d\| = d$ if $a \in \Sigma_s$ and $d \in \overline{\mathbb{T}}$, so that the duration of a sequence $w = s_1 s_2 \cdots$ in Σ^∞ is $\|w\| = \sum_{i\geq 1} \|s_i\| \in \overline{\mathbb{T}}$. Note that the duration restricted to finite SE-words with finite durations is a morphism from Σ^* into $(\mathbb{T}, +)$. A *Zeno* word is a SE-word with finite duration and whose normal form is infinite. A *signal-event language* (or *timed language*) is a set of SE-words.

Example 1. Let $\Sigma_e = \{f, g\}$ and $\Sigma_s = \{a, b\}$. The SE-word $w = a^3 f f g \tau^{4.5} a^1 b^5$ can be viewed as the following sequence of observations: first, the signal a during 3 time units, then a sequence of three instantaneous events ffg, then some unobservable signal during 4.5 time units, again the signal a during 1 time unit and then the signal b during 5 time units. The total duration of w is 13.5. For infinite SE-words, we have for instance: $a^3 g f a^1 \prod_{i\geq 1} a^2 \approx a^1 a^2 g f \prod_{i\geq 1} a^4$ and the normal form is written $a^3 g f a^\infty$. Note also that an infinite timed sequence in Σ^ω may be a finite SE-word with finite duration: $\prod_{i\geq 0} a^{1/2^i} \approx a^2$.

2.2 Signal-Event (Timed) Automata

Our model of *signal-event automata* (also called *timed automata* in the sequel) is a variant of the basic models proposed in the literature, integrating both instantaneous and durational semantics: signals are associated with the control states, while instantaneous events occur when the system switches between two states.

Clocks and guards. Let X be a set of variables with values in $\mathbb{T}$, called clocks. The set $\mathcal{C}(X)$ of guards or clock constraints over X consists of conjunctions of atomic formulas $x \bowtie c$, for a clock x, a constant $c \in \mathbb{T}$ and a binary operator $\bowtie$ in $\{<, \leq, =, \geq, >\}$.

A clock valuation $v : X \to \mathbb{T}$ is a mapping that assigns to each clock x a time value $v(x)$. The set of all clock valuations is $\mathbb{T}^X$. We write $v \models g$ when the clock valuation v satisfies the clock constraint g. If t is an element of $\mathbb{T}$ and α a subset of X, the valuations $v + t$ and $v[\alpha]$ are defined respectively by $(v + t)(x) = v(x) + t$, for each clock x in X and $(v[\alpha])(x) = 0$ if $x \in \alpha$, and $(v[\alpha])(x) = v(x)$ otherwise.

Signal-event (timed) automata. A Büchi signal-event automaton over the time domain $\mathbb{T}$ is a tuple $\mathcal{A} = (\Sigma_e, \Sigma_s, X, Q, Q_0, F, R, I, \ell, \Delta)$, where Σ_e and Σ_s are alphabets of events and signals, X is a finite set of $\mathbb{T}$-valued clocks, Q is a finite set of control states, $Q_0 \subseteq Q$ is a subset of initial states, $F \subseteq Q$ is a subset of final states and $R \subseteq Q$ corresponds to a Büchi acceptance condition. The mapping $I : Q \to \mathcal{C}(X)$ associates with a state $q \in Q$ an *invariant* $I(q)$ being a conjunction of constraints of the form $x \bowtie c$, with $\bowtie \in \{<, \leq\}$, and $\ell : Q \to \Sigma_s$ associates a signal with each state.

The set of transitions is $\Delta \subseteq Q \times \mathcal{C}(X) \times \Sigma_e \cup \{\varepsilon\} \times \mathcal{P}(X) \times Q$. A transition, also written $q \xrightarrow{g,a,\alpha} q'$, is labeled by a guard g, an instantaneous event in Σ_e or the unobservable event ε, and the subset α of clocks to be reset. When $a = \varepsilon$, it is called an ε-transition or a silent transition. Recall that, contrary to the untimed case, ε-transitions increase the expressive power of timed automata [5].

First examples of signal-event automata are given in Figure 1 (where double-circled nodes correspond to Büchi repeated states). The semantics of SE-automata will be given below. But intuitively,

- A SE-word is accepted by $\mathcal{A}_1$ if it is of the form $a^{d_1}b^{d_2}$ with $d_1 \geq 1$.
- A SE-word is accepted by $\mathcal{A}_2$ if it is of the form $a^{d_1}ca^{d_2}c\ldots$ with $d_i < 1$ for any i.
- Since the concatenation merges consecutive identical signals ($a^{d_1}a^{d_2} = a^{d_1+d_2}$), the language accepted by $\mathcal{A}_3$ consists of the signal a emitted for a duration $d \geq 1$.
- $\mathcal{A}_4$ accepts the signal a emitted for a duration $d \leq 1$ (note that a^1 is accepted by an infinite run with successive durations $\frac{1}{2}$, $\frac{1}{4}$, $\frac{1}{8}$, $\ldots$ for instance).

Semantics. In order to define the semantics of SE-automata, we recall the notions of path and timed run through a path. A path in $\mathcal{A}$ is a finite or infinite sequence of consecutive transitions:

$$P = q_0 \xrightarrow{g_1,a_1,\alpha_1} q_1 \xrightarrow{g_2,a_2,\alpha_2} q_2 \ldots, \text{ where } (q_{i-1}, g_i, a_i, \alpha_i, q_i) \in \Delta,\ \forall i > 0$$

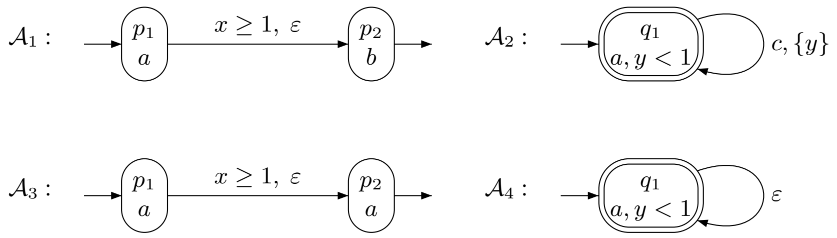


Fig. 1. Some signal automata

The path is said to be *accepting* if it starts in an initial state ($q_0 \in Q_0$) and *either* it is finite and ends in a final state, *or* it is infinite and visits infinitely often a *repeated* state $q \in R$. A *run* of the automaton through the path P is a sequence of the form:

$$\langle q_0, v_0 \rangle \xrightarrow{d_0} \langle q_0, v_0 + d_0 \rangle \xrightarrow{a_1} \langle q_1, v_1 \rangle \xrightarrow{d_1} \langle q_1, v_1 + d_1 \rangle \xrightarrow{a_2} \langle q_2, v_2 \rangle \dots$$

where

– $d_i \in \mathbb{T}$ for $i \geq 0$ and if P is finite with n transitions then the last step of the run must be $\langle q_n, v_n \rangle \xrightarrow{d_n} \langle q_n, v_n + d_n \rangle$, with $d_n \in \overline{\mathbb{T}}$,
– $(v_i)_{i \geq 0}$ are clock valuations such that $v_0(x) = 0$ for all $x \in X$, and for each $i \geq 0$, we have

$$\begin{cases} v_i + d \models I(q_i), & \forall d \in [0, d_i] \\ v_i + d_i \models g_{i+1} \\ v_{i+1} = (v_i + d_i)[\alpha_{i+1}] \end{cases}$$

Note that if d_i is finite, the condition about invariant $I(q_i)$ can be replaced simply by $v_i + d_i \models I(q_i)$.

The signal-event (timed) word generated by this run is simply (the equivalence class of) $\ell(q_0)^{d_0} a_1 \ell(q_1)^{d_1} a_2 \ell(q_2)^{d_2} \dots$. The signal-event (timed) language accepted by $\mathcal{A}$ over the time domain $\mathbb{T}$ and the alphabet Σ, written $\mathcal{L}(\mathcal{A})$, is the set of SE-words generated by (finite or infinite) accepting runs of $\mathcal{A}$. Two automata $\mathcal{A}$ and $\mathcal{B}$ are *equivalent* if $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$.

The set of all signal-event (timed) automata is denoted by SEA_ε and the family of signal-event (timed) languages generated by some automaton in SEA_ε is denoted by SEL_ε.

Remark 1. A *Zeno run* is an infinite run for which the time sequence defined by $t_i = \sum_{j \leq i} d_j$ for $i \geq 0$, is convergent (keeping the notations just above). We did not include the non Zeno condition for runs, requiring that each infinite accepting run has an infinite duration. Thus, Zeno runs accepting finite words with finite duration may occur. Note that they also appear in the semantics of model-checking tools like UPPAAL, as can easily be checked with an example.

3 Normal Forms

We propose in this section two technical results on the existence of "normal forms" for timed runs and signal-event automata. These existences will be crucial for the proof of the main result in the next section.

Recall that the normal form, obtained by merging consecutive identical signals and removing factors of the form τ^0, contains only visible events b in Σ_e and visible blocks a^d with either $d > 0$ or $a \neq \tau$. The *alternating normal form* (ANF) of a SE-word insists on a strict alternation between events and signals at the expense of keeping some invisible events (ε) and some invisible signals (τ^0).

More precisely, a sequence $w = a_0^{d_0} b_1 a_1^{d_1} b_2 \cdots$ over $\Sigma \cup \{\epsilon\}$ is in ANF if for all k we have $b_k = \varepsilon$ implies $a_{k-1} \neq a_k$ and $a_{k-1}^{d_{k-1}} \neq \tau^0 \neq a_k^{d_k}$.

Example 2. Let $\Sigma_e = \{f, g\}$ and $\Sigma_s = \{a, b\}$.

- $a^1 f a^3 g a^{2.5}$ is both in ANF and in NF,
- $f a^3 g f a^{2.5} b^4$ is in NF but not in ANF. Its ANF is $\tau^0 f a^3 g \tau^0 f a^{2.5} \varepsilon b^4$,
- $\tau^0 f \tau^0 g (a^2 \varepsilon b^3 \varepsilon)^\omega$ is in ANF but not in NF. Its NF is $fg(a^2 b^3)^\omega$.

Note that the ANF of a SE-word is unique. Indeed, assume that $w = e_0^{f_0} c_1 e_1^{f_1} c_2 \cdots$ is also in ANF. Assume that $a_0^{d_0} \neq e_0^{f_0}$. Either $a_0 = e_0$ and $d_0 < f_0$ and we must have $b_1 = \varepsilon$. We deduce that $a_1 \neq a_0$ and $a_0^{d_0} \neq \tau^0 \neq a_1^{d_1}$. A contradiction since in this case w cannot start simultaneously by $e_0^{f_0}$ and by $a_0^{d_0} b_1 a_1^{d_1}$. Or $a_0 \neq e_0$ and for instance $a_0 = \tau \neq e_0$. Then we must have $d_0 = 0$ and $b_1 = \varepsilon$, a contradiction with the ANF. Hence we have $a_0^{d_0} = e_0^{f_0}$. Assume now that $b_1 \neq c_1$. Then for instance $c_1 \neq \varepsilon$ and we must have $b_1 = \varepsilon$. Once again, this implies $a_1 \neq a_0$ and $a_0^{d_0} \neq \tau^0 \neq a_1^{d_1}$ leading to a contradiction as above. By induction, this shows that the ANF is unique.

We extend the ANF to runs and paths of a SE-automaton $\mathcal{B}$ as follows. Let $\rho = \langle p_0, v_0 \rangle \xrightarrow{d_0} \langle p_0, v_0 + d_0 \rangle \xrightarrow{b_1} \langle p_1, v_1 \rangle \xrightarrow{d_1} \cdots$ be a run for a SE-word w through some path $P = p_0 \xrightarrow{g_1, b_1, \alpha_1} p_1 \xrightarrow{g_2, b_2, \alpha_2} \cdots$ of $\mathcal{B}$. For the sake of simplicity, we assume that ρ and P are infinite. Our construction is similar when ρ and P are finite. For $0 \leq i < k \leq \infty$, we define $\ell(i,k) = \{\ell(p_j) \mid i \leq j < k \text{ and } \ell(p_j)^{d_j} \neq \tau^0\}$.

We build inductively the ANF of ρ and P and we simultaneously fix some notation. We start with $j_0 = 0$. Assume that $j_k < \infty$ has been defined. Intuitively, from the state p_{j_k}, we look for the first state where either an event $b \neq \epsilon$ is performed or a different signal is produced. Formally, we let $j_{k+1} = \inf\{j > j_k \mid b_j \neq \varepsilon \text{ or } |\ell(j_k, j+1)| = 2\}$ with the convention $\inf(\emptyset) = \infty$. Let P_k be the subpath of P starting at p_{j_k} and ending at p_{n-1} if $n = j_{k+1} < \infty$ and similarly, let ρ_k be the subrun of ρ starting at $\langle p_{j_k}, v_{j_k} \rangle$ and ending at $\langle p_{n-1}, v_{n-1} + d_{n-1} \rangle$ if $n = j_{k+1} < \infty$. Let also $a_k = \tau$ if $\ell(j_k, j_{k+1}) = \emptyset$ and $\{a_k\} = \ell(j_k, j_{k+1})$ otherwise. Finally, let $D_k = \sum_{j_k \leq j < j_{k+1}} d_j$. For $j_k < j < j_{k+1}$ we have $b_j = \varepsilon$ and for $j_k \leq j < j_{k+1}$ we have either $\ell(p_j) = a_k$ or $\ell(p_j) = \tau$ and $d_j = 0$. Hence, ρ_k is a run for $a_k^{D_k}$ through P_k.

By construction, we have $P = P_0 \xrightarrow{g_{j_1}, b_{j_1}, \alpha_{j_1}} P_1 \xrightarrow{g_{j_2}, b_{j_2}, \alpha_{j_2}} P_2 \cdots$, and $\rho = \rho_0 \xrightarrow{b_{j_1}} \rho_1 \xrightarrow{b_{j_2}} \rho_2 \cdots$ which are the ANF of P and ρ respectively. We also have $w = a_0^{D_0} b_{j_1} a_1^{D_1} b_{j_2} a_2^{D_2} \cdots$ and this is the ANF of w. Indeed, if $b_{j_k} = \varepsilon$ then $|\ell(j_{k-1}, j_k +$

$1)| = 2$ hence $a_{k-1}^{D_{k-1}} \neq \tau^0$, $\ell(p_{j_k})^{d_{j_k}} \neq \tau^0$ and $\ell(p_{j_k}) \neq a_{k-1}$. We deduce that $a_k = \ell(p_{j_k}) \neq a_{k-1}$ and $a_k^{D_k} \neq \tau^0$.

A normal form for signal-event automata. We show how to transform a signal-event automaton into an equivalent one, in which finite accepting runs correspond exactly to finite words with finite duration. The result is interesting in itself for implementation issues: when a finite word with finite duration is accepted by an infinite run, we build instead a finite accepting run for this word. Furthermore, conditions (†) and (‡) below will be used in the next section for the intersection construction.

Note that the transformation removes a particular type of Zeno runs, those which contain ultimately only ε-transitions and a single signal. But it keeps Zeno runs corresponding to infinite words of finite duration.

Theorem 1. *Let $\mathcal{A}$ be a SE-automaton. We can effectively construct an equivalent SE-automaton $\mathcal{A}'$ such that:*

(†) *no infinite run of $\mathcal{A}'$ accepts a finite word with finite duration, and*
(‡) *no finite run of $\mathcal{A}'$ accepts a word with infinite duration.*

We start from an automaton $\mathcal{A} = (\Sigma_e, \Sigma_s, X, Q, Q_0, F, R, I, \ell, \Delta)$. We first construct an automaton satisfying condition (†). We need to distinguish whether time progresses infinitely often or not.

First step. We first suppress infinite runs where time does not progress infinitely often. Such a run generates a word of the form wa^h with $h < \infty$ and loops eventually through some repeated state r. The loop is constituted of ε-transitions and states of label a or τ, all crossed instantaneously. The intuitive idea is to detect such a loop and to replace it by a single transition to a new final state. More formally, for any signal a, we build an automaton $\mathcal{A}(a, 0)$ which contains the states and transitions of $\mathcal{A}$ together with some new states and transitions described below (see Fig. 2 for an intuitive view).

Let $z \notin X$ be a new clock. The automaton $\mathcal{A}(a, 0)$ contains the states and transitions of $\mathcal{A}$. Moreover, for each $(p, g, \varepsilon, \alpha, q) \in \Delta$ with $\{\ell(p), \ell(q)\} \subseteq \{a, \tau\}$, we add the following states and transitions:

$$(1) \qquad \begin{array}{ll} (p, g, \varepsilon, \alpha \cup \{z\}, \overline{q}) & \text{if } q \text{ is a repeated state of } \mathcal{A} \\ (\overline{p}, g, \varepsilon, \alpha, (q, p)) & \\ ((p, r), g, \varepsilon, \alpha, (q, r)) & \end{array}$$

For the new states, we let $\ell(\overline{r}) = \ell(r)$, $I(\overline{r}) = I(r)$, $\ell(p, r) = \ell(p)$, $I(p, r) = I(p)$. These new transitions simulate a loop around some repeated state r: just before reaching r, we move into the copy and reach $\overline{r}$ instead. We remember r until reaching (r, r),

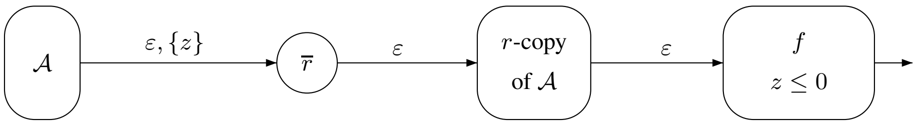


Fig. 2. The automaton $\mathcal{A}(a, 0)$

where we know that the loop has been successfully completed. Therefore, we also add the transitions $((r,r),\varepsilon,f)$ where f is a new state which is the only final state of $\mathcal{A}(a,0)$, with $\ell(f)=\tau$ and $I(f) = z \le 0$. The invariant of f ensures that the states of the form $\overline{r}$ or (p,q) have been crossed instantaneously.

The automaton $\mathcal{A}(a,0)$ has no repeated states and its initial states are those of $\mathcal{A}$.

Second step. We now treat the case of Zeno runs where time progresses infinitely often. As in the first step, the idea is to introduce copies of $\mathcal{A}$ in which we unfold once a loop around some repeated state to check if it can be taken infinitely often within finite time. But the construction is much more tricky since timing constraints have to be verified.

Let wa^d be a finite word with finite duration accepted by $\mathcal{A}$ with some infinite run ρ through some infinite path P and such that time progresses infinitely often in ρ. We denote by $\mathcal{L}_f(\mathcal{A},a)$ the set of such words. Let Y be the set of clocks that are not reset infinitely often in P. For each $y \in Y$, let $m_y = \sup\{c \mid y > c \text{ or } y \ge c \text{ occurs infinitely often in the guards of } P\}$ with the convention $\sup\emptyset = 0$. Similarly, for each $y \in Y$, let $M_y = \inf\{c \mid y < c \text{ or } y \le c \text{ occurs infinitely often in the guards or invariants of } P\}$ with the convention $\inf\emptyset = \infty$. Finally, we let $m = (m_y)_{y\in Y}$ and $M = (M_y)_{y\in Y}$. Note that all transitions (p,g,b,α,q) used infinitely often in P satisfy

(2)
- $\ell(p), \ell(q) \in \{a,\tau\}$, $b=\varepsilon$ and $\alpha \cap Y = \emptyset$,
- $\forall y \in Y$, if $y > c$ or $y \ge c$ is a constraint in g then $m_y \ge c$,
- $\forall y \in Y$, if $y < c$ or $y \le c$ is a constraint in g, $I(p)$ or $I(q)$ then $M_y \le c$,
- $\forall x \in X \setminus Y$, if $x > c$ or $x \ge c$ is a constraint in g then $c = 0$.

The last condition holds since wa^d has a finite duration.

We build an automaton $\mathcal{A}(a,Y,m,M)$ depending only on (a,Y,m,M) which accepts wa^d with a finite run (note that the number of distinct tuples (a,Y,m,M) is finite). Let $z \notin X$ be a new clock. The automaton $\mathcal{A}(a,Y,m,M)$ contains the states and transitions of $\mathcal{A}$ together with some new states and transitions described below (see Fig. 3 for an intuitive view). It has no repeated states and its initial states are those of $\mathcal{A}$. Let $(p,g,\varepsilon,\alpha,q) \in \Delta$ be a transition of $\mathcal{A}$ satisfying (2). We add the following states and transitions:

$$(3)\quad \begin{array}{ll} (p, g \wedge \bigwedge_{y\in Y} y \ge m_y, \varepsilon, \alpha\cup\{z\}, \overline{q}) & \text{if } \ell(q)=a,\\ (\overline{p}, g \wedge z > 0, \varepsilon, \alpha\cup\{z\}, (q,p,\alpha,(q\in R))) & \text{if } \ell(p)=a,\\ ((p,r,\beta,\varphi), g, \varepsilon, \alpha\cup\{z\}, (q,r,\alpha\cup\beta,\varphi\vee(q\in R))) & \end{array}$$

For the new states, we let $\ell(\overline{r}) = a$, $I(\overline{r}) = I(r)$, $\ell(p,r,\beta,\varphi) = \ell(p)$, $I(p,r,\beta,\varphi) = I(p)$ if $\ell(p) = a$ and $I(p,r,\beta,\varphi) = I(p) \wedge z \le 0$ otherwise. These new transitions

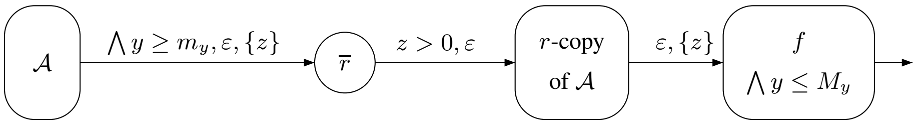


Fig. 3. The automaton $\mathcal{A}(a,Y,m,M)$

simulate a loop around some repeated state r: just before reaching r, we move into the copy and reach $\overline{r}$ instead, while satisfying the lower constraints. We remember r until reaching $(r, r, X \setminus Y, \texttt{true})$, where we know that the loop has successfully terminated because we crossed some repeated state $((q \in R))$. Therefore, we also add the transitions $((r, r, X \setminus Y, \texttt{true}), \texttt{true}, \varepsilon, \{z\}, f)$ where f is a new state which is the only final state of $\mathcal{A}(a, Y, m, M)$ and with $\ell(f) = a$ and $I(f) = \bigwedge_{y \in Y} y \leq M_y$ (with the convention that $y \leq \infty$ is $\texttt{true}$).

Third step. The resulting automaton $\mathcal{A}_1$ is the disjoint union of all automata $\mathcal{A}(a, 0)$ and $\mathcal{A}(a, Y, m, M)$. All finite words with finite duration accepted by $\mathcal{A}$ can be accepted by finite runs of $\mathcal{A}_1$ and $\mathcal{L}(\mathcal{A}_1) \subseteq \mathcal{L}(\mathcal{A})$. To obtain an equivalent SE-automaton $\mathcal{A}_2$ satisfying the condition $(\dagger)$, it remains to keep only the infinite runs of $\mathcal{A}$ which accept words with either an infinite duration or an infinite length. For this, we define an automaton $\mathcal{B}$ which goes to a repeated state whenever at least one time unit has elapsed or when a visible event is executed or when a *new* signal is emitted. The first two conditions are trivial to deal with. For the last one we need to keep track of the last signal that has been observed (a^d with $a \neq \tau$ or $d > 0$) so that we can enter a repeated state when a *new* signal is observed (b^δ with $b \neq a$ and $b \neq \tau$ or $\delta > 0$). The automaton $\mathcal{A}_2$ is obtained as a cartesian product of $\mathcal{A}_1$ and $\mathcal{B}$. Note that we cannot use an intersection operation since Theorem 1 is used in the proof of Theorem 2.

Fourth step. Finally, it remains to transform $\mathcal{A}_2$ into an automaton $\mathcal{A}'$ satisfying also condition $(\ddagger)$. The only problem comes from final states whose invariant is $\texttt{true}$. Words of the form wa^∞ accepted by finite runs ending in such states must now be accepted by infinite runs. The idea is to use again the new clock z to measure time intervals of length one. For each signal $a \in \Sigma_s$, we add a new repeated state r_a with label a and invariant $z \leq 1$. We also add loops $(r_a, z = 1, \varepsilon, \{z\}, r_a)$ and for each final state p with label a and invariant $\texttt{true}$ we add the transition $(p, \texttt{true}, \varepsilon, \{z\}, r_a)$ and p is not final anymore. This gives the automaton $\mathcal{A}'$ satisfying both conditions $(\dagger)$ and $(\ddagger)$ and concludes the construction. $\square$

Remark 2. If Zeno runs are not allowed (see Remark 1), condition $(\dagger)$ is true by definition of an accepted run. Hence the construction of an automaton satisfying Theorem 1 reduces to Fourth step above and is therefore much simpler.

In the same way, if ε-transitions are not allowed, an infinite run can accept only an infinite word and Theorem 1 reduces to condition $(\ddagger)$.

4 Intersection

We present in this section the main construction of this paper.

Theorem 2. *The class SEL_ε is closed under intersection.*

Note that one of the problems arising in the construction of the intersection comes from the fact that a word can be accepted in two different automata by a finite and an infinite run respectively. For instance, consider the two automata $\mathcal{A}_3$ and $\mathcal{A}_4$ in Figure 1. We have $\mathcal{L}(\mathcal{A}_3) = \{a^d \mid d \geq 1\}$ and $\mathcal{L}(\mathcal{A}_4) = \{a^d \mid d \leq 1\}$, so that $\mathcal{L}(\mathcal{A}_3) \cap \mathcal{L}(\mathcal{A}_4) =$

$\{a^1\}$. And this word a^1 is accepted in $\mathcal{A}_3$ by a finite run and in $\mathcal{A}_4$ by an infinite run. We will then use in a crucial way the normal form proposed by Theorem 1.

Before giving the general construction, let us point out some other difficulties. The treatment of visible events is easy and will be done like in the untimed case through a synchronized product. The case of signals is more tricky and needs more attention. Indeed, let us consider the following example:

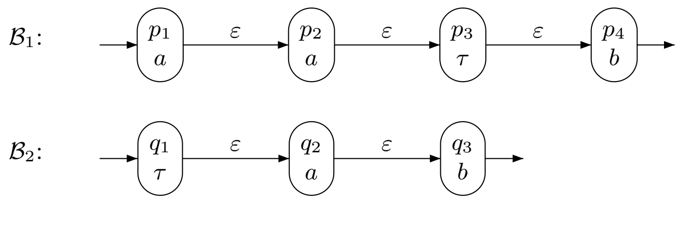


Fig. 4. Automata $\mathcal{B}_1$ and $\mathcal{B}_2$

If automaton $\mathcal{B}_1$ is in state p_1 and automaton $\mathcal{B}_2$ in state q_3, they can not compute anymore a SE-word which would be in the intersection of their languages. Indeed this word should have at the same time a factor a^d with $d \geq 0$ and a factor b^δ with $\delta \geq 0$, which is not possible since a and b are different from τ.

If automaton $\mathcal{B}_1$ is in state p_1 and automaton $\mathcal{B}_2$ in state q_2, they can both produce a signal a.

Now if automaton $\mathcal{B}_1$ is in state p_1 and automaton $\mathcal{B}_2$ in state q_1, whereas the labels of the two states are different, it is still possible to produce a word of the intersection. Indeed, it is sufficient to force $\mathcal{B}_2$ to leave immediately q_1 (i.e. to stay in q_1 0 time unit), $\mathcal{B}_2$ will thus produce a signal $\tau^0 \approx \varepsilon$ and thus not visible. This last case shows that we should allow in the intersection *asynchronous* moves where only one of the automata executes an ε-transition.

We now proceed to the construction of a SE-automaton accepting the intersection of the languages recognized by two automata

$$\mathcal{A}_j = (\Sigma_e, \Sigma_s, X_j, Q_j, Q_j^0, F_j, R_j, I_j, \ell_j, \Delta_j)$$

for $j = 1, 2$ on the same alphabet, satisfying the conditions (†) and (‡) of Theorem 1. We assume that Q_1 and Q_2 (respectively X_1 and X_2) are disjoint and, when no confusion can arise, we simply write ℓ for both labelling functions ℓ_1 and ℓ_2. We also assume that the automata $\mathcal{A}_1$ and $\mathcal{A}_2$ do not contain a trivial loop of the form $(p, \texttt{true}, \varepsilon, \emptyset, p)$.

We define the automaton $\mathcal{A} = (\Sigma_e, \Sigma_s, X, Q, Q^0, F, R, I, \ell, \Delta)$ designed to accept $\mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$ as follows.

- We set $X = X_1 \cup X_2 \cup \{z\}$, where z is a new clock used to control if the time elapsed in a state of $\mathcal{A}$ is zero or not.
- The set $Q \subseteq \Sigma_s \times Q_1 \times Q_2 \times \{0, 1, 2\}$ consists of all tuples (a, p, q, i) satisfying
  - $\ell_1(p), \ell_2(q) \in \{a, \tau\}$ and

  - $i = 1$ if and only if $\ell_1(p) = \ell_2(q) = a$.

  Note that, the conjunction of these two constraints implies that if the first component is $a = \tau$ then the last component must be $i = 1$.
- For $(a, p, q, i) \in Q$, we set
  - $\ell(a, p, q, i) = a$ and $I(a, p, q, i) = I_1(p) \wedge I_2(q)$ if $i = 1$ and
  - $\ell(a, p, q, i) = \tau$ and $I(a, p, q, i) = I_1(p) \wedge I_2(q) \wedge z \leq 0$ otherwise.

The intuitive idea behind the fourth component of the states of $\mathcal{A}$ is the following:

- Value 0 means that one of the automata is ready to perform some signal $a \neq \tau$ and is waiting for the other to reach a state labelled a with ε-transitions and instantaneously traversing τ-labelled states. If a synchronization is not possible on signal a, then the whole computation will not produce any accepting SE-word of the intersection,
- Value 1 means that the two automata emit the same signals,
- Value 2 means that the two automata were producing the same signals but have "lost" their synchronisation (due an ε-transition performed by one of them). As in the case of value 0, they will try to re-synchronize. But the whole computation can still progress even if this synchronization is not possible anymore.

The transition relation Δ consists of *synchronous* moves where both automata progress simultaneously and of *asynchronous* moves where one automaton is idle while the second one performs an ε-transition.

A synchronous move is not possible in a state of the form $(a, p, q, 0)$ since a synchronization is expected first. Consider two states (a, p, q, i) and (a', p', q', i') in Q with $i \neq 0$ and $i' \neq 2$. For any two transitions $\delta_1 = (p, g, b, \alpha, p') \in \Delta_1$ and $\delta_2 = (q, h, b, \beta, q') \in \Delta_2$ with $b \neq \varepsilon$ if $a = a'$, we add in Δ the synchronous transition

$$\delta = (a, p, q, i) \xrightarrow{g \wedge h, b, \alpha \cup \beta \cup \{z\}} (a', p', q', i')$$

and we set $\pi_j(\delta) = \delta_j$ for $j = 1, 2$.

Consider now a state $(a, p, q, i) \in Q$. For any transition $\delta_1 = (p, g, \varepsilon, \alpha, p') \in \Delta_1$ with $\ell_1(p') \in \{a, \tau\}$ we add in Δ the asynchronous transition

$$\delta = (a, p, q, i) \xrightarrow{g, \varepsilon, \alpha \cup \{z\}} (a, p', q, i')$$

where i' is updated so that (a, p', q, i') is a legal state and the choice between values 0 and 2 is made according to the abstract description in Fig. 5. Formally, if $a = \tau$ then $i' = 1$ is the only possibility. Now, if $a \neq \tau$ we have the following cases:

- $i' = 1$ if $\ell_1(p') = \ell_2(q) = a$: synchronization on a is active,
- $i' = 0$ if $i = 0$ and ($\ell_1(p') = \tau$ or $\ell_2(q) = \tau$): synchronization on a has not yet been achieved,
- $i' = 2$ if $i \neq 0$ and ($\ell_1(p') = \tau$ or $\ell_2(q) = \tau$): synchronization on a has been lost.

We set $\pi_1(\delta) = \delta_1$ and $\pi_2(\delta) = \varepsilon$. We proceed symmetrically for asynchronous transitions of $\mathcal{A}_2$.

In the construction above, the subset of states with first component a is designed to handle maximal blocks of the form a^d. This part of the intersection is represented for $a \neq \tau$ by the abstract automaton in Figure 5. Note that all the transitions are asynchronous ε-transitions which reset the clock z.

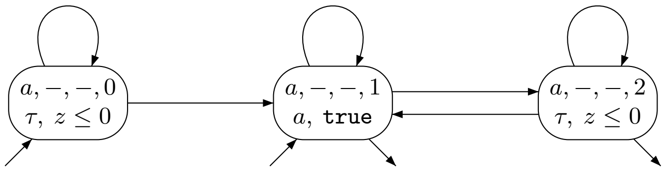


Fig. 5. Handling blocks a^d, for $a \neq \tau$

Since we have assumed that $\mathcal{A}_1$ and $\mathcal{A}_2$ do not contain a trivial loop of the form $(p, \mathtt{true}, \varepsilon, \emptyset, p)$, the projections $\pi_j(\delta)$ for $j = 1, 2$ are well-defined. Indeed, if $\delta = ((a, p_1, p_2, i), g, b, \alpha \cup \{z\}, (a', q_1, q_2, i')) \in \Delta$ then g is of the form $g_1 \wedge g_2$ where g_j involves clocks of $\mathcal{A}_j$ only. Hence, if we let $\alpha_j = \alpha \cap X_j$ we get $\pi_j(\delta) = \varepsilon$ if ($g_j = \mathtt{true}$ and $b = \varepsilon$ and $\alpha_j = \emptyset$ and $q_j = p_j$) and $\pi_j(\delta) = (p_j, g_j, b, \alpha_j, q_j)$ otherwise.

A path P of $\mathcal{A}$ can be seen as a sequence $\delta_1\delta_2\cdots$ of transitions in Δ. Clearly, the projection $\pi_j(P) = \pi_j(\delta_1)\pi_j(\delta_2)\cdots$ is a path of $\mathcal{A}_j$.

The initial and final states are defined by $Q^0 = Q \cap (\Sigma \times Q_1^0 \times Q_2^0 \times \{0,1\})$ and $F = Q \cap (\Sigma \times F_1 \times F_2 \times \{1,2\})$. We will not define the repeated states R explicitely. Instead, an infinite run P of $\mathcal{A}$ will be accepting if and only if each projection $\pi_j(P)$ is infinite and accepting in $\mathcal{A}_j$. It is well-known how to turn this intersection of Büchi conditions into a Büchi condition using some additional information [13]. For the sake of simplicity, we skip this easy construction here.

Note that, since conditions (†) and (‡) hold for $\mathcal{A}_1$ and $\mathcal{A}_2$, they also hold for $\mathcal{A}$.

Examples. The next easy examples illustrate the construction and the usefulness of the additional components a and i. Consider the two automata $\mathcal{B}_1$ and $\mathcal{B}_2$ in Figure 4, which have only finite runs and thus satisfy condition (†). We could easily ensure that they also satisfy the condition (‡) by adding invariants in the final states, which is omitted for simplicity. Recall that in our model, the signal τ^0 is equivalent to the empty word ε. Consequently, the language accepted by $\mathcal{B}_1$ is $\{a^{d_1}\tau^{d_2}b^{d_3} \mid d_1, d_3 \geq 0, d_2 > 0\} \cup \{a^{d_1}b^{d_3} \mid d_1, d_3 \geq 0\}$ while $\mathcal{B}_2$ accepts $\{\tau^{d_1}a^{d_2}b^{d_3} \mid d_1 > 0, d_2, d_3 \geq 0\} \cup \{a^{d_2}b^{d_3} \mid d_2, d_3 \geq 0\}$. Hence $L(\mathcal{B}_1) \cap L(\mathcal{B}_2) = \{a^{d_1}b^{d_3} \mid d_1, d_3 \geq 0\}$. A word is in the intersection (if and) only if the states labeled by τ are crossed instantaneously by an accepting run.

The automaton $\mathcal{B}$ constructed for the intersection is represented in Figure 6. All transitions are ε-transitions which reset the clock z.

We now modify automata $\mathcal{B}_1$ and $\mathcal{B}_2$ into $\mathcal{B}_1'$ and $\mathcal{B}_2'$ by adding loops, as represented in Figure 7. In this case, the words in the intersection may contain factors of the form τ^d, as can be seen on the resulting automaton $\mathcal{B}'$ in Figure 8.

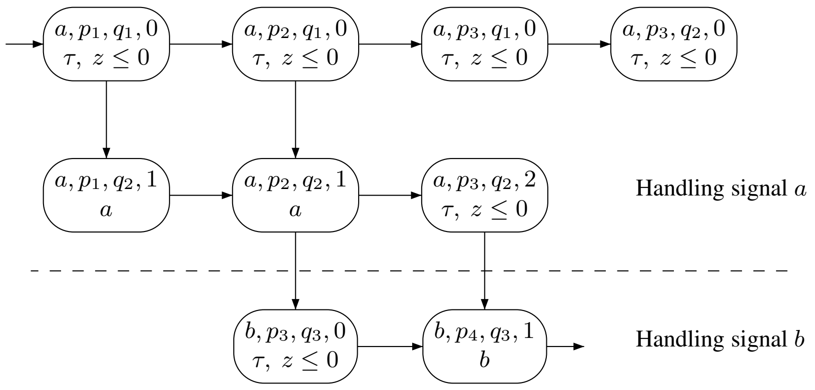

$a, p_1, q_1, 0$ $\tau,\ z \le 0$
$a, p_2, q_1, 0$ $\tau,\ z \le 0$
$a, p_3, q_1, 0$ $\tau,\ z \le 0$
$a, p_3, q_2, 0$ $\tau,\ z \le 0$
$a, p_1, q_2, 1$ a
$a, p_2, q_2, 1$ a
$a, p_3, q_2, 2$ $\tau,\ z \le 0$
Handling signal a
$b, p_3, q_3, 0$ $\tau,\ z \le 0$
$b, p_4, q_3, 1$ b
Handling signal b

Fig. 6. Resulting automaton $\mathcal{B}$

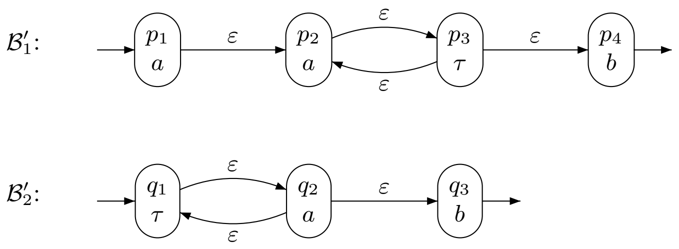


Fig. 7. Automata $\mathcal{B}'_1$ and $\mathcal{B}'_2$

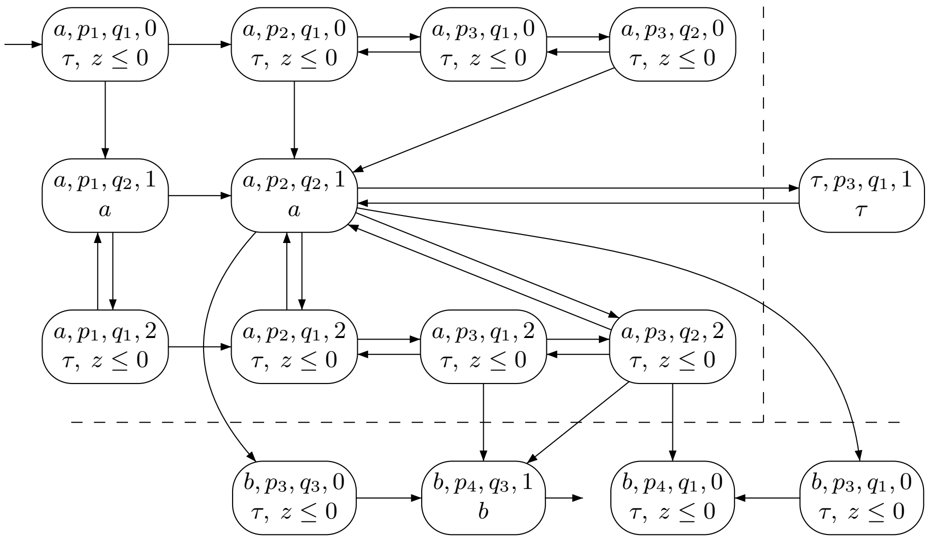

$a, p_1, q_1, 0$ $\tau,\ z \le 0$
$a, p_2, q_1, 0$ $\tau,\ z \le 0$
$a, p_3, q_1, 0$ $\tau,\ z \le 0$
$a, p_3, q_2, 0$ $\tau,\ z \le 0$
$a, p_1, q_2, 1$ a
$a, p_2, q_2, 1$ a
$\tau, p_3, q_1, 1$ τ
$a, p_1, q_1, 2$ $\tau,\ z \le 0$
$a, p_2, q_1, 2$ $\tau,\ z \le 0$
$a, p_3, q_1, 2$ $\tau,\ z \le 0$
$a, p_3, q_2, 2$ $\tau,\ z \le 0$
$b, p_3, q_3, 0$ $\tau,\ z \le 0$
$b, p_4, q_3, 1$ b
$b, p_4, q_1, 0$ $\tau,\ z \le 0$
$b, p_3, q_1, 0$ $\tau,\ z \le 0$

Fig. 8. Resulting automaton $\mathcal{B}'$

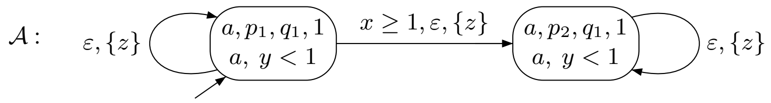


Fig. 9. Resulting automaton $\mathcal{A}$

We insist that the construction relies on condition $(\dagger)$. Consider again the two automata $\mathcal{A}_3$ and $\mathcal{A}_4$ in Figure 1. Condition $(\dagger)$ does not hold for $\mathcal{A}_4$, because a^1 is accepted by an infinite run. We have $\mathcal{L}(\mathcal{A}_3) \cap \mathcal{L}(\mathcal{A}_4) = \{a^1\}$. Note that the construction given in the proof of Theorem 2 would fail in this case since it would yield the automaton $\mathcal{A}$ in Figure 9. We have $\mathcal{L}(\mathcal{A}) = \emptyset$ for two reasons. First $\mathcal{A}$ contains no final state and it admits also no accepting infinite run since the first projection of the run cannot be infinite. This is not actually the main problem. We could have defined $\mathcal{A}$ so that a path P is accepting if and only if both projections $\pi_1(P)$ and $\pi_2(P)$ are accepting (finite or not). Then the argument above does not apply anymore. Still we would have $\mathcal{L}(\mathcal{A}) = \emptyset$ due to the invariant $y < 1$ and the guard $x \geq 1$.

Remark 3. If we consider signal-event automata where ε-transitions are not allowed, the treament of the intersection becomes much simpler. Indeed, the intersection of two SE-automata without ε-transitions can be done in a classical way, i.e., as a product of automata and a suitable treatment of Büchi conditions.

5 Conclusion

We proposed in this paper a construction for the intersection of two signal-event automata in the most general framework, working on finite and infinite signal-event words and taking into account signal stuttering, unobservability of zero-duration τ-signals and Zeno runs.

While constructions were proposed in the literature for important particular cases, it is the first time, up to our knowledge, that the general case is treated. There has been in the area of timed automata some examples of subtly erroneous constructions (e.g. with respect to forward analysis [8]) which should convince us of the importance to publish complete and proved constructions.

Moreover, it turns out that the closure of signal-event automata under intersection, and the normal form achieved in Theorem 1, are crucial to study the closure of SE-languages recognized by such automata under timed substitutions [6].

References

1. R. Alur and D.L. Dill. Automata for modeling real-time systems. In *Proceedings of ICALP'90*, number 443 in LNCS, pages 322–335. Springer, 1990.
2. R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.
3. E. Asarin, P. Caspi, and O. Maler. A Kleene theorem for timed automata. In *Proceedings of LICS'97*, pages 160–171. IEEE Comp. Soc. Press, 1997.

4. E. Asarin, P. Caspi, and O. Maler. Timed regular expressions. *Journal of the ACM*, 49(2):172–206, 2002.
5. B. Bérard, V. Diekert, P. Gastin, and A. Petit. Characterization of the expressive power of silent transitions in timed automata. *Fundamenta Informaticae*, 36:145–182, 1998.
6. B. Bérard, P. Gastin, and A. Petit. Refinements and abstractions of signal-event (timed) languages. In *Proceedings of FORMATS'06*, number 4202 in LNCS. Springer, 2006.
7. B. Bérard, P. Gastin, and A. Petit. Timed substitutions for regular signal-event languages. Research Report LSV-06-04, Laboratoire Spécification et Vérification, ENS Cachan, France, February 2006.
8. P. Bouyer. Forward analysis of updatable timed automata. *Formal Methods in System Design*, 24(3):281–320, May 2004.
9. P.J.L. Cuijpers, M.A. Reniers, and A.G. Engels. Beyond zeno-behaviour. Technical Report CSR 01-04, Department of Computing Science, University of Technology, Eindhoven, 2001.
10. C. Dima. Real-Time Automata and the Kleene Algebra of Sets of Real Numbers. In *Proceedings of STACS'2000*, number 1770 in LNCS, pages 279–289. Springer, 2000.
11. J. Durand-Lose. A Kleene theorem for splitable signals. *Information Processing Letters*, 89:237–245, 2004.
12. M.R. Hansen, P.K. Pandya, and C. Zhou. Finite divergence. *Theoretical Computer Science*, 138:113–139, 1995.
13. D. Perrin and J.-E. Pin. *Infinite words*. Elsevier, 2004.

Refinements and Abstractions of Signal-Event (Timed) Languages

Béatrice Bérard[1], Paul Gastin[2], and Antoine Petit[2]

[1] LAMSADE, Université Paris Dauphine & CNRS,
Place du Maréchal de Lattre de Tassigny, F-75775 Paris Cedex 16, France
[2] LSV, ENS de Cachan & CNRS,
61 av. du Président Wilson, F-94235 Cachan Cedex, France

Abstract. In the classical framework of formal languages, a refinement operation is modeled by a substitution and an abstraction by an inverse substitution. These mechanisms have been widely studied, because they describe a change in the specification level, from an abstract view to a more concrete one, or conversely. For timed systems, there is up to now no uniform notion of substitutions. In this paper, we study the timed substitutions in the general framework of signal-event languages, where both signals and events are taken into account. We prove that regular signal-event languages are closed under substitutions and inverse substitutions.

1 Introduction

Refinements and abstractions. Operations of refinements and abstractions are essential tools for the design and the study of systems, real-time or not. They allow to consider a given system at different levels of abstractions. For instance, some procedure or function can simply be viewed at some abstract level as a single action, and can be later expanded into all its possible behaviours at some more concrete level. Or conversely, a set of behaviours are merged together and replaced by a single action, in order to obtain a more abstract description.

These operations can be formally modeled by substitution and inverse substitution respectively. Therefore, substitutions have been extensively studied in the untimed framework, with the underlying idea that interesting classes of languages have to be closed under these operations.

Timed languages. In order to accept timed words, the model of timed automata was first proposed in [1,2]. It has been widely studied for the last fifteen years and successfully applied to industrial cases. For this model, an observation, called a time-event word, may be viewed as an alternating sequence of waiting times and instantaneous actions. A timed automaton is a finite automaton extended with variables called clocks, designed to recognize time-event words: time elapses while the control stays in a given node and an event is observed when a discrete transition occurs.

Another model was introduced by [3], and further studied in [10,4,11] with the aim of describing hardware systems. In this case, an observation is a signal word, i.e., a sequence of factors a^d, where a is a signal and d is its duration. The original model of timed automata was then modified to fit this setting: a signal is emitted while the automaton stays in some state and no event is produced when a discrete transition is fired. In this framework, when a transition occurs between two states with the same signal a, we obtain a^{d_1} followed by a^{d_2}, which are merged into $a^{d_1+d_2}$. This phenomenon is called stuttering.

It was noticed in [4] that both approaches are complementary and can be combined in an algebraic formalism to obtain the so-called signal-event monoid. Timed automata can be easily adapted to take both signals and events into account, thus yielding signal-event automata: states emit signals and transitions produce events.

We consider in this paper both finite and infinite behaviors of signal-event automata and we also include unobservable events (ε-transitions) and hidden signals (τ-labeled states). It turns our that these features are very useful, for instance for handling abstractions. They also allow us to get as special cases the initial models of timed automata and signal automata.

Timed substitutions. Timed substitutions were studied in [8] for regular transfinite time-event languages. In [8], although no signal appear explicitely, actions are handled in a way similar (but not identical) to signals, *without* stuttering. Here we restrict the study of substitutions to finite and ω-sequences but we do handle signal stuttering which is a major difficulty.

Our contribution. The aim of this paper is to study the closure by substitutions and inverse substitutions of the families SEL_ε and SEL of languages accepted by signal-event automata, with or without ε-transitions. We prove that the class SEL_ε is closed under arbitrary substitutions and under arbitrary inverse substitutions. These closure properties are not verified by the class SEL in general. Nevertheless, we show that SEL is closed under inverse substitutions acting on events only, i.e., leaving signals unchanged, and we give a sufficient condition for its closure under substitutions. These results again show the robustness of the class SEL_ε, which is in favour of signal-event automata including ε-transitions.

Outline of the paper. We first give in Section 2 precise definitions of finite and infinite signal-event languages, with the corresponding notion of signal-event automata and we recall some technical results on signal-event automata that will be crucial for further proofs. In Section 3, we define timed substitutions which are duration preserving mappings. We then study in Section 4 the closures of the classes SEL and SEL_ε under reconizable substitutions and their inverses.

For lack of space, this paper does not contain the proofs of the correctness of the different automata constructions we proposed. These proofs are available in the technical report [6].

2 Signal-Event Words and Signal-Event Automata

Let Z be any set. We write Z^* (respectively Z^ω) the set of finite (respectively infinite) sequences of elements in Z, with ε for the empty sequence, and $Z^\infty = Z^* \cup Z^\omega$ the set of all sequences of elements in Z. The set Z^∞ is equipped with the usual partial concatenation defined from $Z^* \times Z^\infty$ to Z^∞.

Throughout this paper, we consider a time domain $\mathbb{T}$ which can be either the set $\mathbb{N}$ of natural numbers, the set $\mathbb{Q}_+$ of non-negative rational numbers or the set $\mathbb{R}_+$ of non-negative real numbers and we set $\overline{\mathbb{T}} = \mathbb{T} \cup \{\infty\}$.

2.1 Signal-Event Words

We now describe the most general class of systems where both piecewise-constant signals and discrete events can occur, based on the signal-event monoid defined in [4]. We consider two finite alphabets Σ_e and Σ_s, with $\Sigma = \Sigma_e \cup (\Sigma_s \times \overline{\mathbb{T}})$: an element in Σ_e is the label of an instantaneous event, while a pair $(a, d) \in \Sigma_s \times \overline{\mathbb{T}}$, written a^d, associates a duration d with a signal a. Moreover, Σ_s includes the special symbol τ for an internal (or hidden) signal, the purpose of which is to represent a situation where no signal can be observed.

Intuitively, *signal-event words* (SE-words for short and sometimes called *timed words*) correspond to sequences obtained from Σ^∞ by merging consecutive identical signals and removing internal τ-signals with duration 0. But note that signals different from τ may have a null duration.

Formally, the partial monoid of signal-event words is the quotient $\Sigma^\infty / \approx$ where $\approx$ is the congruence (with respect to the partial concatenation on Σ^∞) generated by

$$\begin{cases} \tau^0 \approx \varepsilon & \text{and} \\ \prod_{i\in I} a^{d_i} \approx \prod_{j\in J} a^{d'_j} & \text{if } \sum_{i\in I} d_i = \sum_{j\in J} d'_j \end{cases}$$

where the index sets I and J above may be infinite. The partial monoid $\Sigma^\infty / \approx$ will be denoted $SE(\Sigma, \mathbb{T})$ or simply $SE(\Sigma)$ or SE when there is no ambiguity. We write a^∞ for the equivalence class of any sequence of the form $\prod_{i\geq 1} a^{d_i}$, where $\sum_{i\geq 1} d_i = \infty$. Note that for two words of the forms ua^d and $a^{d'}v$ with $d < \infty$, the concatenation is $ua^{d+d'}v$.

A finite or infinite sequence in $\Sigma^\infty \cup \Sigma^* \cdot (\Sigma_s \times \{\infty\})$ which does not contain τ^0 and such that two consecutive signals are distinct is said to be in *normal form* (NF). SE-words are often identified with sequences in normal form. A SE-word is *finite* if its normal form is a finite sequence (even if it ends with a^∞).

A duration can be associated with each element of Σ by: $\|a\| = 0$ if $a \in \Sigma_e$ and $\|a^d\| = d$ if $a \in \Sigma_s$ and $d \in \overline{\mathbb{T}}$, so that the duration of a sequence $w = s_1 s_2 \cdots$ in Σ^∞ is $\|w\| = \sum_{i\geq 1} \|s_i\| \in \overline{\mathbb{T}}$. Note that the duration restricted to finite SE-words with finite durations is a morphism from Σ^* into $(\mathbb{T}, +)$. A *Zeno* word is a SE-word with finite duration and whose normal form is infinite. A *signal-event language* (or *timed language*) is a set of SE-words.

Example 1. Let $\Sigma_e = \{f, g\}$ and $\Sigma_s = \{a, b\}$. The SE-word $w = a^3 f f g \tau^{4.5} a^1 b^5$ can be viewed as the following sequence of observations: first, the signal a during 3 time units, then a sequence of three instantaneous events ffg, then some unobservable signal during 4.5 time units, again the signal a during 1 time unit and then the signal b during 5 time units. The total duration of w is 13.5. For infinite SE-words, we have for instance: $a^3 g f a^1 \prod_{i\geq 1} a^2 \approx a^1 a^2 g f \prod_{i\geq 1} a^4$ and the normal form is written $a^3 g f a^\infty$. Note also that an infinite timed sequence in Σ^ω may be a finite SE-word with finite duration: $\prod_{i\geq 0} a^{1/2^i} \approx a^2$.

2.2 Signal-Event (Timed) Automata

Our model of *signal-event automata* (also called *timed automata* in the sequel) is a variant of the basic models proposed in the literature, integrating both instantaneous and durational semantics: signals are associated with the control states, while instantaneous events occur when the system switches between two states.

Clocks and guards. Let X be a set of variables with values in $\mathbb{T}$, called clocks. The set $\mathcal{C}(X)$ of guards or clock constraints over X consists of conjunctions of atomic formulas $x \bowtie c$, for a clock x, a constant $c \in \mathbb{T}$ and a binary operator $\bowtie$ in $\{<, \leq, =, \geq, >\}$.

A clock valuation $v : X \to \mathbb{T}$ is a mapping that assigns to each clock x a time value $v(x)$. The set of all clock valuations is $\mathbb{T}^X$. We write $v \models g$ when the clock valuation v satisfies the clock constraint g. If t is an element of $\mathbb{T}$ and α a subset of X, the valuations $v + t$ and $v[\alpha]$ are defined respectively by $(v + t)(x) = v(x) + t$, for each clock x in X and $(v[\alpha])(x) = 0$ if $x \in \alpha$, and $(v[\alpha])(x) = v(x)$ otherwise.

Signal-event (timed) automata. A Büchi signal-event automaton over the time domain $\mathbb{T}$ is a tuple $\mathcal{A} = (\Sigma_e, \Sigma_s, X, Q, Q_0, F, R, I, \ell, \Delta)$, where Σ_e and Σ_s are alphabets of events and signals, X is a finite set of $\mathbb{T}$-valued clocks, Q is a finite set of control states, $Q_0 \subseteq Q$ is a subset of initial states, $F \subseteq Q$ is a subset of final states and $R \subseteq Q$ corresponds to a Büchi acceptance condition. The mapping $I : Q \to \mathcal{C}(X)$ associates with a state $q \in Q$ an *invariant* $I(q)$ being a conjunction of constraints of the form $x \bowtie c$, with $\bowtie \in \{<, \leq\}$, and $\ell : Q \to \Sigma_s$ associates a signal with each state.

The set of transitions is $\Delta \subseteq Q \times \mathcal{C}(X) \times \Sigma_e \cup \{\varepsilon\} \times \mathcal{P}(X) \times Q$. A transition, also written $q \xrightarrow{g,a,\alpha} q'$, is labeled by a guard g, an instantaneous event in Σ_e or the unobservable event ε, and the subset α of clocks to be reset. When $a = \varepsilon$, it is called an ε-transition or a silent transition.

First examples of signal-event automata are given in Figure 1 (where double-circled nodes correspond to Büchi repeated states). The semantics of *SE*-automata will be given below. But intuitively,

- A *SE*-word is accepted by $\mathcal{A}_1$ if it is of the form $a^{d_1} b^{d_2}$ with $d_1 \geq 1$.
- A *SE*-word is accepted by $\mathcal{A}_2$ if it is of the form $a^{d_1} c a^{d_2} c \ldots$ with $d_i < 1$ for any i.
- Since the concatenation merges consecutive identical signals ($a^{d_1} a^{d_2} = a^{d_1 + d_2}$), the language accepted by $\mathcal{A}_3$ consists of the signal a emitted for a duration $d \geq 1$.
- $\mathcal{A}_4$ accepts the signal a emitted for a duration $d \leq 1$ (note that a^1 is accepted by an infinite run with successive durations $\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \ldots$ for instance).

Semantics. In order to define the semantics of *SE*-automata, we recall the notions of path and timed run through a path. A path in $\mathcal{A}$ is a finite or infinite sequence of consecutive transitions:

$$P = q_0 \xrightarrow{g_1, a_1, \alpha_1} q_1 \xrightarrow{g_2, a_2, \alpha_2} q_2 \ldots, \text{ where } (q_{i-1}, g_i, a_i, \alpha_i, q_i) \in \Delta,\ \forall i > 0$$

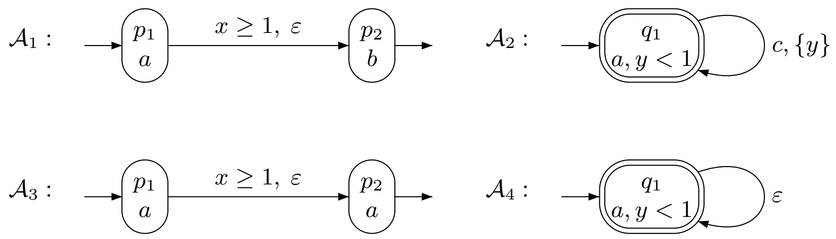


Fig. 1. Some signal automata

The path is said to be *accepting* if it starts in an initial state ($q_0 \in Q_0$) and *either* it is finite and ends in a final state, *or* it is infinite and visits infinitely often a *repeated* state $q \in R$. A *run* of the automaton through the path P is a sequence of the form:

$$\langle q_0, v_0\rangle \xrightarrow{d_0} \langle q_0, v_0 + d_0\rangle \xrightarrow{a_1} \langle q_1, v_1\rangle \xrightarrow{d_1} \langle q_1, v_1 + d_1\rangle \xrightarrow{a_2} \langle q_2, v_2\rangle \ldots$$

where

- $d_i \in \mathbb{T}$ for $i \geq 0$ and if P is finite with n transitions then the last step of the run must be $\langle q_n, v_n\rangle \xrightarrow{d_n} \langle q_n, v_n + d_n\rangle$, with $d_n \in \overline{\mathbb{T}}$,
- $(v_i)_{i \geq 0}$ are clock valuations such that $v_0(x) = 0$ for all $x \in X$, and for each $i \geq 0$, we have

$$\begin{cases} v_i + d \models I(q_i), & \forall d \in [0, d_i] \\ v_i + d_i \models g_{i+1} \\ v_{i+1} = (v_i + d_i)[\alpha_{i+1}] \end{cases}$$

  Note that if d_i is finite, the condition about invariant $I(q_i)$ can be replaced simply by $v_i + d_i \models I(q_i)$.

The signal-event (timed) word generated by this run is simply (the equivalence class of) $\ell(q_0)^{d_0} a_1 \ell(q_1)^{d_1} a_2 \ell(q_2)^{d_2} \ldots$. The signal-event (timed) language accepted by $\mathcal{A}$ over the time domain $\mathbb{T}$ and the alphabet Σ, written $\mathcal{L}(\mathcal{A})$, is the set of SE-words generated by (finite or infinite) accepting runs of $\mathcal{A}$. Two automata $\mathcal{A}$ and $\mathcal{B}$ are *equivalent* if $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$.

The set of all signal-event (timed) automata is denoted by SEA_ε whereas SEA is the set of all signal-event automata using transitions with observable events only (i.e. with labels in Σ_e instead of $\Sigma_e \cup \{\varepsilon\}$). The family of signal-event (timed) languages generated by some signal-event automaton in SEA_ε (respectively SEA) is denoted by SEL_ε (respectively SEL).

Remark 1. A *Zeno run* is an infinite run for which the time sequence defined by $t_i = \sum_{j \leq i} d_j$ for $i \geq 0$, is convergent (keeping the notations just above).

We did not include the non Zeno condition for runs, requiring that each infinite accepting run has an infinite duration. Thus, Zeno runs accepting finite words with finite duration may occur. Although Zeno behaviours have been studied (see for instance [12,9]), it has been argued that excluding Zeno runs directly from the semantics of timed automata is a realistic assumption, since they do not appear in "real" systems. However, the semantics of timed automata used by model-checking tools like UPPAAL do include Zeno runs while performing forward reachability analysis (this can be easily checked with an example). Hence, we think that the theory should include Zeno runs as well.

We state now two properties that will be used to prove the main results of this paper. The constructions related to the following two propositions can be found in [7]. It turns out that any SE-automaton can be transformed into an equivalent one, in which finite accepting runs correspond exactly to finite words with finite duration. Note that the transformation removes a particular type of Zeno runs, those which contain ultimately only ε-transitions and a single signal. But it keeps Zeno runs corresponding to infinite words of finite duration.

Proposition 1. *Let $\mathcal{A}$ be a SE-automaton. We can effectively construct an equivalent SE-automaton $\mathcal{A}'$ such that:*

($\dagger$) *no infinite run of $\mathcal{A}'$ accepts a finite word with finite duration, and*
($\ddagger$) *no finite run of $\mathcal{A}'$ accepts a word with infinite duration.*

The previous result is interesting in itself for implementation issues: when a finite word with finite duration is accepted by an infinite run, we build instead a finite accepting run for this word. Furthermore, conditions ($\dagger$) and ($\ddagger$) are crucial to prove the following closure property and are also used in the proof of Theorem 2.

Proposition 2. *The class SEL_ε is closed under intersection.*

If we consider signal-event automata where ε-transitions are not allowed, the treament of the intersection becomes much simpler. Indeed, the intersection of two SE-automata without ε-transitions can be done in a classical way, i.e. as a product of automata and a suitable treatment of the Büchi conditions. But in the general case, the construction of an automaton recognizing the intersection is much more tricky. It can be found in [7], together with references on related works.

3 Signal-Event (Timed) Substitutions

Recall that substitutions are a suitable model for refinements. In the untimed framework, the image of each letter $a \in \Sigma$ is a given language L_a over another alphabet Σ' and a substitution is a morphism extending this mapping.

Dealing with timed words requires to preserve durations. Therefore, an instantaneous event must be replaced by SE-words with null duration, while a signal a with duration d must be replaced by SE-words of the same duration d. Formally, the new alphabet is also of the form $\Sigma' = \Sigma'_e \cup (\Sigma'_s \times \mathbb{T})$ and a substitution σ is defined by a family of SE-languages $(L_a)_{a \in \Sigma_e \cup \Sigma_s}$ such that:

- $L_a \subseteq (\Sigma'_e \cup (\Sigma'_s \times \{0\}))^*$ if $a \in \Sigma_e$,
- if $a \in \Sigma_s \setminus \{\tau\}$, then L_a is a SE-language of *non Zeno* SE-words over Σ'.

Using Zeno-words in substitutions may give rise to transfinite sequences, therefore we have excluded them from the languages L_a.

Moreover, we assume that the internal signal τ is never modified, so that we always have $L_\tau = \{\tau\} \times \overline{\mathbb{T}}$. Then for each $a \in \Sigma_e$, $\sigma(a) = L_a$ and for each $a^d \in \Sigma_s \times \overline{\mathbb{T}}$, $\sigma(a^d) = \{w \in L_a \mid \|w\| = d\}$. A substitution is thus a duration preserving mapping.

For a SE-word $v = v_1 v_2 \cdots$ in normal form over Σ, $\sigma(v)$ is the set of SE-words obtained from $\sigma(v_1)\sigma(v_2)\cdots$ by merging consecutive identical signals. Note that $w \in \sigma(v)$ can be written $w_1 w_2 \cdots$ with $w_i \in \sigma(v_i)$ for each $i \geq 1$. However this decomposition of w may not be in normal form. Finally, for a timed language L over Σ, we set $\sigma(L) = \cup_{v \in L} \sigma(v)$.

Example 2

- Choosing $L_f = \{f\}$ for $f \in \Sigma_e$ and $L_a = \{\tau^0\} \cup \{a\} \times (\overline{\mathbb{T}} \setminus \{0\})$ for $a \in \Sigma_s$ leads to a substitution that cancel all signals with a null duration.
- Hiding some signal a is simulated by a substitution where $L_a = \{\tau\} \times \overline{\mathbb{T}}$.

Timed substitutions and morphisms. It should be noticed that, while substitutions are morphisms in the untimed framework, this is not the case with our definition. For instance, assume that a is a signal such that $L_a = \{b^2\}$, then $\sigma(a^1) = \emptyset$ and $\sigma(a^2) = \{b^2\} \neq \sigma(a^1)\sigma(a^1)$.

We call *m-substitution* a SE-substitution which is a morphism with respect to the partial concatenation. We have the following characterization:

Proposition 3. *Let σ be a SE-substitution, given by a family $(L_a)_{a \in \Sigma_e \cup \Sigma_s}$. Then, σ is a morphism if and only if for each signal $a \in \Sigma_s$ we have*

1. *L_a is closed under concatenation: for all $u, v \in L_a$ with $\|u\| < \infty$, we have $uv \in L_a$,*
2. *L_a is closed under decomposition: for each $v \in L_a$ with $\|v\| = d$, for all $d_1 \in \mathbb{T}$, $d_2 \in \overline{\mathbb{T}}$ such that $d = d_1 + d_2$, there exist $v_i \in L_a$ with $\|v_i\| = d_i$ such that $v = v_1 v_2$.*

This proposition is easy to prove and shows that rather restrictive conditions should be added on the languages L_a to obtain morphisms.

The notion of abstraction is fundamental in the study of systems, and in particular in their verification. It consists in replacing a set of behaviors with a single action in order to obtain a smaller system, simpler to study and to understand. The abstraction operation is thus the inverse of refinement. As in the untimed case, inverse substitutions provide a suitable model for abstractions in our framework.

For a substitution σ, the inverse substitution σ^{-1} is the operation defined for a language $L' \subseteq SE(\Sigma')$ by $\sigma^{-1}(L') = \{v \in SE(\Sigma) \mid \sigma(v) \cap L' \neq \emptyset\}$.

4 Recognizable Substitutions

We now focus on recognizable substitutions, whose associated languages are defined by SE-automata. Formally, a substitution σ defined by a family of languages $(L_a)_{a \in \Sigma_e \cup \Sigma_s}$

is a SEL_ε-substitution (a SEL-substitution resp.) if $L_a \in SEL_\varepsilon$ (resp. $L_a \in SEL$) for each $a \in \Sigma_e \cup \Sigma_s$.

The aim of this paper is precisely to investigate the closure of the two classes of SE-languages SEL and SEL_ε under recognizable substitutions and their inverse.

We first consider the special cases of renaming and event-hiding. Formally, a renaming is simply a substitution such that, for each $f \in \Sigma_e$, $L_f = \{g\}$ for some $g \in \Sigma'_e$, and for each $a \in \Sigma_s \setminus \{\tau\}$, $L_a = \{b\} \times \overline{\mathbb{T}}$ for some $b \in \Sigma'_s$. An event-hiding is a substitution such that $L_f = \{f\}$ or $L_f = \{\varepsilon\}$ for $f \in \Sigma_e$ and $L_a = \{a\} \times \overline{\mathbb{T}}$ for $a \in \Sigma_s$. We have:

Proposition 4

1. *The classes SEL and SEL_ε are closed under renaming,*
2. *The class SEL_ε is closed under event-hiding whereas SEL is not,*

Proof Point 1 is straightforward. For the second point, the closure of SEL_ε under event-hiding was already noticed in [1] and the result easily extends to our framework: in order to hide the event a, one replaces a-labelled transitions by ε-transitions.

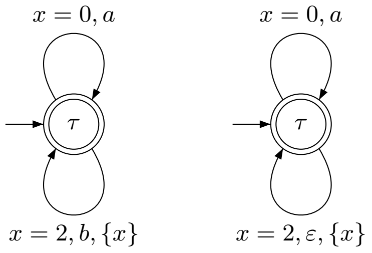
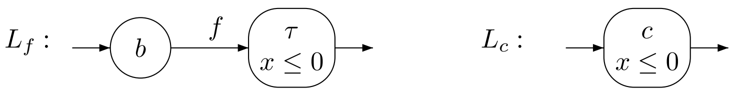


Fig. 2. Timed automata for L_1 and $h(L_1)$

For the class SEL, recall from [5] that the language $Even = ((\tau^2)^* a)^\infty$ cannot be accepted by a timed automaton without ε-transitions. Consider the language $L_1 = ((\tau^2 b)^* a)^\omega \cup ((\tau^2 b)^* a)^* (\tau^2 b)^\omega$ accepted by the automaton on the left of Figure 2, with no final state, where the unique state is labeled τ and is both initial and repeated. Hiding the b's in L_1 yields the language $h(L_1)$ accepted by the automaton in SEA_ε on the right of Figure 2. Since $h(L_1)$ is the analogous of $Even$ in our framework, it is not in the class SEL. □

The class SEL is not closed under SEL-substitution. Indeed, take $L_f = \{b^0 f\}$ and $L_c = \{c^0\}$ which are accepted by the automata below and are therefore both in SEL. Consider also $L = \{c^0 f\} \in SEL$. Then $\sigma(L) = \{c^0 b^0 f\}$ is not in SEL since it contains a SE-word with two consecutive distinct signals. The next result gives a sufficient condition on a substitution for the closure property of SEL to hold.



Theorem 1. *Let L be a language in SEL and σ a SEL-substitution such that for each $f \in \Sigma_e$ the language L_f contains only SE-words starting and ending with (instantaneous) events from Σ'_e. Then $\sigma(L)$ belongs to SEL.*

Proof. We prove the theorem for a substitution acting only on events or acting only on signals. The general case is obtained using compositions of such elementary substitutions and of renaming.

We first show how to handle substitution of events. So let σ be a SEL-substitution satisfying the condition of the theorem and which is defined by a family $(L_a)_{a\in\Sigma_e\cup\Sigma_s}$ of SEL-languages. Since σ acts only on events, we have $L_a = \{a\} \times \overline{\mathbb{T}}$ for all $a \in \Sigma_s$. For $f \in \Sigma_e$, the language L_f is accepted by some SEA-automaton $\mathcal{A}_f$ with only one clock x_f which is never tested or reset, with no repeated states, where all the guards are $\texttt{true}$ and where all final states carry the invariant $x_f \le 0$ in order to ensure that each accepted SE-word has a null duration. Moreover, from the hypothesis on σ, we can assume that all initial and final states are labelled τ.

Now, let $L \subseteq SE(\Sigma)$ be accepted by a SEA-automaton $\mathcal{A}$. We build from $\mathcal{A}$ a SEA-automaton $\mathcal{A}'$ accepting $\sigma(L)$ as follows. For each state q, we consider a new copy $\mathcal{A}_f^q$ of $\mathcal{A}_f$ and we replace any transition (p, g, f, α, q) by the following set of transitions:

- for each transition $(q_0, \texttt{true}, b, \emptyset, q_1)$ in $\mathcal{A}_f^q$ with q_0 initial, we add the transition $(p, g, b, \alpha \cup \{x_f\}, q_1)$ to $\mathcal{A}'$,
- for each transition $(q_1, \texttt{true}, b, \emptyset, q_2)$ in $\mathcal{A}_f^q$ with q_2 final, we add the transition $(q_1, x_f = 0, b, \emptyset, q)$ to $\mathcal{A}'$,
- for each transition $(q_0, \texttt{true}, b, \emptyset, q_2)$ in $\mathcal{A}_f^q$ with q_0 initial and q_2 final, we add the transition (p, g, b, α, q) to $\mathcal{A}'$,
- all states and transitions of $\mathcal{A}_f^q$ are kept unchanged in $\mathcal{A}'$.

Again, the clock operations on x_f ensure an instantaneous traversal of $\mathcal{A}_f^q$. The initial, final and repeated states remain those of $\mathcal{A}$. Clearly $\mathcal{A}'$ is a SEA-automaton and we can show that it accepts $\sigma(L)$.

We now handle substitution of signals. Let σ be a SEL-substitution defined by a family $(L_a)_{a\in\Sigma_e\cup\Sigma_s}$ of SEL-languages. We assume that σ acts only on signals, i.e., $L_f = \{f\}$ for each $f \in \Sigma_e$. For $a \in \Sigma_s$, the language L_a is accepted by some SEA-automaton $\mathcal{A}_a = (\Sigma'_e, \Sigma'_s, X_a, Q_a, Q_a^0, F_a, R_a, I_a, \ell_a, \Delta_a)$. We assume that all the X_a's are pairwise disjoints.

Let $\mathcal{A} = (\Sigma_e, \Sigma_s, X, Q, Q_0, F, R, I, \ell, \Delta)$ be a SEA-automaton accepting a language $L \subseteq SE(\Sigma)$. We assume that X is disjoint from the X_a's. We build from $\mathcal{A}$ a SEA-automaton $\mathcal{A}'$ accepting $\sigma(L)$ as follows. For each state p of $\mathcal{A}$ with label a, we consider a new copy $\mathcal{A}_p$ of $\mathcal{A}_a$ in which the invariant $I(p)$ is added to all states: $I_p(r_p) = I_a(r) \wedge I(p)$ if r_p is the copy of state $r \in Q_a$. To build $\mathcal{A}'$, we start with the disjoint union of all the $\mathcal{A}_p$'s and we add *switching* transitions:

> If (p, g, f, α, q) is a transition of $\mathcal{A}$ then for each final state r_p of $\mathcal{A}_p$ and each initial state s_q of $\mathcal{A}_q$ we add the transition $(r_p, g, f, \alpha \cup X_{\ell(q)}, s_q)$ to $\mathcal{A}'$.

The initial states of $\mathcal{A}'$ are the initial states of all $\mathcal{A}_p$ such that $p \in Q_0$. The final states of $\mathcal{A}'$ are the final states of all $\mathcal{A}_p$ such that $p \in F$. An infinite path of $\mathcal{A}'$ is accepting if

– either it uses infinitely many switching transitions $(r_p, g, f, \alpha \cup X_{\ell(q)}, s_q)$ with $p \in R$,
– or else, it stays ultimately in some $\mathcal{A}_p$ with $p \in F$ and it visits R_p infinitely often.

We can easily transform the automaton $\mathcal{A}'$ in order to get a classical Büchi condition if needed. We can check that $\mathcal{A}'$ accepts $\sigma(L)$. □

We will now show that the class SEL_ε is closed under SEL_ε-substitutions. The construction for the substitution of signals given in the previous proof does not work. Indeed, by definition, a substitution must be applied to a word in normal form. The difficulty comes from the fact that in the automaton for L, a factor a^d of some normal form may be generated by a path with several a-labelled states and even τ-labelled states that are crossed instantaneously if all these states are linked by ε-transitions. So we cannot simply replace each a-labelled state by a copy of $\mathcal{A}_a$.

To circumvent this difficulty, we use a proof technique inspired from rational transductions and that can be applied to establish the closure of the class SEL_ε both under SEL_ε-substitutions and their inverse. Hence, we state and prove both results simultaneously. It should be noted that these closure properties hold for arbitrary substitutions, showing once again the robustness of the class SEL_ε.

Theorem 2. *The class SEL_ε is closed under SEL_ε-substitution and inverse SEL_ε-substitution.*

Proof. Let σ be a SEL_ε-substitution from $SE(\Sigma)$ to $SE(\Sigma')$ given by a family of automata $(\mathcal{A}_a)_{a\in\Sigma_e\cup\Sigma_s}$, with $\mathcal{A}_a = (\Sigma'_e, \Sigma'_s, X_a, Q_a, Q^0_a, F_a, R_a, I_a, \ell_a, \Delta_a) \in SEA_\varepsilon$. We assume that these automata satisfy condition (†) of Proposition 1.

We use the same technique to prove both closure properties. We show that $\sigma(L)$ and $\sigma^{-1}(L)$ can both be expressed as a projection of the intersection of a SEL_ε-language with an inverse projection of L. This is in the spirit of rational transductions for classical word languages.

Consider a new alphabet $\hat{\Sigma} = \hat{\Sigma}_e \cup \hat{\Sigma}_s \times \mathbb{T}$ with $\hat{\Sigma}_e = \Sigma_e \uplus \Sigma'_e$ ($\uplus$ is the disjoint union) and $\hat{\Sigma}_s = \Sigma_s \times \Sigma'_s$ (we identify (τ,τ) with the unobservable signal τ). The projections π_1 and π_2 are the morphisms defined by:

$\pi_1(f) = f$ and $\pi_2(f) = \varepsilon$ if $f \in \Sigma_e$,
$\pi_1(f) = \varepsilon$ and $\pi_2(f) = f$ if $f \in \Sigma'_e$,
$\pi_1((a,b)^d) = a^d$ and $\pi_2((a,b)^d) = b^d$ if $(a,b)^d \in \Sigma_s \times \Sigma'_s \times \overline{\mathbb{T}}$.

With this definition we have $\pi_i((a,b)^{d_1+d_2}) = \pi_i((a,b)^{d_1})\pi_i((a,b)^{d_2})$. Note that projection π_i is a composition of an event-hiding and a signal-renaming. By Proposition 4 we deduce that the projection by π_i of a SEL_ε language is again in the class SEL_ε.

For $L \subseteq SE(\Sigma)$, we let $\pi_1^{-1}(L) = \{w \in SE(\hat{\Sigma}) \mid \pi_1(w) \in L\}$. We define similarly $\pi_2^{-1}(L)$ for $L \subseteq SE(\Sigma')$. We will show later that if L is recognizable then so is $\pi_i^{-1}(L)$. We will also define a recognizable language $M \subseteq SE(\hat{\Sigma})$ with the following properties:

1. for each $w \in M$, we have $\pi_2(w) \in \sigma(\pi_1(w))$,
2. for each $u \in SE(\Sigma)$ and $v \in \sigma(u)$, there exists $w \in M$ such that $u = \pi_1(w)$ and $v = \pi_2(w)$.

Then, for $L \subseteq SE(\Sigma)$, we have $\sigma(L) = \pi_2(\pi_1^{-1}(L) \cap M)$. Indeed, Let $v \in \sigma(L)$ and let $u \in L$ with $v \in \sigma(u)$. Using property 2 of M we find $w \in M$ with $\pi_1(w) = u$ and $\pi_2(w) = v$. Then, $w \in \pi_1^{-1}(L) \cap M$ and $v \in \pi_2(\pi_1^{-1}(L) \cap M)$. Conversely, let $v \in \pi_2(\pi_1^{-1}(L) \cap M)$ and let $w \in \pi_1^{-1}(L) \cap M$ with $\pi_2(w) = v$. By definition, we have $u = \pi_1(w) \in L$ and using property 1 of M, we get $v \in \sigma(u) \subseteq \sigma(L)$.

Similarly, for $L \subseteq SE(\Sigma')$, we have $\sigma^{-1}(L) = \pi_1(\pi_2^{-1}(L) \cap M)$. Indeed, Let $u \in \sigma^{-1}(L)$ and let $v \in \sigma(u) \cap L$. Using property 2 of M we find $w \in M$ with $\pi_1(w) = u$ and $\pi_2(w) = v$. Then, $w \in \pi_2^{-1}(L) \cap M$ and $u \in \pi_1(\pi_2^{-1}(L) \cap M)$. Conversely, let $u \in \pi_1(\pi_2^{-1}(L) \cap M)$ and let $w \in \pi_2^{-1}(L) \cap M$ with $\pi_1(w) = u$. By definition, we have $v = \pi_2(w) \in L$ and using property 1 of M, we get $v \in \sigma(u)$. Hence, $\sigma(u) \cap L \neq \emptyset$ and $u \in \sigma^{-1}(L)$.

We already know that SEL_ε-languages are closed under intersection (Proposition 2) and projections π_i. To conclude the proof of Theorem 2, it remains to show that they are also closed under inverse projections π_i^{-1} and to define the SEL_ε-language M with the properties above.

We show first that SEL_ε-langagues are closed under inverse projections π_i^{-1}. Let L be recognized by some automaton $\mathcal{A} = (\Sigma_e, \Sigma_s, X, Q, Q^0, F, R, I, \ell, \Delta) \in SEA_\varepsilon$. We build an automaton $\hat{\mathcal{A}}$ accepting $\pi_1^{-1}(L)$. The set of states is $\hat{Q} = Q \uplus Q \times \Sigma'_s$, with $\hat{Q}^0 = Q^0$ for initial states and $\hat{F} = F$ for final states. The set of clocks is $X \uplus \{z\}$. The labels and invariants are defined by $\hat{\ell}(q) = \tau$ and $\hat{I}(q) = I(q) \wedge (z \leq 0)$ for $q \in Q$, and $\hat{\ell}((q, b)) = (\ell(q), b)$ and $\hat{I}((q, b)) = I(q)$ for $(q, b) \in Q \times \Sigma'_s$. The set of transitions $\hat{\Delta}$ is defined by:

1. All transitions $(p, g, f, \alpha, q) \in \Delta$ are kept in $\hat{\Delta}$.
2. For all $f' \in \Sigma'_e$ and $q \in Q$, we put $(q, \texttt{true}, f', \emptyset, q)$ in $\hat{\Delta}$.
3. For all $(q, b) \in Q \times \Sigma'_s$, we put $(q, \texttt{true}, \varepsilon, \emptyset, (q, b))$, $((q, b), \texttt{true}, \varepsilon, \{z\}, q)$ in $\hat{\Delta}$.

We use a generalized acceptance condition for infinite paths. By transforming the automaton we can get a classical Büchi condition if needed. An infinite path is accepting if it uses

- either infinitely many transitions (p, g, f, α, q) of type 1 with $q \in R$,
- or ultimately transitions of type 2 and 3 only around some state $q \in F$.

We show that $\mathcal{L}(\hat{\mathcal{A}}) = \pi_1^{-1}(L)$. Let w be a word over $\hat{\Sigma}$ accepted by $\hat{\mathcal{A}}$. We consider a run of $\hat{\mathcal{A}}$ for w through an accepting path $\hat{P}$. Erasing from this path all transitions of type 2 and 3 above, we obtain a path P of $\mathcal{A}$. If $\hat{P}$ is finite then it ends in a final state $q \in \hat{F} = F$ and P is also accepting since it ends in state q. If $\hat{P}$ uses infinitely many transitions (p, g, f, α, q) of type 1 with $q \in R$, then P is infinite and visits R infinitely often, hence it is also accepting. Finally, if $\hat{P}$ is infinite and uses ultimately transitions of type 2 and 3 only around some state $q \in F$, then P is finite and accepting since it ends in state q. In all cases, P is an accepting path of $\mathcal{A}$. We can show that $\pi_1(w)$ admits a run through P. Therefore, $\pi_1(w) \in L$.

Conversely, let $w \in SE(\hat{\Sigma})$ and assume that $\pi_1(w) \in L$ is accepted by a run $\langle q_0, v_0 \rangle \xrightarrow{d_0} \langle q_0, v_0 + d_0 \rangle \xrightarrow{f_1} \langle q_1, v_1 \rangle \xrightarrow{d_1} \langle q_1, v_1 + d_1 \rangle \xrightarrow{f_2} \langle q_2, v_2 \rangle \cdots$ through an accepting path $P = q_0 \xrightarrow{g_1, f_1, \alpha_1} q_1 \xrightarrow{g_2, f_2, \alpha_2} q_2 \cdots$ of $\mathcal{A}$. Then, we have $\pi_1(w) \approx$

$a_0^{d_0} f_1 a_1^{d_1} f_2 \cdots$ with $a_i = \ell(q_i)$ for $i \geq 0$. We deduce that $w \approx w_0 f_1 w_1 f_1 w_2 \cdots$ with $\pi_1(w_i) = a_i^{d_i}$. Now, if w_i is finite with finite duration then we find a path $\hat{P}_i$ following the normal form of w_i, starting and ending in q_i, and using transitions of type 2 and 3 only. If w_i is infinite or with infinite duration then the path P must be finite ending in state $q_i \in F$ since P is accepting. Then, we find an *infinite* path $\hat{P}_i$ for w_i using only transitions of type 2 and 3 around q_i. Note that if w_i is finite with infinite duration, then it ends with b^∞ for some $b \in \Sigma'_s$ and we still need an infinite path ultimately alternating between states q_i and (q_i, b). Finally, $\hat{P} = \hat{P}_0 \xrightarrow{g_1, f_1, \alpha_1} \hat{P}_1 \xrightarrow{g_2, f_2, \alpha_2} \hat{P}_2 \cdots$ is a path in $\hat{\mathcal{A}}$ and it is easy to see that $\hat{P}$ is accepting. Moreover, we can show that w admits a run through $\hat{P}$ and therefore, w is accepted by $\hat{\mathcal{A}}$.

We turn now to the definition of M. For $f \in \Sigma_e$ and $a \in \Sigma_s \setminus \{\tau\}$, we define

$$\begin{aligned} M_f &= \{w \in SE(\hat{\Sigma}) \mid w = (\tau, b_0)^0 f_1 (\tau, b_1)^0 f_2 \cdots (\tau, b_n)^0 \\ &\qquad\qquad \text{with } b_0^0 f_1 b_1^0 f_2 \cdots b_n^0 \in \sigma(f)\} \cdot f \\ M_a &= \{w \in SE(\hat{\Sigma}) \mid w = (a, b_0)^{d_0} f_1 (a, b_1)^{d_1} f_2 \cdots \\ &\qquad\qquad \text{with } b_0^{d_0} f_1 b_1^{d_1} f_2 \cdots \in \sigma(a^{d_0 + d_1 + \cdots})\} \end{aligned}$$

We also let $M_\tau = \{(\tau, \tau)^d \mid d \in \overline{\mathbb{T}} \setminus \{0\}\}$. Note that for $w \in M_a$ with $a \in \Sigma_e \cup \Sigma_s$ we have $\pi_2(w) \in \sigma(\pi_1(w))$ as required by property 1 of M. Moreover, if $f \in \Sigma_e$ and $v \in \sigma(f)$ then there exists $w \in M_f$ such that $\pi_1(w) = f$ and $\pi_2(w) = v$. Similarly, if $a^d \in \Sigma_s \times \overline{\mathbb{T}}$ (with $d > 0$ if $a = \tau$) and $v \in \sigma(a^d)$ then there exists $w \in M_a$ such that $\pi_1(w) = a^d$ and $\pi_2(w) = v$.

Intuitively, M consists of finite or infinite products of words in $\bigcup_{a \in \Sigma_e \cup \Sigma_s} M_a$ except that, in order to ensure that the first projection is in normal form, we should not allow consecutive factors associated with the same signal. Formally, we define

$$M = \{w_1 w_2 \cdots \mid \exists a_1, a_2, \ldots \in \Sigma_e \cup \Sigma_s \text{ with } w_i \in M_{a_i} \text{ and } a_i \in \Sigma_s \Rightarrow a_{i+1} \neq a_i\}.$$

We show that M satisfies property 1. Let $w = w_1 w_2 \cdots \in M$ and let $a_1, a_2, \ldots \in \Sigma_e \cup \Sigma_s$ be such that $w_i \in M_{a_i}$ and $a_i \in \Sigma_s \Rightarrow a_{i+1} \neq a_i$. Then, $\pi_1(w) = \pi_1(w_1)\pi_1(w_2) \cdots$ is in normal form and we have seen above that $\pi_2(w_i) \in \sigma(\pi_1(w_i))$. Therefore, $\pi_2(w) \in \sigma(\pi_1(w))$ and property 1 is proved.

We show that M satisfies property 2. Let $u \in SE(\Sigma)$ and $v \in \sigma(u)$. Write $u = u_1 u_2 \cdots$ in normal form and $v = v_1 v_2 \cdots$ with $v_i \in \sigma(u_i)$. Let $a_i = u_i$ if $u_i \in \Sigma_e$ and $a_i = a$ if $u_i = a^d \in \Sigma_s \times \overline{\mathbb{T}}$. Since the product $u = u_1 u_2 \cdots$ is in normal form, we have $a_i \in \Sigma_s \Rightarrow a_{i+1} \neq a_i$ and $u_i = \tau^d \Rightarrow d > 0$. Since $v_i \in \sigma(u_i)$, we have seen above that there exists $w_i \in M_{a_i}$ such that $\pi_1(w_i) = u_i$ and $\pi_2(w_i) = v_i$. Then, $w = w_1 w_2 \cdots \in M$ and $\pi_1(w) = u$ and $\pi_2(w) = v$ as required by property 2.

It remains to show that M is recognizable by some SEA_ε-automaton. For each $a \in \Sigma_e \cup \Sigma_s$ we first show that M_a is accepted by some automaton $\hat{\mathcal{A}}_a \in SEA_\varepsilon$ derived from $\mathcal{A}_a$. This is clear for $a = \tau$. Note that $\hat{\mathcal{A}}_\tau$ needs two states to ensure the positive duration required by M_τ. For $a \in \Sigma_s \setminus \{\tau\}$, the automaton $\hat{\mathcal{A}}_a$ is simply a copy of $\mathcal{A}_a$, with new label $(a, \ell_a(q))$ for $q \in Q_a$. For $f \in \Sigma_e$, the set of states is $\hat{Q}_f = Q_f \uplus \{q_f\}$, where q_f is a new state, which is also the only final state. The label of $q \in Q_f$ is $(\tau, \ell_f(q))$, the label of q_f is (τ, τ) and its invariant is $x_f \leq 0$ where x_f is the clock ensuring instantaneous traversal of $\mathcal{A}_f$. The transitions are those in $\mathcal{A}_f$, to which we add (q, f, q_f) for any state q which was final in $\mathcal{A}_f$. Note that, since $\mathcal{A}_a$ satisfies (†) for $a \in \Sigma_e \cup \Sigma_s$, then so does $\hat{\mathcal{A}}_a$.

Since M is essentially the iteration of the languages M_a, it should be clear that $M \in SEL_\varepsilon$. The SEA_ε-automaton $\mathcal{B}$ recognizing M is the disjoint union of the automata $\hat{\mathcal{A}}_a$ to which we add ε-transitions allowing to switch between automata: if p is a final state of $\hat{\mathcal{A}}_a$ and q is an initial state of $\hat{\mathcal{A}}_b$ and $a \in \Sigma_s \Rightarrow b \neq a$ then we add the transition $(p, \texttt{true}, \varepsilon, X_b, q)$. All initial (resp. final, repeated) states of the $\hat{\mathcal{A}}_a$'s are initial (resp. final, repeated) in $\mathcal{B}$. But we also need to accept runs that switch infinitely often between the $\hat{\mathcal{A}}_a$'s, i.e., taking infinitely many switching ε-transitions. If needed, it is easy to transform the automaton so that it uses classical Büchi condition to accept also these runs.

We can show that any SE-word accepted by $\mathcal{B}$ is in M. Condition (†) is needed to prove the converse. Indeed, a finite SE-word $v \in M_a$ with $\|v\| < \infty$ could appear as an internal factor of some SE-word $uvw \in M$ with $w \neq \varepsilon$. If v could only be accepted by an infinite run in $\hat{\mathcal{A}}_a$, then we would not be able to build a (non transfinite) run for uvw in $\mathcal{B}$. □

The class SEL is not closed under inverse SEL-substitutions. Indeed, assume $\Sigma_s = \Sigma'_s = \{a, b\}$, $\Sigma_e = \Sigma'_e = \{f\}$ and let σ be the SEL-substitution defined by $L_a = \{a^1 f\}$, $L_b = \{b^0\}$ and $L_f = \{f\}$. Then the inverse image by σ of the SEL-language $\{a^1 f b^0\}$ is the language $\{a^1 b^0\}$ which is not in the class SEL. Nevertheless sufficient conditions on the substitutions can be proposed to ensure the closure of the class SEL under inverse substitutions.

Theorem 3. *The class SEL is closed under inverse SEL-substitutions acting on events only.*

Proof. Let σ be a substitution defined by a family $(L_a)_{a \in \Sigma_e \cup \Sigma_s}$ of SEL-languages. We assume that σ acts only on events, i.e., $L_a = \{a\} \times \overline{\mathbb{T}}$ for all $a \in \Sigma_s$. Let $\mathcal{A}_f = (\Sigma'_e, \Sigma'_s, \{x_f\}, Q_f, Q^0_f, F_f, \emptyset, I_f, \ell_f, \Delta_f)$ for $f \in \Sigma_e$ be an automaton without ε-transitions accepting L_f. Since the guards and resets on the transitions of $\mathcal{A}_f$ are always $\texttt{true}$ and $\emptyset$ respectively, we write a transition of $\mathcal{A}_f$ simply (r, f', s) to simplify the notation. We consider now a language $L \in SEL$, recognized by some automaton without ε-transition $\mathcal{A}_2 = (\Sigma'_e, \Sigma_s, X, Q, Q^0, F, R, I, \ell, \Delta) \in SEA$.

We build an automaton $\mathcal{A}_1$ in SEA accepting $\sigma^{-1}(L)$ essentially by changing the transitions of $\mathcal{A}_2$. If there is a path P in $\mathcal{A}_2$ from p to q having some instantaneous run for some word in $\sigma(f)$ then we add to $\mathcal{A}_1$ a transition (p, g, f, α, q) with a suitable guard g and reset α. The difficulty is to compute a suitable pair (g, α) for each triple (p, q, f).

Given a guard g and a subset of clocks α, we define the restriction of g by α, written $g[\alpha]$, as the guard g where all clocks from α have been replaced by 0. For instance, if g is $(x < 3) \wedge (y > 2)$ and $\alpha = \{x, z\}$ then $g[\alpha]$ is (equivalent to) $(y > 2)$. We let $\mathcal{G}$ be the smallest set of guards including all guards of $\mathcal{A}_2$ and closed under conjunctions and restrictions. Formally $\mathcal{G}$ is not a finite set, but it can be identified with its finite quotient under equivalence of formulae: two formulae φ and ψ are equivalent if $v \models \varphi$ iff $v \models \psi$ for all valuations v. The set $\Gamma = \mathcal{G} \times \mathcal{P}(X)$ is thus a finite monoid, with $(True, \emptyset)$ as neutral element, for the associative composition:

$$(g_1, \alpha_1) \cdot (g_2, \alpha_2) = (g_1 \wedge g_2[\alpha_1], \alpha_1 \cup \alpha_2).$$

Finally, we define a morphism $\gamma : \Delta^* \mapsto \Gamma$ by $\gamma((p, g, f, \alpha, q)) = (g, \alpha)$.

Let $P = q_0 \xrightarrow{g_1,b_1,\alpha_1} q_1 \xrightarrow{g_2,b_2,\alpha_2} \cdots \xrightarrow{g_n,b_n,\alpha_n} q_n$ be a path in $\mathcal{A}_2$. We define $\mathcal{W}(P) = \ell(q_0)^0 b_1 \ell(q_1)^0 b_2 \cdots b_n \ell(q_n)^0$. Now, given a triple (p, q, f) where $p, q \in Q$ and $f \in \Sigma_e$, we denote by $L^f_{p,q}$ the set of paths P from p to q in $\mathcal{A}_2$ with $\mathcal{W}(P) \in \sigma(f)$. We can build an automaton $\mathcal{B}^f_{p,q}$ recognizing the language $L^f_{p,q} \subseteq \Delta^*$. This is not difficult since we are dealing with automata without ε-transitions, hence we can perform a simple synchronized product as follows. The set of states is $Q' = \{(r, s) \in Q_f \times Q \mid \ell_f(r) = \ell(s)\}$, the set of initial states is $Q' \cap Q^0_f \times \{p\}$, the set of final states is $Q' \cap F_f \times \{q\}$. The transitions of $\mathcal{B}^f_{p,q}$ are the triples $((r_1, s_1), (s_1, g, b, \alpha, s_2), (r_2, s_2))$ with $(r_1, b, r_2) \in \Delta_f$ and $(s_1, g, b, \alpha, s_2) \in \Delta$.

We let $\Delta_R \subseteq \Delta^*$ be the set of sequences containing at least a transition ending in some repeated state of R. From the automaton $\mathcal{B}^f_{p,q}$, we can effectively compute a rational expression for the language $L^f_{p,q} \cap \Delta_R$. We deduce that we can effectively compute the finite set $\gamma(L^f_{p,q} \cap \Delta_R)$. Similarly, we can effectively compute the finite set $\gamma(L^f_{p,q} \setminus \Delta_R)$. These two sets are used to define the transitions of a new automaton $\mathcal{A}_1 = (\Sigma_e, \Sigma_s, X, Q \uplus \overline{Q}, Q^0, F, \overline{Q}, I, \ell, \Delta_1)$ in *SEA*:

$$\begin{aligned} \Delta_1 = \quad & \{(p, g, f, \alpha, q), (\overline{p}, g, f, \alpha, q) \mid (g, \alpha) \in \gamma(L^f_{p,q} \setminus \Delta_R)\} \\ & \cup \{(p, g, f, \alpha, \overline{q}), (\overline{p}, g, f, \alpha, \overline{q}) \mid (g, \alpha) \in \gamma(L^f_{p,q} \cap \Delta_R)\}. \end{aligned}$$

The automaton $\mathcal{A}_1$ can therefore be effectively computed from the automata $\mathcal{A}_2$ and $(\mathcal{A}_f)_{f \in \Sigma_e}$. We can show that $L(\mathcal{A}_1) = \sigma^{-1}(L)$. Therefore, the language $\sigma^{-1}(L) \in$ *SEL* and Theorem 3 is proved. □

5 Conclusion

We have shown in this paper that the class *SEL* of signal-event languages is not closed under arbitrary *SEL*-substitutions and inverse *SEL*-substitutions but that natural sufficient conditions ensure closure properties for this class.

But our main contribution is to propose effective constructions to prove the closure of the larger class SEL_ε under arbitrary SEL_ε-substitutions and inverse SEL_ε-substitutions. We give these constructions in the general framework of signal-event automata and languages. The usual cases of event languages [1,2] or signal languages [3,10,4,11] are particular cases for which the interested reader will check that simplified constructions can be given.

References

1. R. Alur and D.L. Dill. Automata for modeling real-time systems. In *Proceedings of ICALP'90*, number 443 in LNCS, pages 322–335. Springer, 1990.
2. R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

3. E. Asarin, P. Caspi, and O. Maler. A Kleene theorem for timed automata. In *Proceedings of LICS'97*, pages 160–171. IEEE Comp. Soc. Press, 1997.
4. E. Asarin, P. Caspi, and O. Maler. Timed regular expressions. *Journal of the ACM*, 49(2):172–206, 2002.
5. B. Bérard, V. Diekert, P. Gastin, and A. Petit. Characterization of the expressive power of silent transitions in timed automata. *Fundamenta Informaticae*, 36:145–182, 1998.
6. B. Bérard, P. Gastin, and A. Petit. Timed substitutions for regular signal-event languages. Research Report LSV-06-04, Laboratoire Spécification et Vérification, ENS Cachan, France, February 2006.
7. B. Bérard, P. Gastin, and A. Petit. Intersection of regular signal-event (timed) languages. In *Proceedings of FORMATS'06*, number 4202 in LNCS. Springer, 2006.
8. B. Bérard and C. Picaronny. Accepting Zeno words: a way toward timed refinements. *Acta Informatica*, 37(1):45–81, 2000.
9. P.J.L. Cuijpers, M.A. Reniers, and A.G. Engels. Beyond zeno-behaviour. Technical Report CSR 01-04, Department of Computing Science, University of Technology, Eindhoven, 2001.
10. C. Dima. Real-Time Automata and the Kleene Algebra of Sets of Real Numbers. In *Proceedings of STACS'2000*, number 1770 in LNCS, pages 279–289. Springer, 2000.
11. J. Durand-Lose. A Kleene theorem for splitable signals. *Information Processing Letters*, 89:237–245, 2004.
12. M.R. Hansen, P.K. Pandya, and C. Zhou. Finite divergence. *Theoretical Computer Science*, 138:113–139, 1995.

Bridging the Gap Between Timed Automata and Bounded Time Petri Nets

Bernard Berthomieu, Florent Peres, and François Vernadat

LAAS-CNRS, 7 avenue du Colonel Roche, 31077 Toulouse, France
fax: +33 (0)5.61.33.64.11; tel.: +33 (0)5.61.33.63.63
{Bernard.Berthomieu, Florent.Peres, Francois.Vernadat}@laas.fr

Abstract. Several recent papers investigate the relative expressiveness of Timed Automata and Time Petri Nets, two widespread models for realtime systems. It has been shown notably that Timed Automata and Bounded Time Petri Nets are equally expressive in terms of timed language acceptance, but that Timed Automata are strictly more expressive in terms of weak timed bisimilarity. This paper compares Timed Automata with Bounded Time Petri Nets extended with static Priorities, and shows that two large subsets of these models are equally expressive in terms of weak timed bisimilarity.

Keywords: Time Petri nets, priorities, Timed Automata, weak timed bisimilarity, real-time systems modeling and verification.

1 Introduction

Among the many models proposed for the specification and verification of real time systems, two are prominent: Time Petri nets and Timed Automata.

Time Petri nets (TPN) [15] extend Petri nets with temporal intervals associated with transitions, specifying firing delay ranges for the transitions. Assuming transition t became last enabled at time θ, and the end-points of its time interval are α and β, then t cannot fire earlier than time $\theta + \alpha$ and must fire no later than $\theta + \beta$, unless disabled by firing some other transition. Firing a transition takes no time. Many other Petri net based models with time extensions have been proposed, but none reaches the acceptance of Time Petri nets. Availability of effective analysis methods, prompted by [5], certainly contributed to their widespread use, together with their ability to cope with a wide variety of modeling problems for realtime systems.

Timed Automata (TA) [2] extend finite automata with clocks, guards, resets and a product operation. Transitions are guarded by boolean conditions on clock values. When taken, they may emit a label and perform resets of some clocks. All clocks progress synchronously as time elapses. Some versions of timed automata support progress annotations as location invariants limiting elapsing of time when at that location, urgency requirements, or transition deadlines. Timed automata are convenient for modeling a large class of realtime problems. They prompted a considerable amount of research work and benefit from a rich theory.

These two models, as well as their analysis techniques, were developed independently for years, though they bear strong relationships. State space abstractions for TPN's preserving various classes of properties can be computed in terms of so-called state classes [5] [4] [6]. State classes represent sets of states by a marking and a polyhedron capturing temporal information. State space abstractions for (Networks of) Timed Automata are based upon geometric regions characterizing sets of states by a location of the underlying automaton and a convex set capturing temporal information. In both cases, the convex sets can be represented by difference systems, or DBM's.

In spite of many technical resemblances and their overlapping application domains, few material was available until recently comparing expressiveness of these two models. A number of recent works finally addressed the issue. [11] translated a subclass of TA's into TPN's, preserving timed language acceptance. Later, [9] proposed a structural encoding of TPN's into TA's, improving an earlier semantics based encoding in [14]. [3] proves than TPN's and TA's are equivalent w.r.t. timed language acceptance, but that TA's are strictly more expressive in terms of timed bisimilarity, they also discuss the subclass of TA's weakly timed bisimilar with some TPN.

In this article, we first extend Time Petri nets with static priorities. In TPN's with Priorities ($PrTPN$'s for short), a transition is not allowed to fire if some transition with higher priority is firable at the same instant. Such priorities have many applications in realtime systems, in scheduling, arbitration, synchronization, and others problems. We then develop an encoding of Timed Automata (without progress requirements) into $PrTPN$'s, preserving weak timed bisimilarity. Next, we extend TA's with invariants and show that TA's with invariants built from $\{\leq, \wedge\}$ can be encoded into $PrTPN$'s. Finally, extending the encoding of [9] of TPN's into TA's, we show that TA's with invariants built from $\{\leq, \wedge\}$ are equally expressive than $PrTPN$'s with unbounded or right-closed intervals. Some corollaries follow that extend available equivalence results between TA's and TPN's (without priorities).

The paper is organized as follows. Section 2 recalls the essentials about timed transition systems (TTS), the common semantic domain for TA's and $PrTPN$'s. Section 3 reviews the terminology of Timed automata and their semantics. Section 4 introduces Time Petri nets with Priorities, and compares their expressiveness with that of TPN's. Section 5 explains how to encode Timed Automata (without invariants) into weakly timed bisimilar $PrTPN$'s. Section 6 discusses progress requirements, extends the encoding to TA's with invariants built from $\{\leq, \wedge\}$, and derives a number of ordering or equivalence results. Finally, Section 7 discusses some consequences, side issues, and prospective work.

2 Timed Transition Systems

The semantics of Timed Automata (TA) and Time Petri Nets ($PrTPN$) will be given in terms of Timed Transition Systems (TTS), as described in e.g. [13]. We review here their terminology and some key concepts used in the next sections.

Timed Transition Systems: $\mathbf{R}^+$ is the set of nonnegative reals. A *Timed Transition System* is a structure $\langle Q, q^0, \Sigma \cup \{\epsilon\}, \rightarrow \rangle$ where:

- Q is a set of *states*
- $q^0 \in Q$ is the *initial state*
- Σ is a finite set of *actions* not containing the *silent* action ϵ
- $\rightarrow \subseteq Q \times (\Sigma \cup \{\epsilon\} \cup \mathbf{R}^+) \times Q$ is the *transition* relation.

$(q, a, q') \in \rightarrow$ is also written $q \xrightarrow{a} q'$. Σ^ϵ abbreviates $\Sigma \cup \{\epsilon\}$. The transitions belonging to $Q \times \Sigma^\epsilon \times Q$ are called *discrete* transitions, those from $Q \times \mathbf{R}^+ \times Q$ are the *continuous* transitions. Continuous transitions are typically required to obey the following properties ($\forall d, d', d'' \in \mathbf{R}^+$):

- *0-delay*: $q \xrightarrow{0} q' \Leftrightarrow q = q'$
- *additivity*: $q \xrightarrow{d} q' \wedge q' \xrightarrow{d'} q'' \Rightarrow q \xrightarrow{d+d'} q''$
- *continuity*: $q \xrightarrow{d+d'} q' \Rightarrow (\exists q'')(q \xrightarrow{d} q'' \wedge q'' \xrightarrow{d'} q')$
- *time-determinism*: $q \xrightarrow{d} q' \wedge q \xrightarrow{d} q'' \Rightarrow q' = q''$

Product of $\mathbf{\mathit{TTS}}$**:** Let $S_1 = \langle Q_1, q_1^0, \Sigma_1^\epsilon, \rightarrow_1 \rangle$ and $S_2 = \langle Q_2, q_2^0, \Sigma_2^\epsilon, \rightarrow_2 \rangle$ be two TTS. We assume that every action of Σ_i labels some transition of S_i. The product of S_1 by S_2 is the TTS $S_1 \parallel S_2 = \langle Q_1 \times Q_2, q_1^0 \parallel q_2^0, \Sigma_1^\epsilon \cup \Sigma_2^\epsilon, \rightarrow \rangle$ where $\rightarrow$ is the smallest relation obeying the following rules ($a \in \Sigma_1 \cup \Sigma_2 \cup \{\epsilon\} \cup \mathbf{R}^+$):

$$\frac{q_1 \xrightarrow{a}_1 q_1'}{q_1 \parallel q_2 \xrightarrow{a} q_1' \parallel q_2}\ (a \in \Sigma_1^\epsilon \backslash \Sigma_2) \qquad \frac{q_2 \xrightarrow{a}_2 q_2'}{q_1 \parallel q_2 \xrightarrow{a} q_1 \parallel q_2'}\ (a \in \Sigma_2^\epsilon \backslash \Sigma_1)$$

$$\frac{q_1 \xrightarrow{a}_1 q_1' \quad q_2 \xrightarrow{a}_2 q_2'}{q_1 \parallel q_2 \xrightarrow{a} q_1' \parallel q_2'}\ (a \neq \epsilon)$$

Timed Bisimilarity: Let $S_1 = \langle Q_1, q_1^0, \Sigma_1^\epsilon, \rightarrow_1 \rangle$ and $S_2 = \langle Q_2, q_2^0, \Sigma_2^\epsilon, \rightarrow_2 \rangle$ be two TTS and $\sim \subseteq Q_1 \times Q_2$. Then S_1 and S_2 are *strongly timed bisimilar* iff $q_1^0 \sim q_2^0$ and, whenever $q_1 \sim q_2$ and $a \in \Sigma_1^\epsilon \cup \Sigma_2^\epsilon \cup \mathbf{R}^+$:

(1) $q_1 \xrightarrow{a}_1 q_1' \Rightarrow (\exists q_2')(q_2 \xrightarrow{a}_2 q_2' \wedge q_1' \sim q_2')$
(2) $q_2 \xrightarrow{a}_2 q_2' \Rightarrow (\exists q_1')(q_1 \xrightarrow{a}_1 q_1' \wedge q_1' \sim q_2')$

Strong timed bisimilarity is often too strong a requirement. A coarser equivalence relation, hiding silent transitions, is obtained from relation $\overset{a}{\Longrightarrow}$, defined from $\xrightarrow{a}$ as follows ($a \in \Sigma \cup \mathbf{R}^+$, $d \in \mathbf{R}^+$):

$$\frac{q \xrightarrow{a} q'}{q \overset{a}{\Longrightarrow} q'} \quad \frac{q \overset{a}{\Longrightarrow} q' \quad q' \xrightarrow{\epsilon} q''}{q \overset{a}{\Longrightarrow} q''} \quad \frac{q \xrightarrow{\epsilon} q' \quad q' \overset{a}{\Longrightarrow} q''}{q \overset{a}{\Longrightarrow} q''} \quad \frac{q \overset{d}{\Longrightarrow} q' \quad q' \overset{d'}{\Longrightarrow} q''}{q \overset{d+d'}{\Longrightarrow} q''}$$

Two timed transition systems are *weakly timed bisimilar* when conditions (1) and (2) above hold, with relations $\xrightarrow{a}_i$ replaced by $\overset{a}{\Longrightarrow}_i$ ($i \in \{1, 2\}$).

3 Timed Automata

$\mathbf{Q}^+$ is the set of nonnegative rationals. Given a finite set of clocks X, the set $\mathcal{C}(X)$ of clock constraints over X is defined by the grammar:

$$g ::= x\#c \mid g \wedge g \mid \texttt{true} \text{ where } x \in X,\ c \in \mathbf{Q}^+ \text{ and } \# \in \{\leq, <, \geq, >\}$$

Definition 1. *A* Timed Automaton *(TA), is a tuple $\langle Q, q^0, X, \Sigma^\epsilon, T\rangle$ in which:*

- *Q is a finite set of* locations
- *$q^0 \in Q$ is the* initial location
- *X is a finite set of clocks*
- *Σ is a finite set of actions, assumed not to contain the* silent *action ϵ*
- *$T \subseteq Q \times (\mathcal{C}(X) \times \Sigma^\epsilon \times 2^X) \times Q$ is a finite set of transitions, or edges.*

$(q, g, a, R, q') \in T$ may be written $q \xrightarrow{g,a,R} q'$. A *configuration* of a Timed Automata is a pair (q, v), where $q \in Q$ and v is a vector of clock values (one for each clock in X). $v[R := 0]$ denotes the vector in which the components corresponding to clocks in R are 0 and the others are as in v. $|E|$ is $card(E)$.

Definition 2. *The semantics of a Timed Automaton $\langle Q, q^0, X, \Sigma^\epsilon, T\rangle$ is the TTS $\langle S, s^0, \Sigma^\epsilon, \rightarrow\rangle$ where $S = Q \times (\mathbf{R}^+)^{|X|}$, $s^0 = (q^0, \overline{0})$, and $\rightarrow$ is defined by:*

- $(q, v) \xrightarrow{a} (q', v')$ *iff* $(\exists(q, g, a, R, q') \in T)(g(v) \wedge v' = v[R := 0])$
- $(q, v) \xrightarrow{d} (q, v + d)$ *iff* $d \in \mathbf{R}^+$

Product of timed automata: A product construction $||$ is defined for timed automata, allowing one to express complex realtime systems as synchronized components. A definition of this product can be found in e.g. [1]. That product construction is compositional in that the TTS denoted by a product of timed automata is the product of the TTS's respectively denoted by the components.

Progress requirements: Our definition of TA, taken from [2], does not include any means of enforcing progress. The need for enforcing progress has been recognized early and several solutions have been proposed to extend TA with such requirements, including location invariants, urgency and transition deadlines. Progress enforcement will be added to TA's and discussed in Section 6.

Priorities: We only consider here static priorities. The definition of TA's is extended with a partial irreflexive, asymmetric and transitive priority relation among transitions. The semantics is updated accordingly: a transition can only be taken when no transition with higher priority can be taken at the same instant.

Static priorities do not add expressiveness to TA's. If t_1, with guard g_1, has lower priority than t_2, with guard g_2, then it suffices to replace g_1 by $g_1 \wedge \neg g_2$. Negating a guard may introduce disjunctions, but these can be removed by distributing the components of the disjunction among several transitions, the transformation preserves strong timed bisimilarity. Composition and analysis of TA with priorities raise specific issues out of the scope of this paper.

4 Time Petri Nets with Priorities

$PrTPN$'s extend TPN's with a priority relation on transitions. Since we want to discuss bisimulations, we also add an alphabet of actions and a labeling function for transitions. $\mathbf{I}^+$ is the set of nonempty real intervals with nonnegative rational end-points. For $i \in \mathbf{I}^+$, $\downarrow i$ denotes its left end-point, and $\uparrow i$ its right end-point (if i bounded) or ∞. For any $\theta \in \mathbf{R}^+$, $i \dot{-} \theta$ denotes the interval $\{x-\theta | x \in i \wedge x \geq \theta\}$.

Definition 3. *A* Time Petri Net with Priorities *(or PrTPN for short) is a tuple* $\langle P, T, \mathbf{Pre}, \mathbf{Post}, m^0, Is, Pr, \Sigma^\epsilon, L\rangle$ *in which:*

– $\langle P, T, \mathbf{Pre}, \mathbf{Post}, m^0\rangle$ *is a Petri net, with places* P*, transitions* T*, initial marking* $m^0 : P \rightarrow \mathbf{N}^+$ *and* $\mathbf{Pre}, \mathbf{Post} : T \rightarrow P \rightarrow \mathbf{N}^+$*,*
– $Is : T \rightarrow \mathbf{I}^+$ *is a function called the* Static Interval *function,*
– $Pr \subseteq T \times T$ *is the Priority relation, irreflexive, asymmetric and transitive,*
– Σ *is a finite set of* Actions*, or* Labels*, not containing the* Silent *action* ϵ*,*
– $L : T \rightarrow \Sigma^\epsilon$ *is a function called the* Labeling *function.*

For $f, g : P \rightarrow \mathbf{N}^+$, $f \geq g$ means $(\forall p \in P)(f(p) \geq g(p))$ and $f\{+|-\}g$ maps $f(p)\{+|-\}g(p)$ with every p. A marking is a function $m : P \rightarrow \mathbf{N}^+$, $t \in T$ is *enabled* at m iff $m \geq \mathbf{Pre}(t)$, $\mathcal{E}_N(m)$ denotes the set of transitions enabled at m in net N. $(t_1, t_2) \in Pr$ is written $t_1 \succ t_2$ or $t_2 \prec t_1$ (t_1 has priority over t_2).

Definition 4. *The semantics of PrTPN* $\langle P, T, \mathbf{Pre}, \mathbf{Post}, m^0, Is, Pr, \Sigma, L\rangle$ *is the timed transition system* $\langle S, (m^0, Is^0), \Sigma, \rightarrow\rangle$ *where:*

– Is^0 *is function* Is *restricted to the transitions enabled at* m^0
– *the states of* S *are pairs* (m, I) *in which* m *is a marking and* $I : T \rightarrow \mathbf{I}^+$ *associates a time interval with every transition enabled at* m*,*
– $(m, I) \xrightarrow{L(t)} (m', I')$ *iff* $t \in T$ *and*
  1. $m \geq \mathbf{Pre}(t)$
  2. $0 \in I(t)$
  3. $(\forall k \in T)(m \geq \mathbf{Pre}(k) \wedge 0 \in I(k) \Rightarrow \neg(k \succ t))$
  4. $m' = m - \mathbf{Pre}(t) + \mathbf{Post}(t)$
  5. $(\forall k \in T)(m' \geq \mathbf{Pre}(k) \Rightarrow$
     $I'(k) = \textbf{ if } k \neq t \wedge m - \mathbf{Pre}(t) \geq \mathbf{Pre}(k) \textbf{ then } I(k) \textbf{ else } Is(k))$
– $(m, I) \xrightarrow{d} (m, I')$ *iff* $(\forall k \in T)(m \geq \mathbf{Pre}(k) \Rightarrow d \leq \uparrow I(k) \wedge I'(k) = I(k) \dot{-} d)$

Transition t may fire from (m, I) if t is enabled at m, firable instantly, and no transition with higher priority satisfies these conditions. In the target state, the transitions that remained enabled while t fired (t excluded) retain their intervals, the others are associated with their static intervals. A continuous transition by d is possible iff d is not larger than any $\uparrow I(t)$.

Boundedness: A PN is *bounded* if the marking of each place is bounded, boundedness implies finiteness of the set of reachable markings. Boundedness is undecidable for TPN's, and thus for $PrTPN$'s, but there are a number of decidable sufficient conditions for this property [5]. All nets considered in this paper are assumed bounded.

Product of PN, TPN, $PrTPN$: A product construction for labeled Petri nets has been used in many works, a definition appeared e.g. in [12]. It can be seen as a generalization to concurrent systems of the product of automata.

Definition 5. *Given a PN $\langle P, T, \mathbf{Pre}, \mathbf{Post}, m^0\rangle$ and $E \subseteq T$, the* product *of the transitions in E is the transition t such that, for any $p \in P$:*

$$\mathbf{Pre}(t)(p) = \sum_{k \in E} \mathbf{Pre}(k)(p) \quad \textit{and} \quad \mathbf{Post}(t)(p) = \sum_{k \in E} \mathbf{Post}(k)(p)$$

Definition 6. *Consider two labeled PN's not sharing any place or transition $N_1 = \langle P_1, T_1, \mathbf{Pre}_1, \mathbf{Post}_1, m_1^0, \Sigma_1^\epsilon, L_1\rangle$, $N_2 = \langle P_2, T_2, \mathbf{Pre}_2, \mathbf{Post}_2, m_2^0, \Sigma_2^\epsilon, L_2\rangle$. The product $N = N_1||N_2$ of N_1 and N_2 can be built as follows:*

- *Start with N made of the union of nets N_1 and N_2, after having removed from each the transitions labeled on $\Sigma_1 \cap \Sigma_2$,*
- *Then, for each pair $(t_1, t_2) \in T_1 \times T_2$ such that $L_1(t_1) = L_2(t_2) \neq \epsilon$, add to N a transition defined as the product of t_1 and t_2, inheriting their label.*

Product $||$ can be extended to $PrTPN$'s by specifying the static interval of a synchronized pair of transitions (t_1, t_2) as $Is_1(t_1) \cap Is_2(t_2)$, assuming it nonempty, and defining Pr from the local priority relations as follows:

- Let $R = \{(x, y) \in T \times T | (\exists (t, t') \in Pr_1 \cup Pr_2)(x \in S(t) \wedge y \in S(t'))\}$ where, for any $t \in T_1 \cup T_2$, $S(t)$ is either $\{t\}$ if t is not synchronized, or the set of transitions of T obtained as a product involving t otherwise (see Def. 6),
- Pr is the transitive closure of R, assuming it asymmetric.

Product $||$ is commutative and associative. For Petri nets, it is compositional: the transition system denoted by $N_1||N_2$ is the product of those denoted by N_1 and N_2. For TPN's or $PrTPN$'s, compositionality does not hold in general, but it holds when all synchronized transitions have interval $[0, \infty[$ and no pair of non-synchronized transitions is in the priority relation. The latter condition implies that Pr exactly coincides with relation R above. Compositionality in this special case is expressed by Theorem 1, its proof is given in the Appendix.

Theorem 1 (Restricted compositionality of product $||$ for $PrTPN$'s)

Let $\lceil n \rceil$ *be the TTS associated with PrTPN* n,
$N_1 = \langle P_1, T_1, \mathbf{Pre}_1, \mathbf{Post}_1, m_1^0, Is_1, Pr_1, \Sigma_1^\epsilon, L_1\rangle$
$N_2 = \langle P_2, T_2, \mathbf{Pre}_2, \mathbf{Post}_2, m_2^0, Is_2, Pr_2, \Sigma_2^\epsilon, L_2\rangle$
be two labeled PrTPN's with disjoint sets of places and transitions,
$N = \langle P, T, \mathbf{Pre}, \mathbf{Post}, m^0, Is, Pr, \Sigma^\epsilon, L\rangle$ *be their product by* $||$,
$T_{i\backslash j}$ *be the subset of transitions of* T_i *labeled over* $\Sigma_i^\epsilon \setminus \Sigma_j$
$T_{1\times 2} = T \backslash (T_{1\backslash 2} \cup T_{2\backslash 1})$

Then $(\forall t \in T_{1\backslash 2})(Is_1(t) = [0, \infty[) \wedge (\forall t \in T_{2\backslash 1})(Is_2(t) = [0, \infty[)$
$\wedge \; Pr \cap (T_{1\backslash 2} \times T_{2\backslash 1}) = Pr \cap (T_{2\backslash 1} \times T_{1\backslash 2}) = \emptyset$
$\Rightarrow \lceil N_1 \;||\; N_2 \rceil = \lceil N_1 \rceil \;||\; \lceil N_2 \rceil$

The following Theorem will be needed in Section 5, it follows from Def. 4:

Theorem 2. *Assume PrTPN N has two transitions t_1 and t_2 unrelated by the priority relation, both with interval $[0, \infty[$, and not sharing any input place. Then adding to N the transition t defined as the product of t_1 and t_2 does not change its state space and t can fire exactly at the times at which both t_1 and t_2 can.*

The power of priorities: Contrarily to TA's, priorities add expressiveness to bounded TPN's. To illustrate this, consider the TA and nets in Figure 1.

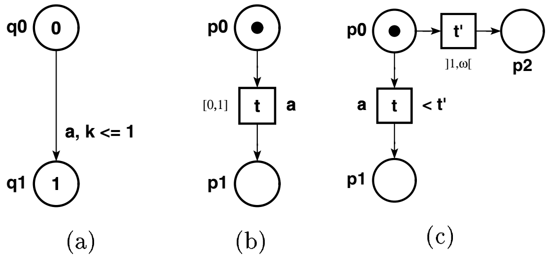


Fig. 1. Expressiveness of priorities in TPN's

As mentioned in [3], no TPN is, even weakly, timed bisimilar with TA 1(a). In particular, the TPN 1(b) is not: when at location q_0 in the TA, time can elapse of an arbitrary amount, while time cannot progress beyond 1 in TPN 1(b). Consider now the $PrTPN$ 1(c). Priorities are specified by transition annotations, we have here $t \prec t'$. Transition t' is silent and firable at any time greater than 1, transition t bears label a and interval $[0, \infty[$. Transition t may fire at any time less than or equal to 1, but not later, since t' is then firable and it has priority over t. Indeed, $PrTPN$ 1(c) is weakly timed bisimilar with TA 1(a).

Priorities do not forbid time to elapse, but enrich firing constraints for transitions. It is shown in the next Section that the above trick generalizes to any TA. Addition of progress requirements will be addressed in Section 6.

5 Encoding Guards

5.1 Notations

Let $\mathcal{A} = \langle Q, q^0, X, \Sigma^\epsilon, T \rangle$ be some TA. Without loss of generality, it is assumed that every clock in X is involved in some guard, that every clock which is reset in some transition is used in some guard, and that $\mathcal{A}$ is not expressed as a product.

- For each transition $t \in T$, let:
  - σ_t be the action of $\Sigma \cup \{\epsilon\}$ associated with t,
  - X_t be the set of clocks involved in transition t (in its guard or reset),

  - E_t^k, for $k \in X_t$, be the set of atomic guards in the guard of t involving clock k, augmented with $(k := 0)$ if t resets k,
  - $E_t = \sigma_t \cup \bigcup_{k \in X_t} \{E_t^k\}$, E_t holds action σ_t and all sets E_t^k for $k \in X_t$.
- For each clock $k \in X$, let G_k be the set of atomic guards in $\mathcal{A}$ involving k.

5.2 Encoding Atomic Clock Guards and Resets

Each atomic guard will be modeled by a *PrTPN*, Figure 2 shows the four required models for clock k and constant $c = 5$. The upper-left (resp. lower-left) net models $k \leq 5$ (resp. $k < 5$), the upper-right (resp. lower-right) net models $k \geq 5$ (resp. $k > 5$). Priorities are specified by transition annotations, e.g. in the upper-left net, we have $C \prec B$, $A \prec B$ and $A \prec Z$. The transitions are either unlabeled (implicitly labeled with silent action ϵ), or carry a label made of a clock constraint and/or a reset.

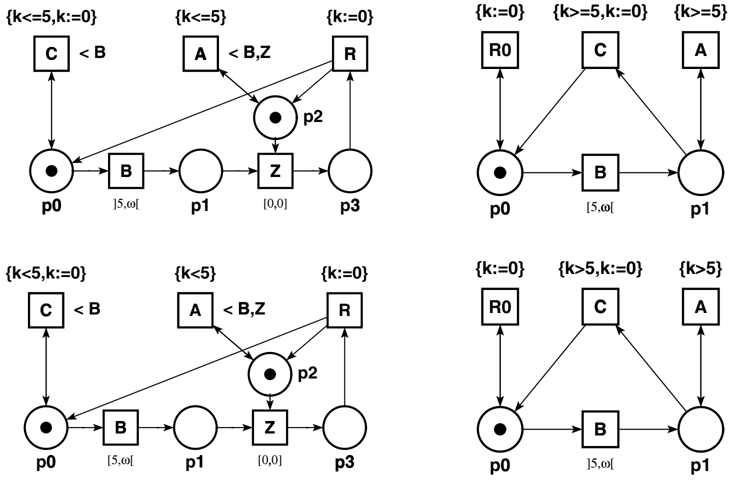


Fig. 2. *PrTPN*'s encoding atomic guards and resets for clock k and constant $c = 5$

Theorem 3. *Let k be the time elapsed since the initial markings of the nets in Figure 2 were last established, then, for each of these nets:*

- *the transitions whose label includes a condition on k are firable exactly at the times at which that condition holds,*
- *all transitions whose label includes $k := 0$ restore the initial state of the net,*
- *from any state a sequence of duration 0 firing a transition whose label includes $k := 0$ is firable*

Proof. For net $k \leq 5$: If $k \leq 5$, C and A may fire, firing C restores the initial state, firing A preserves the current state. At any $k > 5$, B has priority over A and C, B may be arbitrarily delayed. After B fired, A and Z are enabled but Z has higher priority, and time cannot elapse since Z carries $[0, 0]$. After Z fired, only R may fire, restoring the initial state. The proof is similar for net $k < 5$. The nets for $k \geq 5$ and $k > 5$ are self-explanatory, they do not use priorities.

5.3 Encoding a Clock

For each $k \in X$, let N_k be the net built as follows:

1. Assume $\{c_1, \ldots, c_n\}$ is the set of nets encoding the guards in G_k. Let K be the product $(c_1 \backslash F \mid\mid \ldots \mid\mid c_n \backslash F) \backslash H$, with relabelings F and H as follows:
   - F relabels any transition whose label includes $k := 0$ with ρ (ρ new),
   - H relabels any t obtained from a product of transitions by the union of their labels in nets c_i.
2. Next, starting with $N_k = K$, add to N_k, for each $E \subseteq G_k$ with $card(E) > 1$, a transition labeled E defined as the product (Def. 6) of all transitions of K with their label intersecting E.

The first step synchronizes resets among all the nets encoding atomic guards for clock k. Step 2 adds transitions checking all possible conjunctions of atomic guards for k (without reset). Net N_k has as transitions:

- The transitions not labeled (internal) in the component nets,
- For each nonempty $E \subseteq G_k$, a transition labeled with set E,
- For each (possibly empty) $E \subseteq G_k$, a transition labeled $E \cup \{k := 0\}$.

Theorem 4. *Let k be the time elapsed since the initial marking of net N_k was last established, then:*

- *the transitions whose label includes a condition (atomic or compound) are firable exactly at the times at which that condition holds,*
- *all transitions whose label includes $k := 0$ restore the initial state of the net,*
- *from any state a sequence of duration 0 firing a transition whose label includes $k := 0$ is firable*

Proof. Follows from Theorems 1, 2 and 3, and definitions of relabelings F and H. Note that all assumptions required by Theorem 1 hold.

5.4 Encoding the Timed Automaton $\mathcal{A}$

The *PrTPN* $\mathcal{N}$ encoding $\mathcal{A} = \langle Q, q^0, X, \Sigma^\epsilon, T \rangle$ is obtained as follows.

1. Let N_A be the net built as follows:
   - For each $q \in Q$ add a new place to N_A, and mark the place encoding q^0,
   - For each transition $q \xrightarrow{t} q'$ of $\mathcal{A}$, add to N_A a transition between the places encoding q and q', labeled by E_t (as computed in Section 5.1).
2. Next, let N_K be the net built as follows:
   - Start with $N_K = ||_{k \in X}\ N_k$ where N_k encodes clock k (see Section 5.3),
   - Then, for each E_t, add to N_K a transition labeled E_t defined as the product of all transitions of N_K with their label belonging to E_t,
   - Remove all labeled transitions of N_K whose label is not in any E_t,

3. Finally, let $\mathcal{N} = N_A \mid\mid N_K$ and relabel each labeled transition by the action from Σ^ϵ belonging to its label.

Intuitively, N_A after step 1 is a copy of the underlying automaton of $\mathcal{A}$, with each transition labeled by the set of atomic guards of the corresponding transition of A, augmented with resets, and partitioned per clock. N_K at step 2 is the product of the nets encoding clocks, augmented with transitions checking all guards of $\mathcal{A}$ and performing resets if needed. The transitions removed at step 2 correspond to combinations of atomic guards not used in $\mathcal{A}$ or to transitions resetting clocks which are never reset in $\mathcal{A}$. Step 3 restores the labeling of $\mathcal{A}$.

Theorem 5. *The above TA $\mathcal{A}$ and PrTPN $\mathcal{N}$ are weakly timed bisimilar.*

Proof. Follows again from the properties of the operations used to built the net and of those the nets being composed. The nets N_k encoding clocks do not share any label. After step 2, and from Theorems 2 and 4, the labeled transitions of N_K are firable exactly when the conjunction of the constraints they are labeled with hold. At step 3, guard checks and resets are ordered like in $\mathcal{A}$.

Theorem 6. *Any TA can be encoded into a PrTPN without right-open intervals, preserving weak timed bisimilarity, or:*

TA $\lesssim_{\mathcal{W}}$ PrTPN with unbounded or right-closed intervals.

Proof. The encoding applied to $\mathcal{A}$ is applicable to any TA (without progress requirements). Further, it produces $PrTPN$'s without right-open intervals.

6 Encoding Time Automata Invariants

6.1 Expressing Progress Requirements

Enforcing progress conditions in Timed Automata is undoubtedly necessary for modeling systems with hard time constraints, but there is no consensus yet about how to express them. Progress requirements are most often expressed using location invariants or transition deadlines. Location invariants (as e.g. in [1]) are compositional but may introduce timelocks (sink states resulting from elapsing of time), transition deadlines [7] avoid timelocks but break compositionality [10]. We will concentrate here on invariants, leaving alternatives for further work.

Definition 7. *A TA with Invariants is a tuple $\langle Q, q^0, X, \Sigma^\epsilon, T, I\rangle$ in which:*

- *$\langle Q, q^0, X, \Sigma^\epsilon\rangle$ is a TA, as in Definition 1,*
- *$I : Q \rightarrow \mathcal{C}(X)$ maps a clock constraint with every location. These constraints are typically restricted to conjunctions of constraints of form $k \leq c$ or $k < c$.*

Definition 8. *The semantics of TA with invariants only differs by the rule concerning continuous transitions (see Definition 2), which is replaced by:*

- *$(q, v) \xrightarrow{d} (q, v + d)$ iff $d \in \mathbf{R}^+ \wedge (\forall d')(0 \leq d' \leq d \Rightarrow I(q))$*

Time may elapse at some location as long as the invariant at that location holds. Some variations of this rule are found in the literature, as well as further conditions related to progress and not addressed here, such as non-Zenoeness.

The class of TA with invariants as above will be noted $TA + \{\leq, <, \wedge\}$, $TA + \{\leq, \wedge\}$ is its subclass in which no invariant constraint is strict, TA is the class with no invariants.

6.2 Encoding TA+$\{\leq, \wedge\}$

Adapting the translation of invariants in [3], our encoding of TA in Section 5 can be extended to handle invariants with non strict constraints.

For encoding the effects of invariant $k_1 \leq c_1 \wedge \ldots \wedge k_n \leq c_n$ at location q_i we will monitor property $k_1 \geq c_1 \vee \ldots \vee k_n \geq c_n$ when the place p_i materializing location q_i in net N_A (see Section 5.4) is marked, and prevent time to progress as soon as the property holds.

As for guards, a $PrTPN$ will be associated with each atomic constraint to be monitored. Because of the above "no-delay" requirement, we cannot use for this the nets implementing guard check $k \geq c$ (Figure 2), we will use instead nets like the one in Figure 3 (this net could be used to check $k \geq 5$ guards as well, though). Note that the transition that checks $k \geq 5$ in that net carries a label distinguishing it from the transition realizing guard check $k \geq 5$. Of course, these monitoring $PrTPN$'s, for each clock k, must have their reset transitions synchronized with those of net N_k implementing clock k (see Section 5.3).

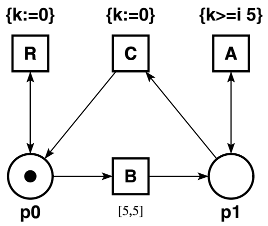


Fig. 3. $PrTPN$ encoding monitor for constraint $k \geq 5$

The encoding of invariants is sketched in Figure 4. For checking an atomic invariant, say $k \leq 5$, at location q_i, a loop will be added to place p_i in net N_A. The transition in this loop will be synchronized with the transition labeled $k \geq i$ 5 in the net monitoring $k \geq 5$, shown Figure 3. After synchronization, the transition will be made "urgent" by assigning it firing interval $[0, 0]$. Consequently, when p_i is marked, time elapsing is prevented as soon as $k = 5$.

For checking compound invariants, one such loop is added for each constraint to be monitored, as shown in Figure 4. The process is repeated for each location carrying an invariant. The constructions of Sections 5 are easily extended to encode invariants that way, details are omitted.

The above encoding does not extend to invariants with strict constraints, unfortunately (other solutions might possibly work, though). At that time, with

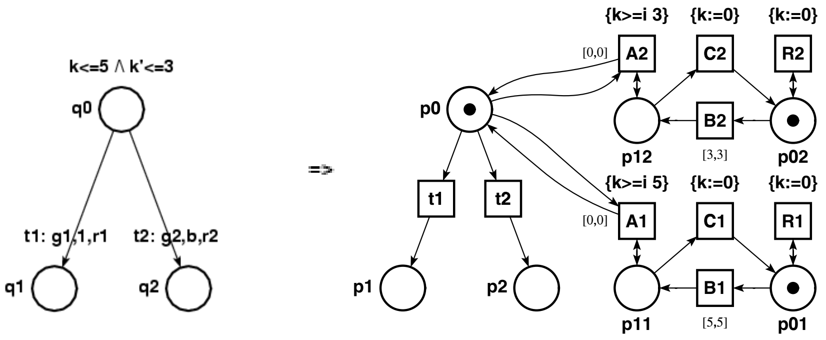


Fig. 4. Checking conjunctions of $k \leq c$ invariants

the above encoding of invariants, we have $TA+\{\leq, \wedge\} \lesssim_{\mathcal{W}} PrTPN$. Observing that encoding such extended TA's yields $PrTPN$'s without right-open intervals, this result can be strengthened to:

Theorem 7. *$TA+\{\leq, \wedge\} \lesssim_{\mathcal{W}} PrTPN$ with unbounded or right-closed intervals*

6.3 From *PrTPN* to *TA*

Some equivalences, rather than orderings, can be obtained by adapting the translation of [9] of bounded TPN's (without priorities) into TA's.

[9] formalizes the idea that (bounded) TPN's can be encoded into TA's with invariants using one clock per transition of the TPN. Each transition is encoded into a timed automaton mimicking its "behavior": check for enabledness, removal of input tokens and addition of output tokens. The TPN is encoded as a composition of the automata modeling its transitions and of a supervisor TA sequencing the above operations for all transitions simultaneously. "earliest firing time" constraints of the TPN are encoded into guard constraints of form $k \geq c$ and "latest firing time" constraints into invariants of form $k \leq c$[1]. Though the method is defined for TPN's with closed or left-closed unbounded intervals, it is applicable unchanged to arbitrary TPN's.

The method can be adapted to encode $PrTPN$'s into TA's with priorities: It suffices to assign to the transitions labeled *?pre* in their encoding the priorities assigned to the corresponding transitions of the net. Now, as noted in Section 3, $PrTA \approx_{\mathcal{W}} TA$, hence, for Bounded $PrTPN$'s:

Theorem 8. *$PrTPN \lesssim_{\mathcal{W}} TA + \{\leq, <, \wedge\}$.*

Theorem 9. *$TA+\{\leq, \wedge\} \approx_{\mathcal{W}} PrTPN$ with right-closed or unbounded intervals*

Proof. From such restricted $PrTPN$'s, [9] (adapted for $PrTPN$'s) would yield TA's with invariants built from $\{\leq, \wedge\}$. From such TA's, our encoding yields $PrTPN$'s in which all transitions have unbounded or right-closed firing intervals.

As corollaries concerning Bounded TPN's, we obtain (proofs omitted):

1 [9] adds these constraints to guards too, but this is not necessary.

Corollary 1. *TA with guards built from $\{\geq, >, \wedge\}$ $\approx_{\mathcal{W}}$ TPN with unbounded intervals*

Corollary 2. *TA+$\{\leq, \wedge\}$ with guards built from $\{\geq, >, \wedge\}$ $\approx_{\mathcal{W}}$ TPN with right-closed or unbounded intervals.*

7 Conclusion

Summary: The now available comparison results are shown in the Table below, for Bounded $PrTPN$'s and TPN's. These results hold for safe (1-bounded) nets too. They cannot be strengthened to strong bisimulation or preorders results, due to the necessary internal transitions in the TA translations of Section 5 and the TPN translations of [9]. No previous work compared TA's with $PrTPN$'s. For TPN's, Corollary 2 strengthens Corollaries 5 and 6 in [3].

	TA guards	invariants		TPN intervals	priorities
Theorem 7	$\leq < \geq > \wedge$	$\leq \wedge$	$\lesssim_{\mathcal{W}}$	$(]a\vert[a),(b]\vert\infty[)$	Y
Theorem 8	$\leq < \geq > \wedge$	$\leq < \wedge$	$\gtrsim_{\mathcal{W}}$	$(]a\vert[a),(b]\vert b[\vert\infty[)$	Y
Theorem 9	$\leq < \geq > \wedge$	$\leq \wedge$	$\approx_{\mathcal{W}}$	$(]a\vert[a),(b]\vert\infty[)$	Y
[9]	$\geq > \wedge$	$\leq < \wedge$	$\gtrsim_{\mathcal{W}}$	$(]a\vert[a),(b]\vert b[\vert\infty[)$	N
Corollary 2	$\geq > \wedge$	$\leq \wedge$	$\approx_{\mathcal{W}}$	$(]a\vert[a),(b]\vert\infty[)$	N
Corollary 1	$\geq > \wedge$	$\emptyset$	$\approx_{\mathcal{W}}$	$(]a\vert[a),\infty[$	N

These results improve understanding of the differences between two of the most widely used models of realtime systems. Theorem 9 suggests that, augmenting TPN's with priorities, these differences are small, if any.

They also promise sharing of analysis methods for these two models, at the time rather complementary: Analysis methods for TPN's mostly focused on time abstracting representations, preserving markings and LTL properties [5], states and LTL [6], or states and CTL (time abstracting bisimulations) [6] . Analysis methods for TA's focused more on model-checking of "timed" temporal properties such as those expressible in $TCTL$. Theorem 8 and [9] allow to use for $PrTPN$'s or TPN's analysis methods developed for TA's. Theorem 7 allows the converse for a large subclass of TA's, provided analysis methods for TPN's can be extended to $PrTPN$'s.

Further work: For easing proofs, the encoding of guards presented in Section 5 is rather brute force. It should be convenient in a number of cases, but it yields $PrTPN$'s with many more transitions than the TA, in general. For practical purposes, clock models could be made smaller. Also, *read-arcs*, a well known extension of Petri nets consisting of special arcs that test boolean conditions on markings but do not transfer tokens, would significantly compact encodings. Development of such improved encodings is left as further work.

To apply TPN-style analysis methods to TA's through the encodings explained, these methods should be extended to cope with priorities. Extending to $PrTPN$'s the methods building state space abstractions preserving LTL properties in [5,6] should be straightforward, it resumes to take into account the extra constraints brought by priorities when developing the state class graph, these constraints are linear. Extending the methods building time abstracting bisimulations [6] is more subtle as continuous transitions cannot all be merged with discrete transitions anymore, but we are confident that they can be adapted.

Finally, encoding of timed automata with alternative proposals for progress requirements needs be investigated.

References

1. R. Alur. Timed automata. In *11th International Conference on Computer Aided Verification, CAV'99, Springer LNCS 1633*, pages 8–22, July 1999.
2. R. Alur and D.L. Dill. A theory of timed automata. *TCS*, 126:183–235, 1994.
3. B. Bérard, F. Cassez, S. Haddad, O. H. Roux, and D. Lime. Comparison of the Expressiveness of Timed Automata and Time Petri Nets. In *Formal Modeling and Analysis of Timed Systems (FORMATS'05), LNCS 3829*, pages 211–225, 2005.
4. B. Berthomieu and M. Diaz. Modeling and verification of time dependent systems using time Petri nets. *IEEE Trans. on Software Engineering*, 17(3):259–273, 1991.
5. B. Berthomieu and M. Menasche. An enumerative approach for analyzing time Petri nets. *IFIP Congress Series*, 9:41–46, 1983.
6. B. Berthomieu and F. Vernadat. State class constructions for branching analysis of time Petri nets. In *Proc. Tools and Algorithms for the Construction and Analysis of Systems, Springer LNCS 2619*, 2003.
7. S. Bornot, J. Sifakis, and S. Tripakis. Modeling urgency in timed systems. In *Compositionality: The Significant Difference – International Symposium COMPOS'97, Springer LNCS 1536*, pages 103–129, September 1997.
8. P. Bouyer, S. Haddad, and P-A. Reynier. Extended timed automata and time Petri nets. In *Proc. of 6th Int. Conf. on Application of Concurrency to System Design (ACSD'06)*, Turku, Finland, June 2006. IEEE Computer Society Press.
9. F. Cassez and O. H. Roux. Structural translation from time petri nets to timed automata. *Journal of Systems and Software*, 2006. forthcoming.
10. P. R. D'Argenio and B. Gebremichael. The coarsest congruence for timed automata with deadlines contained in bisimulation. In *CONCUR 2005 - Concurrency Theory, Springer LNCS 3653*, pages 125–140, August 2005.
11. S. Haar, L. Kaiser, F. Simonot-Lion, and J. Toussaint. Equivalence of Timed State Machines and safe Time Petri Nets. In *Proc. of the 6th Workshop on Discrete Events Systems (WODES'02)*, pages 119–124, Zaragoza, Spain, October 2002.
12. M. Hack. Petri net languages. TR 159, MIT, Cambridge, Mass., 1976.
13. K. G. Larsen, P. Pettersson, and W. Y. Yi. Model-checking for real-time systems. In *Fundamentals of Computation Theory, LNCS 965*, pages 62–88, August 1995.
14. D. Lime and O. H. Roux. State class timed automaton of a time Petri net. In *10th International Workshop on Petri Nets and Performance Models, (PNPM'03)*, pages 124–133, Urbana, USA, September 2003. IEEE Computer Society.
15. P. M. Merlin and D. J. Farber. Recoverability of communication protocols: Implications of a theoretical study. *IEEE Tr. Comm.*, 24(9):1036–1043, Sept. 1976.

A Proof of Theorem 1

Let $f[E]$ be the restriction of function f to its domain intersected with E, $f \oplus g$ be the function that behaves like f on the domain of f or like g otherwise, I^∞ be the function that associates interval $[0,\infty[$ with any transition.

It directly follows from Definition 4 that if some transition has static firing interval $[0,\infty[$, then its interval in any state of the net is $[0,\infty[$. Hence we have:

Lemma 1. (i) $(\forall(m,I) \in \lceil N_1 \mid\mid N_2 \rceil)(\forall t \in T_{1\times 2})(I(t) = [0,\infty[)$
(ii) $(\forall(m_i,I_i) \in \lceil N_i \rceil)(\forall t \in \mathcal{E}_N(m_i) \backslash T_{i\backslash j})(I_i(t) = [0,\infty[)$

Next, consider the mapping $\phi : \lceil N_1 \mid\mid N_2 \rceil \rightarrow \lceil N_1 \rceil \mid\mid \lceil N_2 \rceil$ defined by:

$$\begin{aligned}\phi(m,I) = {} & (m_1, I[T_{1\backslash 2}] \oplus I^\infty[\mathcal{E}_{N_1}(m_1) \backslash T_{1\backslash 2}])) \text{ where } m_1 = m[P_1] \\ & \mid\mid (m_2, I[T_{2\backslash 1}] \oplus I^\infty[\mathcal{E}_{N_2}(m_2) \backslash T_{2\backslash 1}])) \text{ where } m_2 = m[P_2]\end{aligned}$$

By Lemma 1 and because N_1 and N_2 are disjoint, ϕ is injective. We have:

$$\begin{aligned}\phi^{-1}((m_1,I_1) \mid\mid (m_2,I_2)) = {} & (m, I_1[T_{1\backslash 2}] \oplus I^\infty[\mathcal{E}_N(m) \cap T_{1\times 2}] \oplus I_2[T_{2\backslash 1}]) \\ & \text{where } m = m_1[P_1] \oplus m_2[P_2]\end{aligned}$$

We show by induction that the pair (ϕ, id), where id is the identity mapping over $\Sigma_1 \cup \Sigma_2 \cup \{\epsilon\} \cup \mathbf{R}^+$, is a graph isomorphism between $\lceil N_1 \mid\mid N_2 \rceil$ and $\lceil N_1 \rceil \mid\mid \lceil N_2 \rceil$:

(i) $s_1^0 \mid\mid s_2^0 = \phi(s^0)$, where s^0 and the s_i^0 are the initial states of $\lceil N \rceil$ and $\lceil N_i \rceil$: By definition, $s^0 = (m^0, Is[\mathcal{E}_N(m^0))$ and $s_i^0 = (m_i^0, Is_i[\mathcal{E}_{N_i}(m_i^0)])$. From the assumptions, $m_i^0 = m^0[P_i]$ and $I_i^0 = I^0[T_{i\backslash j}] \oplus I^\infty[\mathcal{E}_{N_i}(m_i^0) \backslash T_{i\backslash j}]$‘ <, hence $s_1^0 \mid\mid s_2^0 = \phi(s_0)$.

(ii) $(\forall a)(\forall s, s' \in \lceil N \rceil)(s \xrightarrow{a} s' \Leftrightarrow \phi(s) \xrightarrow{a} \phi(s'))$: Four cases must be considered:

1. $a \in R^+$: Let $(m,I) = s$ and $(m_1,I_1) \mid\mid (m_2,I_2) = \phi(s)$.
From Definition 4, we have $(m,I) \xrightarrow{a} (m,I')$ iff:
(a) $(\forall t \in \mathcal{E}_N(m))(a \leq \uparrow I(t) \wedge I'(t) = I(t) \dot{-} a)$
From the definition of the TTS product, we have $(m_1,I_1) \mid\mid (m_2,I_2) \xrightarrow{a} (m_1,I_1') \mid\mid (m_2,I_2')$ iff $(m_1,I_1) \xrightarrow{a} (m_1,I_1')$ and $(m_2,I_2) \xrightarrow{a} (m_2,I_2')$, or:
(b1) $(\forall t \in \mathcal{E}_{N_1}(m_1))(a \leq \uparrow I_1(t) \wedge I_1'(t) = I_1(t) \dot{-} a) \wedge$
(b2) $(\forall t \in \mathcal{E}_{N_2}(m_2))(a \leq \uparrow I_2(t) \wedge I_2'(t) = I_2(t) \dot{-} a)$
By Lemma 1, (a) (resp. bi) is trivially satisfied for the transitions in $T_{1\times 2}$ (resp. $\mathcal{E}_{N_i}(m_i) \backslash T_{i\backslash j}$). Further, from ϕ, $m_i = m[P_i]$ and I agrees with I_1 (resp. I_2) on the transitions of $T_{1\backslash 2}$ (resp. $T_{2\backslash 1}$), hence (a) $\Leftrightarrow$ (b1) $\wedge$ (b2) and $(m_1,I_1') \mid\mid (m_2,I_2') = \phi(m,I')$.
2. $a \in \Sigma_1 \cap \Sigma_2$: Let $(m,I) = s$ and $(m_1,I_1) \mid\mid (m_2,I_2) = \phi(s)$.
We have $(m,I) \xrightarrow{L(t)} (m',I')$ iff $c_N^1 \wedge \ldots \wedge c_N^5$, where conditions c_N^i are those in Definition 4 for discrete transitions, specialized for N. On the other hand, from the definition of the TTS product, we have $(m_1,I_1) \mid\mid (m_2,I_2) \xrightarrow{a} (m_1',I_1') \mid\mid (m_2',I_2')$ iff $(m_1,I_1) \xrightarrow{a} (m_1',I_1')$ by some t_1 and $(m_2,I_2) \xrightarrow{a} (m_2',I_2')$ by some t_2, that is if $(c_{N_1}^1 \wedge \ldots \wedge c_{N_1}^5) \wedge (c_{N_2}^1 \wedge \ldots \wedge c_{N_2}^5)$. As $a \in \Sigma_1 \cap \Sigma_2$, there must be t, t_1 and t_2 such that t is the product of t_1 and t_2.

Then, we have $c^1_N \Leftrightarrow c^1_{N_1} \wedge c^1_{N_2}$ from ϕ and the definitions of products, $c^2_N \Leftrightarrow c^2_{N_1} \wedge c^2_{N_2}$ by Lemma 1, and $c^3_N \Leftrightarrow c^3_{N_1} \wedge c^3_{N_2}$ by the properties of Pr: t is preempted in N iff either t_1 is preempted in $N1$ or t_2 in $N2$. Finally, it is easily shown that $(m_i, I_i) \xrightarrow{a} (m'[P_i], I'[T_{i\backslash j}] \oplus I^{\infty}[\mathcal{E}_{N_i}(m'_i[P_i])\backslash T_{i\backslash j}]$, hence $(m'_1, I'_1) \mid\mid (m'_2, I'_2) = \phi(s')$.

3. $a \in \Sigma^{\epsilon}_1 \setminus \Sigma_2$: Let $(m, I) = s$ and $(m_1, I_1) \mid\mid (m_2, I_2) = \phi(s)$.
   We have $(m, I) \xrightarrow{a} (m', I')$ by t iff $c^1_N \wedge \ldots \wedge c^5_N$. From the definition of products, we have $(m_1, I_1) \mid\mid (m_2, I_2) \xrightarrow{a} (m'_1, I'_1) \mid\mid (m_2, I_2)$ iff, by the previous t, $(m_1, I_1) \xrightarrow{a} (m'_1, I'_1)$, or $c^1_{N_1} \wedge \ldots \wedge c^5_{N_1}$. Then, as for case 2, we have that the preconditions for both transitions are equivalent and that the target states obey $(m'_1, I'_1) \mid\mid (m'_2, I'_2) = \phi(s')$.
4. for $a \in \Sigma^{\epsilon}_2 \setminus \Sigma_1$: like case 3, symmetrically.

Matching Scenarios with Timing Constraints*

Prakash Chandrasekaran and Madhavan Mukund

Chennai Mathematical Institute, Chennai, India
{prakash, madhavan}@cmi.ac.in

Abstract. Networks of communicating finite-state machines equipped with local clocks generate timed MSCs. We consider the problem of checking whether these timed MSCs are "consistent" with those provided in a timed MSC specification. In general, the specification may be both positive and negative. The system should execute all positive scenarios "sensibly". On the other hand, negative scenarios rule out some behaviours as illegal. This is more complicated than the corresponding problem in the untimed case because even a single timed MSC specification implicitly describes an infinite family of timed scenarios. We outline an approach to solve this problem that can be automated using Uppaal.

1 Introduction

In a distributed system, several agents interact with each other to generate a global behaviour. The interaction between these agents is usually described in terms of scenarios, using mechanisms such as use-cases and message sequence charts (MSCs) [8].

In general, scenarios could be of two types, positive and negative. Positive scenarios are those that the system is designed to execute—for instance, these may describe a handshaking protocol to set up a reliable communication channel between two hosts on a network. Negative scenarios indicate undesirable behaviours, such as a situation when both hosts independently initiate the activity of setting up a channel, leading to a collision.

This leads to a natural verification problem: given a distributed system and a scenario, does the system exhibit the scenario? In the context of message sequence charts, this is referred to as the scenario matching problem, for which efficient algorithms have been identified [10]. An approach to solve this problem using the modelchecker Spin was proposed in [4].

In this paper, we extend the study of scenario matching to timed systems. We consider communicating finite-state machines equipped with local clocks. Clock constraints are used to guard transitions and specify location invariants, as in other models of timed automata [3]. Just as the runs of timed automata can be described in terms of timed words, the interactions exhibited by communicating finite-state machines with clocks can be described using timed MSCs.

* Partially supported by *Timed-DISCOVERI*, a project under the Indo-French Networking Programme.

We define a version of scenarios with timing constraints that we call timed MSC templates. These templates are built from fixed underlying MSCs by associating a lower and upper bound on the time interval between certain pairs of events. Timed MSC templates are a natural and useful extension of the untimed notation for scenarios, because protocol specifications typically include timing requirements for message exchanges, as well as descriptions of how to recover from timeouts.

In general, a timed MSC template is compatible with infinitely many timed MSCs. Thus, the scenario matching problem is already more complicated than in the untimed case, where a single scenario describes exactly one pattern of interaction. In our setting, the scenario matching problem can be reformulated in terms of checking whether the intersection of two collections of timed MSCs is nonempty.

We propose an approach to tackle this problem using the modelchecking tool Uppaal, which is designed to verify properties of timed systems. Unfortunately, the basic system model of Uppaal consists of a network of timed automata that communicate via synchronous handshakes, rather than message-passing. We thus need to code up message-passing channels by creating special processes to model buffers. However, we can exploit the handshake mechanism to synchronize the system with the template to be verified. This automatically reduces the behaviours of the system to those that are consistent with the template. The scenario matching problem can then be easily transformed into a modelchecking question for Uppaal to verify on the composite system.

The paper is organized as follows. In the next two sections, we formally define timed MSCs and timed message-passing automata. This enables us to precisely define the scenario matching problem for timed systems in Section 4. In Section 5, we describe our approach to address the scenario matching problem in Uppaal. Next, we look at issues related to the expressiveness of our timed message-passing automaton model. In Section 7, we examine some optimizations that can be introduced when translating the scenario matching problem to Uppaal, to improve the efficiency of verification. We conclude with a brief discussion.

2 Timed MSCs

2.1 Message Sequence Charts

Let $\mathcal{P} = \{p, q, r, \ldots\}$ be a finite set of processes (agents) that communicate with each other through messages via reliable FIFO channels using a finite set of message types $\mathcal{M}$. For $p \in \mathcal{P}$, let $\Sigma_p = \{p!q(m), p?q(m) \mid p \neq q \in \mathcal{P}, m \in \mathcal{M}\}$ be the set of communication actions in which p participates. The action $p!q(m)$ is read as *p sends the message m to q* and the action $p?q(m)$ is read as *p receives the message m from q*. We set $\Sigma = \bigcup_{p\in\mathcal{P}} \Sigma_p$. We also denote the set of *channels* by $Ch = \{(p, q) \mid p \neq q\}$.

Labelled posets. A Σ-labelled poset is a structure $M = (E, \leq, \lambda)$ where $(E, \leq)$ is a poset and $\lambda : E \rightarrow \Sigma$ is a labelling function. For $e \in E$, let ${\downarrow}e = \{e' \mid e' \leq e\}$.

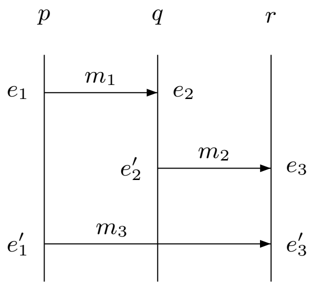


Fig. 1. An MSC over $\{p, q, r\}$

For $p \in \mathcal{P}$ and $a \in \Sigma$, we set $E_p = \{e \mid \lambda(e) \in \Sigma_p\}$ and $E_a = \{e \mid \lambda(e) = a\}$, respectively. For each $(p, q) \in Ch$, we define the relation $<_{pq}$ as follows:

$$e <_{pq} e' \iff \lambda(e) = p!q(m),\ \lambda(e') = q?p(m) \text{ and } |{\downarrow}e \cap E_{p!q(m)}| = |{\downarrow}e' \cap E_{q?p(m)}|$$

The relation $e <_{pq} e'$ says that channels are FIFO with respect to each message—if $e <_{pq} e'$, the message m read by q at e' is the one sent by p at e.

Finally, for each $p \in \mathcal{P}$, we define the relation $\leq_{pp} = (E_p \times E_p) \cap \leq$, with $<_{pp}$ standing for the largest irreflexive subset of $\leq_{pp}$.

Definition 1. *An MSC (over $\mathcal{P}$) is a* finite *Σ-labelled poset $M = (E, \leq, \lambda)$ that satisfies the following conditions.*

1. *Each relation $\leq_{pp}$ is a linear order.*
2. *If $p \neq q$ then for each $m \in \mathcal{M}$, $|E_{p!q(m)}| = |E_{q?p(m)}|$.*
3. *If $e <_{pq} e'$, then $|{\downarrow}e \cap \left(\bigcup_{m\in\mathcal{M}} E_{p!q(m)}\right)| = |{\downarrow}e' \cap \left(\bigcup_{m\in\mathcal{M}} E_{q?p(m)}\right)|$.*
4. *The partial order $\leq$ is the reflexive, transitive closure of the relation $\bigcup_{p,q\in\mathcal{P}} <_{pq}$.*

The second condition ensures that every message sent along a channel is received. The third condition says that every channel is FIFO across all messages.

In diagrams, the events of an MSC are presented in *visual order*. The events of each process are arranged in a vertical line and messages are displayed as horizontal or downward-sloping directed edges. Figure 1 shows an example with three processes $\{p, q, r\}$ and six events $\{e_1, e'_1, e_2, e'_2, e_3, e'_3\}$ corresponding to three messages—m_1 from p to q, m_2 from q to r and m_3 from p to r.

For an MSC $M = (E, \leq, \lambda)$, we let $\text{lin}(M) = \{\lambda(\pi) \mid \pi \text{ is a linearization of } (E, \leq)\}$. For instance, $p!q(m_1)\ q?p(m_1)\ q!r(m_2)\ p!r(m_3)\ r?q(m_2)\ r?p(m_3)$ is one linearization of the MSC in Figure 1.

2.2 Timed MSC Templates

A timed MSC template is an MSC annotated with time intervals between pairs of events along a process line. For instance, consider the interaction between a user, an ATM and a server depicted in Figure 2. This MSC has sixteen events

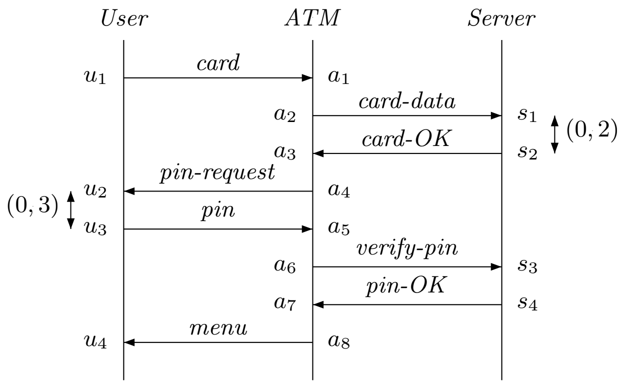


Fig. 2. A timed MSC template describing interaction with an ATM

generated by eight messages. The events u_2 and u_3 are linked by a time interval $(0, 2)$, as are the events s_2 and s_3. These time intervals represent constraints on the delay between the occurrences of the events. Thus, this template specifies that the server is expected to respond to a request to authenticate an ATM card within 2 units of time. Similarly, a user has to type in his PIN within 3 units of time of the ATM requesting the PIN.

For simplicity, we assume that time intervals are bounded by natural numbers. A pair of time points (m, n), $m, n \in \mathbb{N}$, $m \leq n$, denotes the time interval $\{x \in \mathbb{R}_{\geq 0} \mid m \leq x \leq n\}$.

Definition 2. *Let $M = (E, \leq, \lambda)$ be an MSC. An* interval constraint *is a tuple $\langle (e_1, e_2), (t_1, t_2) \rangle$, where:*

- *$e_1, e_2 \in E$ with $e_1 \leq_{pp} e_2$ for some $p \in \mathcal{P}$.*
- *$t_1, t_2 \in \mathbb{N}$ with $t_1 \leq t_2$.*

The restriction on the relationship between e_1 and e_2 ensures that an interval constraint is local to a process.

Definition 3. *A* timed MSC template *is pair $\mathcal{T} = (M, \mathcal{I})$ where $M = (E, \leq, \lambda)$ is an MSC and $\mathcal{I} \subseteq (E \times E) \times (\mathbb{N} \times \mathbb{N})$ is a set of interval constraints.*

2.3 Timed MSCs

In a timed MSC, events are explicitly time-stamped so that the ordering on the time-stamps respects the partial order on the events.

Definition 4. *A* timed MSC *is pair (M, τ) where $M = (E, \leq, \lambda)$ is an MSC and $\tau : E \to \mathbb{R}_{\geq 0}$ assigns a nonnegative time-stamp to each event, such that for all $e_1, e_2 \in E$, if $e_1 \leq e_2$ then $\tau(e_1) \leq \tau(e_2)$.*

A timed MSC satisfies a timed MSC template if the time-stamps assigned to events respect the interval constraints specified in the template.

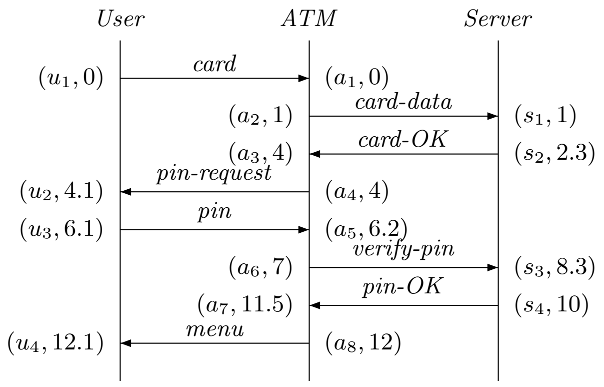


Fig. 3. A timed MSC instance describing interaction with an ATM

Definition 5. *Let $M = (E, \leq, \lambda)$ be an MSC, $\mathcal{T} = (M, \mathcal{I})$ a timed template and $M_\tau = (M, \tau)$ a timed MSC. M_τ is said to* satisfy *$\mathcal{T}$ if the following holds*

$$\text{For each } \langle (e_1, e_2), (t_1, t_2) \rangle \in \mathcal{I}, t_1 \leq \tau(e_2) - \tau(e_1) \leq t_2.$$

Definition 6. *Let $\mathcal{T}$ be a timed MSC template. We denote by $L(\mathcal{T})$ the set of timed MSCs that satisfy $\mathcal{T}$.*

Figure 3 shows a timed MSC that satisfies the template in Figure 2.

Let $M_\tau = (M, \tau)$ be a timed MSC, where $M = (E, \leq, \lambda)$, and let $\pi = e_0 e_1 \ldots e_m$ be a linearization of $(E, \leq)$. By labelling each event with its timestamp, this linearization gives rise to a timed linearization $(e_0, \tau(e_0))(e_1, \tau(e_1)) \cdots (e_n, \tau(e_n))$. As is the case with untimed MSCs, under the FIFO assumption for channels, a timed MSC can be faithfully reconstructed from any one of its timed linearizations.

3 Timed Message-Passing Automata

Message-passing automata are a natural machine model for generating MSCs. We extend the definition used in [6] to include clocks.

Definition 7. *Let $\mathcal{C}$ denote a finite-set of real-valued variables called* clocks. *A* clock constraint *is a conjunctive formula of the form $x \sim n$ or $x - y \sim n$ for $x, y \in \mathcal{C}$, $n \in \mathbb{N}$ and $\sim \in \{\leq, <, =, >, \geq\}$. Let $\Phi(\mathcal{C})$ denote the set of clock constraints over the set of clocks $\mathcal{C}$.*

Clock constraints will be used as guards and location invariants in timed message-passing automata.

Definition 8. *A* clock assignment *for a set of clocks $\mathcal{C}$ is a function $v : \mathcal{C} \to \mathbb{R}_{\geq 0}$ that assigns a nonnegative real value to each clock in $\mathcal{C}$.*

A clock assignment v is said to satisfy *a clock constraint φ if φ evaluates to true when we substitute for each clock c mentioned in φ the corresponding value $v(c)$.*

Let $v : \mathcal{C} \rightarrow \mathbb{R}_{\geq 0}$ be a clock assignment. For $d \in \mathbb{R}_{\geq 0}$, we write $v + d$ to denote the clock assignment that maps each $x \in \mathcal{C}$ to $v(x) + d$. For $X \subseteq \mathcal{C}$, we write $v[X \leftarrow 0]$ to denote the clock assignment that agrees with v for all clocks in $\mathcal{C} \setminus X$ and maps all clocks in X to 0.

Definition 9. *A* timed message-passing automaton (timed MPA) *over Σ with a set of clocks $\mathcal{C}$ is a structure $\mathcal{A} = (\{\mathcal{A}_p\}_{p\in\mathcal{P}}, \Sigma, \mathcal{C})$. Each component $\mathcal{A}_p$ is of the form $(S_p, S^p_{in}, \rightarrow_p, I_p)$, where:*

- *S_p is a finite set of p-local states.*
- *$S^p_{in} \subseteq S_p$, is a set of initial states for p.*
- *$\rightarrow_p \subseteq S_p \times \Phi(\mathcal{C}) \times \Sigma_p \times 2^{\mathcal{C}} \times S_p$ is the p-local transition relation.*
- *$I_p : S \rightarrow \Phi(\mathcal{C})$ assigns an* invariant *to each state.*

The local transition relation $\rightarrow_p$ specifies how the process p sends and receives messages.

The transition $(s, \varphi, p!q(m), X, s')$ says that in state s, p can send the message m to q and move to state s'. This transition is *guarded* by the clock constraint φ—the transition is enabled only when the current values of all the clocks satisfy φ. The set X specfies the clocks whose values are reset to 0 when this transition is taken.

Similarly, the transition $(s, \varphi, p?q(m), X, s')$ signifies that at state s, p can receive the message m from q and move to state s' provided the current clock values satisfy φ. Once again, all clocks in X are reset to 0.

A process can remain in a state s only if the current values of all the clocks satisfy the invariant $I(s)$. To make our model amenable for automated verification, we restrict location invariants to constraints that are downward closed—that is, constraints of the form $x \leq n$ or $x < n$, where x is a clock and n is a natural number.

As is customary with timed automata, we allow timed MPA to perform two types of moves: moves where the automaton does not change state and time elapses, and moves where some local component p changes state instantaneously as permitted by $\rightarrow_p$.

A global state of $\mathcal{A}$ is an element of $\prod_{p\in\mathcal{P}} S_p$. For a global state $\overline{s}$, $\overline{s}_p$ denotes the pth component of $\overline{s}$. A *configuration* is a triple $(\overline{s}, \chi, v)$ where $\overline{s}$ is a global state, $\chi : Ch \rightarrow \mathcal{M}^*$ is the *channel state* describing the message queue in each channel c and $v : \mathcal{C} \rightarrow \mathbb{R}_{\geq 0}$ is a clock assignment. An *initial configuration* of $\mathcal{A}$ is of the form $(\overline{s}_{in}, \chi_\varepsilon, v_0)$ where $\overline{s}_{in} \in \prod_{p\in\mathcal{P}} S^p_{in}$, $\chi_\varepsilon(c)$ is the empty string ε for every channel c and $v_0(x) = 0$ for every $x \in \mathcal{C}$.

The set of reachable configurations of $\mathcal{A}$, $Conf_{\mathcal{A}}$, is defined inductively, together with a transition relation $\Longrightarrow \subseteq Conf_{\mathcal{A}} \times (\Sigma \cup \mathbb{R}_{\geq 0}) \times Conf_{\mathcal{A}}$.

- Every initial configuration $(\overline{s}_{in}, \chi_\varepsilon, v_0)$ is in $Conf_{\mathcal{A}}$.
- If $(\overline{s}, \chi, v) \in Conf_{\mathcal{A}}$ and $d \in \mathbb{R}_{\geq 0}$ such that v and $v + d$ satisfy the invariants $\{I_p(\overline{s}_p)\}_{p\in\mathcal{P}}$, then there is a global move $(\overline{s}, \chi, v) \overset{d}{\Longrightarrow} (\overline{s}, \chi, v + d)$ and $(\overline{s}, \chi, v + d) \in Conf_{\mathcal{A}}$.

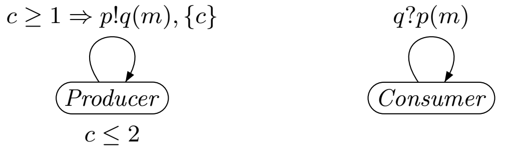


Fig. 4. A timed MPA: producer-consumer

- If $(\overline{s}, \chi, v) \in Conf_{\mathcal{A}}$ and $(\overline{s}_p, \varphi, p!q(m), X, \overline{s}'_p) \in \rightarrow_p$ such that v satsfies φ, there is a global move $(\overline{s}, \chi, v) \overset{p!q(m)}{\Longrightarrow} (\overline{s}', \chi', v[X \leftarrow 0])$ with $(\overline{s}', \chi', v[X \leftarrow 0]) \in Conf_{\mathcal{A}}$, where, for $r \neq p$, $\overline{s}_r = \overline{s}'_r$, $\chi'((p,q)) = \chi((p,q)) \cdot m$, and for $c \neq (p,q)$, $\chi'(c) = \chi(c)$.
- Similarly, if $(\overline{s}, \chi, v) \in Conf_{\mathcal{A}}$ and $(\overline{s}_p, \varphi, p?q(m), X, \overline{s}'_p) \in \rightarrow_p$ such that v satisfies φ, there is a global move $(\overline{s}, \chi, v) \overset{p?q(m)}{\Longrightarrow} (\overline{s}', \chi', v[X \leftarrow 0])$ with $(\overline{s}', \chi', v[X \leftarrow 0]) \in Conf_{\mathcal{A}}$, where, for $r \neq p$, $\overline{s}_r = \overline{s}'_r$, $\chi((q,p)) = m \cdot \chi'((q,p))$, and for $c \neq (q,p)$, $\chi'(c) = \chi(c)$.

Let $\mathrm{prf}(\sigma)$ denote the set of prefixes of a timed word $\sigma = (a_1, t_1)(a_2, t_2) \ldots (a_k, t_k) \in (\Sigma \times \mathbb{R}_{\geq 0})^*$. A run of $\mathcal{A}$ over σ is a map $\rho : \mathrm{prf}(\sigma) \rightarrow Conf_{\mathcal{A}}$ such that $\rho(\varepsilon)$ is assigned an initial configuration $(\overline{s}_{in}, \chi_\varepsilon, v_0)$ and for each $\sigma' \cdot (a_i, t_i) \in \mathrm{prf}(\sigma)$, $\rho(\sigma') \overset{d_i}{\Longrightarrow}\overset{a_i}{\Longrightarrow} \rho(\sigma' \cdot (a_i, t_i))$ with $t_i = t_{i-1} + d_i$ and t_0 implicitly set to 0.

The run ρ is *complete* if $\rho(\sigma) = (s, \chi_\varepsilon, v)$ is a configuration in which all channels are empty. When a run on σ is complete, σ is a timed linearization of a timed MSC. We define $L(\mathcal{A}) = \{\sigma \mid \mathcal{A}$ has a complete run over $\sigma\}$. Thus, $L(\mathcal{A})$ corresponds to the set of timed linearizations of a collection of timed MSCs.

Figure 4 is a simple example of a timed MPA. Here, the traditional producer-consumer system is augmented with a clock c in the producer process. The constraint $c \geq 1$ on the transition ensures that each new message is generated by the producer at least one unit of time after the previous one. The location invariant $c \leq 2$ forces the producer to generate a new message no later than two units of time after the previous one. The consumer process has no timing constraints. Figure 5 shows a typical timed MSC generated by the timed MPA in Figure 4.

4 Verifying Timed Scenarios

Given a timed MSC template $\mathcal{T}$ and a timed MPA $\mathcal{A}$, the verification question that we address is whether $\mathcal{A}$ exhibits any timed scenario that is consistent with $\mathcal{T}$. In other words, we would like to check that $L(\mathcal{T}) \cap L(\mathcal{A})$ is nonempty.

More generally, we identify a state property α that we would like the system to satisfy at the end of the template. We would then like to check that every timed MSC $M_{\mathcal{T}} \in L(\mathcal{T}) \cap L(\mathcal{A})$ satisfies α.

For instance, in the ATM example, suppose the template to be verified is the one in Figure 2. We could associate with this template the condition that the

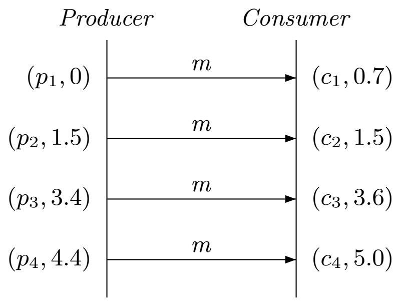


Fig. 5. A timed MSC generated by the producer-consumer system

User is in a state where it can select an item from the menu. We would then insist that every timed MSC that satisfies this template leaves the *User* in the appropriate state.

Sometimes, it is fruitful to describe forbidden scenarios as timed templates. Let $\mathcal{T}$ be such a *negative* template. We then want to check that a timed MPA $\mathcal{A}$ does *not* exhibit a timed scenario consistent with $\mathcal{T}$. In other words, we would like $L(\mathcal{T}) \cap L(\mathcal{A})$ to be empty.

Instead of looking for exact matches, we can also consider the scenario verification problem modulo embedding. The notion of one MSC being embedded in another is the usual one—there is an injective function mapping the messages in the first MSC into the messages in the second MSC that preserves the partial order between the events of the first MSC. Checking for patterns modulo embedding is useful, for instance, when a system introduces auxiliary messages to implement a protocol. For instance, most implementations of the standard telnet protocol exchange additional messages to verify operating system details, even though this is not part of the telnet protocol specification.

When checking for scenarios modulo embedding, we could constrain the nature of the embedding. A *strict* embedding is one in which all additional messages in the target MSC carry labels that are disjoint from those in the source. In other words, if we restrict the target MSC to the message alphabet of the source MSC, we obtain an exact match. In a *weak* embedding, we do not impose this restriction.

For timed MSCs, it seems important to work with weak embeddings. Consider the example in Figure 6. The template we are looking for is a simple message followed by an acknowledgment within 2 units of time, shown on the left. The implementation is designed to resend the message if the acknowledgment does not arrive within the specified time of 2 units. Thus, the implementation may generate a timed MSC such as the one on the right, where the message is sent twice. Nevertheless, it seems reasonable to argue that this system does exhibit a scenario consistent with the timed template.

In the untimed setting, the problem of matching MSCs upto embedding was considered in [10], where it was shown that a straightforward greedy algorithm works under an interpretation in which MSCs are closed with respect to *race conditions* [1], which permits the reordering along a process line of some pairs

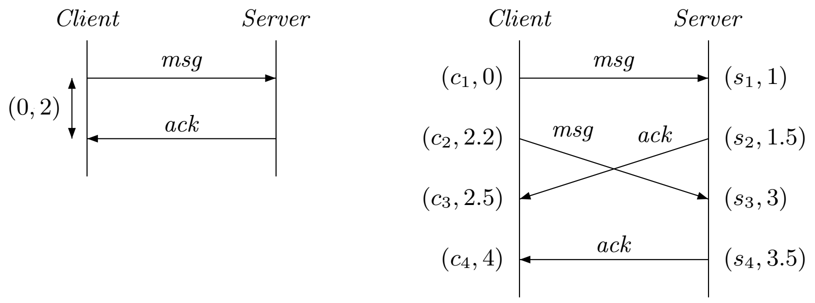


Fig. 6. Weak embedding is required for timed MSCs

of events that are not explicitly causally ordered. If one does not assume closure with respect to race conditions, backtracking seems unavoidable in solving the problem [4].

The scenario matching problem for timed MSCs is more complicated than the same problem for untimed MSCs in one obvious way. Even though a timed template is defined with respect to a single underlying MSC, the set of timed MSCs that satisfy a given template is in general infinite. Thus, even with a single template, the matching problem comes down to one of comparing infinite collections of (timed) MSCs.

5 Using Uppaal for Scenario Verification

In [4], an approach is presented for solving the embedding problem for untimed MSCs using the modelchecker Spin [7]. Message-passing automata are naturally represented in Spin. The strategy suggested by [4] is to augment the underlying system with an auxiliary *monitor process*. Each send and receive action in the system is accompanied by a synchronization action with the monitor process. In this way, the monitor process builds up a global picture of the MSC being generated by the system. When the monitor detects that the given scenario has been embedded, it enters a good state. The scenario embedding problem can then be translated into a standard LTL model-checking problem for the composite system, where the property to be checked includes the condition that the monitor enters a good state.

This approach is not suitable in our setting for two reasons. The first is that Spin has no way of dealing with time. Another complication is that timed MSC templates generate infinite families of timed MSCs. This makes the design of a monitor process more complicated.

Instead, we move over to Uppaal, a modelchecker for timed systems [2]. Uppaal supports the analysis of networks of timed automata for timing properties. Unfortunately, unlike Spin, Uppaal does not have a direct way of modelling asynchronous communication. However, we can simulate asynchronous communication by creating explicit buffer processes. Moreover, we can exploit the synchronous communication paradigm built-in to Uppaal to get around the problem of having a separate process to monitor the communication patterns of the system. Instead, we synchronize the system with the template at each communication action. This allows the system to evolve only along trajectories that are consistent with the template, thus automatically restricting the behaviours of the composite system to those that are of interest.

5.1 Modelling Channels in Uppaal

Since UPPAAL has no notion of buffered communication, we construct an explicit buffer process for each channel between processes. Message passing is simulated by a combination of shared memory and binary synchronization. Let p and q be processes and let c be the channel between p and q. We create a separate process c which maintains, internally, an array of messages M_{pq} whose size corresponds to the capacity of c. This array is used by c as a circular buffer to store the state of the channel. The process c maintains two pointers into the array: the next free slot into which p can write and the slot at the head of the queue from which q will next read a message.

The channel c shares two variables s_{pc} and r_{cq} with p and q, respectively. These are used to transfer information about the actual message between the processes and the channel. The channel c also uses two special actions a_{pc} and a_{cq} to synchronize with p and q, respectively. These synchronizations represent the actual insertions and deletions of messages into and from the channel.

When p sends a message m to q, it sets the shared variable s_{pc} to m and synchronizes with c on a_{pc}. When c synchronizes with p, it copies the message from s_{pc} into the array slot that currently corresponds to the end of the queue and then increments the free slot pointer to point to the next position in the array.

Symmetrically, when q wants to read a message m from p, it sets the shared variable r_{cq} to m and then synchronizes with c on action a_{cq}. In c, this synchronization is guarded by conditions that check that there is at least one message in the queue and that the message at the head of the queue matches the one q is looking for, as recorded in the shared variable r_{cq}.

5.2 Handling Timed MSC Templates

To verify timed MSC templates, the first step is to convert such a template into a timed MPA whose language of timed MSCs corresponds to that of the template. Since the template is built from a single MSC, the communication structure of the MPA is fixed and can be computed easily, as explained in [4]. Each time constraint in an MSC template is local to a process. We introduce a new clock for each constraint and add clock constraints using these clocks to guard the actions of the MPA so that it respects the timed template.

5.3 Computing $L(\mathcal{T}) \cap L(\mathcal{A})$

We can now augment the system description in UPPAAL so that the evolution of the system to be verified is controlled by the external template specification.

Recall that each action corresponding to sending or receiving a message by a local process is broken up into two steps in the Uppaal implementation, one which sets the value of a shared variable s_{pc} and another which communicates with the buffer process via a shared action a_{pc}. We extend this sequence to a third action, b_{pc}, by which the system synchronizes with the specification. A move of the form $s \overset{p!q(m)}{\Longrightarrow} s'$ in the original timed MPA now breaks up, in the Uppaal implementation, into a sequence of three moves $s \overset{s_{pc}=m}{\Longrightarrow} s_1 \overset{a_{pc}}{\Longrightarrow} s_2 \overset{b_{pc}}{\Longrightarrow} s'$. The third action, b_{pc} synchronizes with the corresponding process p in the timed MPA derived from the timed template that is being verified. Thus, the system can progress via this action only if it is consistent with the constraints specified by the template.

Symmetrically, for a receive action of the form $s \overset{p?q(m)}{\Longrightarrow} s'$, the Uppaal implementation would execute a sequence of the form $s \overset{r_{pc}=m}{\Longrightarrow} s_1 \overset{\bar{a}_{cp}}{\Longrightarrow} s_2 \overset{\bar{b}_{cp}}{\Longrightarrow} s'$, where we use the convention that an action a synchronizes with a matching action $\bar{a}$.

By construction, it now follows that the timed MSCs executed by the composite system are those which are consistent with both the timed template and with the underlying timed MPA being modelled in Uppaal. Thus, we have restricted the behaviour of the system to $L(\mathcal{T}) \cap L(\mathcal{A})$, for a given timed template $\mathcal{T}$ and a given timed MPA $\mathcal{A}$. From this, it is a simple matter of invoking the Uppaal modelchecker to verify whether this set of behaviours is empty and whether all behaviours in this set satify a given property. Hence, we get a direct answer to the scenario verification problems posed in the previous section.

5.4 Matching Modulo Weak Embedding

To match scenarios modulo weak embedding, the template has to be relaxed to permit the system to exchange additional messages that are not present in the scenario being verified. This is easily done by introducing a self loop at each state of the template automaton. These self loops are labelled with additional actions and have no timing constraint. For arbitrary weak embeddings, all possible actions are enabled at each self loop. For strict embeddings, only actions that are not mentioned in the template are enabled. We can further tune the nature of the embedding we are looking for by varying the choice of additional actions from one self loop to the next.

6 Expressiveness Issues

6.1 Bounded Channels

Our implementation implicitly assumes that, in any run of the system, channels are uniformly bounded by the size of the queue that is maintained by the channel process. Recent work on timed message-passing automata [9] shows that checking whether channels are universally bounded is undecidable for in general (in particular, it is undecidable with three components and two channels).

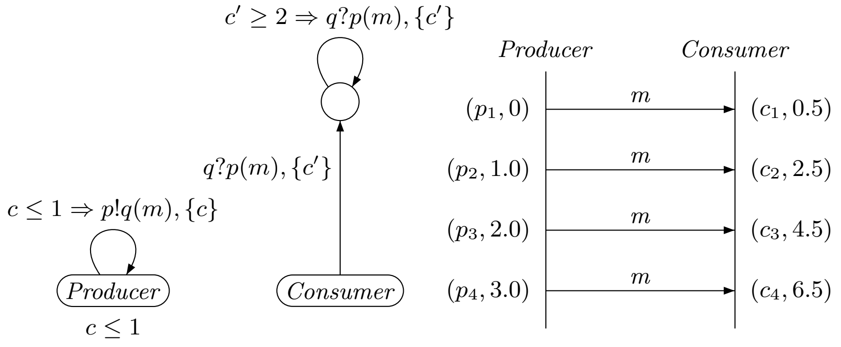


Fig. 7. The difficulty with existentially bounded channels

On the other hand, we can examine the structure of the underlying untimed system, without timing constraints. It is clear that if the underlying untimed automaton has universally bounded channels, then so will the corresponding automaton with timing constraints. Message-passing automata with universally bounded channels have a well understood theory [6], so this is a natural class of systems to which we can restrict our attention.

In the untimed case, many of the characterizations and decidability results for systems with universally bounded channels also go through for systems with existentially bounded channels [5]. A collection of MSCs is said to be existentially bounded if there is a bound B such that, for every MSC in the collection, there is at least one interleaving in which no channel's capacity exceeds B. Since, with the FIFO assumption, we can reconstruct an MSC from even a single interleaving, an existentially bounded collection of MSCs has a faithful sequential represention as a regular language, which allows us to perform operations analogous to those defined for systems with universally bounded channels.

Unfortunately, the existentially bounded linearization that one requires for an MSC may be disallowed by timing constraints. Consider again the example of a producer-consumer system, this time with local timing constraints on both send and receive actions, as shown in Figure 7. The underlying untimed system is 1-bounded and the corresponding linearization is $p_1\ c_1\ p_2\ c_2\ \cdots$, where each message is consumed as soon as it is sent. However, in the timed version shown in the figure, receive events are forced to lag behind send events, so the only timed linearizations are those in which the channel is unbounded.

6.2 Imposing Bounds on Channels

We have assumed that all clocks used in constraints are local to a process. Notice, however, that time itself is global, so all clocks proceed at a uniform rate. This is more a simplifying assumption than a technical necessity, because no comparisons are made across clocks.

One reason to introduce global clocks would be to specify bounds on channel delays. For instance, we could associate a clock with a channel that is set

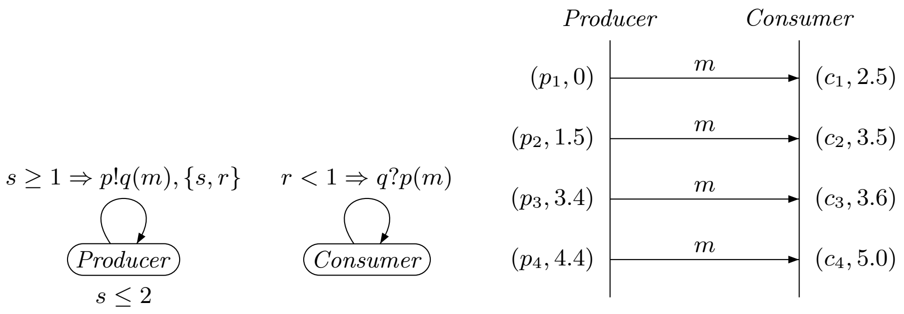


Fig. 8. Modelling channel delays

when a message is sent and use a constraint on this clock's value to impose a bound on the delay before it is received. However, a naïve implementation of this idea fails, as shown in Figure 8. Here, though the clock r seems to bound the channel delay by 1, it is possible for the first message to be received after more than 1 unit (2.5 units, in the example) because the clock r is reset by the next send event. To faithfully model channel delays, we would have to associate an array of clocks with each channel, one for each position in the queue. Even with universally bounded channels, it is not clear how to use such an array of clocks when specifying a timed MPA, because the transition associated with a send action may generate messages at different depths in the queue, depending on the history. We can unfold the MPA, keeping channel information as part of the state, but this leads to a very verbose and cumbersome specification.

7 Optimizing the Translation into Uppaal

We describe some optimizations that can be incorporated when translating the scenario matching problem into UPPAAL to enable more efficient modelchecking.

7.1 Controlling Interleavings with *Committed* Locations

Our implementation of buffered channels adds one new state to the UPPAAL system for each send or receive action in the original timed MPA. The introduction of an extra synchronize action with the corresponding process in the specification adds one new state to the system for each communication in the intersection of $L(\mathcal{T}) \cap L(\mathcal{A})$. These extra states introduce extra interleavings in the execution of the system. In the worst case, when all the actions in the system are communication events from the intersection, the total number of locations in the system gets tripled.

Consider the three moves $s \overset{s_{pc}=m}{\Longrightarrow} s_1 \overset{a_{pc}}{\Longrightarrow} s_2 \overset{b_{pc}}{\Longrightarrow} s'$. Here, the first transition is always enabled, and can be taken at the earliest possible instant. We would like the system to follow the specification, and hence the third transition should ideally be enabled when we reach the state s_2.

Uppaal allows locations be labelled *committed*. If any process is in a committed location, the next transition must involve an edge from one of the committed locations, and time cannot progress while in that location.

Marking the location s as *committed* forces the system to move to s_1 as soon as it enters location s. The system waits in state s_2 only if the corresponding process in the specification is not ready to synchronize. This would mean that the behaviour of the system is already deviating from the template. Since, we are only interested in computing $L(\mathcal{T}) \cap L(\mathcal{A})$, we mark location s_2 as *committed* as well. These committed locations remove the extra interleavings introduced into the system by splitting up a single message send/receive action into a sequence of actions and increases the efficiency of the model checker.

7.2 Guiding Uppaal Using Priorities

Uppaal also permits the allocation of priorities to channels and processes. A higher priority transitions always blocks the lower priority one. Process priorities can be used to guide the model checker, and get quicker results.

In the ATM example, the *User* process chooses from one of the many options in the menu. The *ATM* process accepts the choice made by the *User* process. In the Uppaal model, both the processes have choices regarding sending or receiving the menu options. However, in reality, the choices of the *ATM* are decided by the *User* process. We can inform Uppaal about this by declaring the *User* process to be of higher priority than the *ATM* process. Similarly, since the *Server* process needs to only service requests made by the *ATM* process, it has a lower priority. In Uppaal, these priorities are specified by the directive "`system server < atm < user ;`".

7.3 Meta Variables

The Uppaal system allows some variables to be declared as *meta variables*. Meta variables do not contribute to the state space. By declaring the global arrays within the buffers as meta variables, we can cut down on significantly on the overall set of configurations that the modelchecker has to explore.

8 Discussion

We have considered a useful extension of the scenario matching problem to timed systems. This allows us to specify and verify, more accurately, the interactions associated with typical protocol specifications.

To the best of our knowledge, ours is the first attempt to connect message-passing automata with timing constraints to timed MSCs. The problem of checking whether an MSC with interval timing constraints admits a feasible schedule has been discussed in [1], but this work studies MSCs in isolation, without reference to a system model. On the other hand, timed message-passing automata very similar those defined in this paper have been very recently considered in

[9]. In this work, the emphasis is on proving (un)decidability results for simple verification criteria such as reachability and channel boundedness. This paper does not address the semantics of such automata in terms of (timed) MSCs.

Preliminary results show that our approach to solve the problem of scenario matching in timed systems can be effectively automated using a tool like Uppaal. We are working on more detailed examples to understand better the practical issues involved with timed scenario verification. Another issue to be explored is the extent to which we can enhance the expressive power of timed MSC templates.

Acknowledgments. We thank Benedikt Bollig, Paul Gastin, K. Narayan Kumar, Akshay Sunderaraman and the anonymous referees for helpful comments and suggestions on the preliminary version of this paper.

References

1. R. Alur, G. Holzmann and D. Peled: An analyzer for message sequence charts. *Software Concepts and Tools*, **17(2)** (1996) 70–77.
2. G. Behrmann, A. Davida and K.G. Larsen: A Tutorial on Uppaal, *Proc. SFM 2004*, LNCS **3185**, Springer-Verlag (2004) 200–236.
3. J. Bengtsson and Wang Yi: Timed Automata: Semantics, Algorithms and Tools, *Lectures on Concurrency and Petri Nets 2003*, LNCS **3098**, Springer-Verlag (2003) 87–124.
4. D. de Souza and M. Mukund: Checking consistency of SDL+MSC specifications, *Proc. SPIN Workshop 2003*, LNCS **2648**, Springer-Verlag (2003) 151–165.
5. B. Genest, A. Muscholl and D. Kuske: A Kleene Theorem for a Class of Communicating Automata with Effective Algorithms. *Proc DLT 2004*, LNCS **3340**, Springer-Verlag (2004) 30–48.
6. J.G. Henriksen, M. Mukund, K. Narayan Kumar, M. Sohoni and P.S. Thiagarajan: A Theory of Regular MSC Languages. *Inf. Comp.*, **202(1)** (2005) 1–38.
7. G.J. Holzmann: The model checker SPIN, *IEEE Trans. on Software Engineering*, **23**, 5 (1997) 279–295.
8. ITU-T Recommendation Z.120: *Message Sequence Chart (MSC).* ITU, Geneva (1999).
9. P. Krcal and Wang Yi: Communicating Timed Automata: The More Synchronous, the More Difficult to Verify, *CAV 2006*, LNCS, Springer-Verlag (2006), to appear.
10. A. Muscholl, D. Peled, and Z. Su: Deciding properties for message sequence charts. *Proc. FOSSACS'98*, LNCS **1378**, Springer-Verlag (1998) 226–242.

Verification of the Generic Architecture of a Memory Circuit Using Parametric Timed Automata[⋆]

Remy Chevallier[1], Emmanuelle Encrenaz-Tiphène[2], Laurent Fribourg[2], and Weiwen Xu[2]

[1] STMicroelectronics, FTM, Central R&D, Crolles, France
[2] LSV - CNRS, ENS de Cachan, France

Abstract. Using a variant of Clariso-Cortadella's parametric method for verifying asynchronous circuits, we formally derive a set of linear constraints that ensure the correctness of some crucial timing behaviours of the architecture of SPSMALL memory. This allows us to check two different implementations of this architecture.

1 Introduction

In [10,9], Clariso and Cortadella propose a technique for verifying the timings of asynchronous circuits. The approach infers a set of sufficient linear constraints relating the delays of the internal gates of the circuit to the external delays of the circuit specification that guarantee the correct behavior of the circuit. The method is based on the reachability analysis of a timed model of the circuit (with additional abstract interpretation techniques [11]). As pointed out in [10], such parametric constraint sets are very informative for the designer, as they identify sensitive parts of the circuits (e.g., "critical paths") and interrelations between various data of the specification. Moreover, many technology mappings can be tested immediately (by mere instantiation of the parameters).

We follow here a similar approach for formally verifying some generic properties of a commercial memory designed by STMicroelectronics, called SPSMALL. Such a memory can either read or write a data (depending on the value of an input signal WEN). For the sake of brievity, we focus here on the write operation ($WEN = 0$). In this case, the memory stores the value of the input signal D into an internal memory point (located at address A), and propagates it to the output port Q. The circuit is made of a dozen of elementary components. Each component c_i is associated with an interval $[l_i^{\uparrow}, u_i^{\uparrow}]$ (resp. $[l_i^{\downarrow}, u_i^{\downarrow}]$), which gives lower and upper bounds of the component traversal delay when the input is rising (resp. falling)[1]. Such a circuit is specified by the manufacturer according to several "external" parameters (such as periods of a cyclic clock CK, time of stabilization of signal D, ...). Our timing analysis method derives a set of sufficient

[⋆] Partially supported by project MEDEA+ Blueberries.

[1] This is a straightforward generalization of "bi-bounded delay" model (see [6]), taking into account the rising or falling nature of input signal.

linear constraints relating the external parameters to the internal gate delays that guarantee the correctness of the circuit's behavior. In particular, these constraints can be seen as sufficient conditions for certain paths of the circuit to be "critical" (i.e. those along which the propagation delay is the longest).

Using the model of parametric timed automata (see [3]) and tool HYTECH [14] for reachability analysis, we are able to generate a set of linear constraints that ensures that the correctness of some crucial timing behaviors of the memory: e.g., the result of a write or read command, transmits the value of input signal D to output port Q within one clock cycle. This method is applicable to several instances of SPSMALL memories implemented with different transistor technologies (corresponding to different sets of parameter values).

Comparison with Related Work. As pointed out above, our work is adapted from Clariso-Cortadella's method [10,9]. However, [10,9] focus on a particular form of linear constraints (linear inequalities with coefficients always equal to ± 1) and represent them as a particular form of convex polyhedra, called "octahedra" [9]. In contrast, we use here linear constraints and their classical form of convex polyhedra in their full generality [13]. Other differences with [10,9] are:

- a different level of modelling: the components of the memory are represented here at the "latch" level instead of gate level. At this level of representation, the flow of input signals traverses the circuit in a linear manner (without loops), while the flow is cyclic at the gate level (the ouput of one gate can be an input of another gate and the converse can be simultaneously true),
- the use of mere forward reachability analysis rather than techniques of fixpoint computation of abstract interpretation (using, e.g., widening operators),
- the use of a decompositional approach, which splits the global system into three smaller parts,
- the use of a *step-by-step refinement* of the reachability analysis process: we start with the most general form of constraints on parameters; we then refine them progressively, via *iterative* reachability analysis, detecting, at each run, some erroneous generated states until complete elimination.

Besides [10,9], our work is along the lines of [16,5,17] where timed automata have been used extensively to model and check timing properties of asynchronous circuits (cf. [12]). Reachability analysis is there performed via tool KRONOS [18]. Our work is also a continuation of [4,7] where the SPSMALL memory is modelled as a timed automaton and some of its timing properties are proven by reachability analysis (using tool UPPAAL [15]). The crucial difference here, with respect to these previous works, is that we use the model of *parametric* timed automata (performing reachability analysis with HYTECH [14]).

Plan of the paper. In Sect. 2, we present the general objectives of our verification process. In Sect. 3, we give a general description of our method. In Sect. 4, we explain how we apply it on SPSMALL memory after having split the model into 3 parts. Final remarks are given in Sect. 5.

2 Objectives of Verification

2.1 Timings of a Memory Circuit

A memory circuit aims at storing data at some addressed locations. It is associated with two operations: 'write' and 'read'. The memory circuit has several input ports and one output port (see Fig. 1). The signals driven by input ports are:

- CK, the signal of the periodic clock;
- D, the data to be stored;
- A, the address of the internal memory location;
- WEN, the nature (write/read) of the operation.

The signal driven by the output port is Q. The memory circuit makes use of some internal devices, called 'memory points', to store data. This is depicted on Fig. 1. The write operation ($WEN = 0$ when CK is rising) writes the value of D in the internal memory point selected by A, and propagates D on output port Q. The read operation ($WEN = 1$ when CK is rising) outputs on port Q a copy of the data stored in the memory point selected by A.

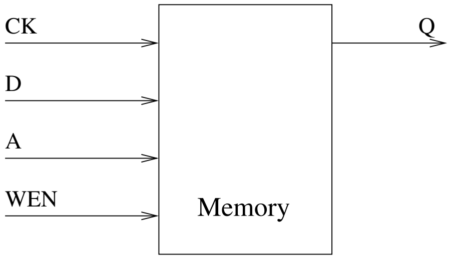


Fig. 1. Simplified interface of a memory

A memory is embedded into a synchronous environment scheduled by a periodic signal, named 'clock' (CK). The period of a cycle, t_{cycle}, decomposes itself into a high period (t_{HI}) and a low period (t_{LO}). In order to be taken into account, each input signal I has to remain stable for a given amount of time, before the rising edge of the clock. This delay is called *setup* time for I, and denoted by t_{setup_I}[2]. A write operation requires a delay, denoted by $t_{CK \to Q}^{D,WEN}$[3] due to the time of traversal of the elementary components of the memory (See Fig. 2).

The specification states the maximum delay, denoted by t_{max}, needed by a write operation.

These values of the parameters of the specification (t_{HI}, t_{LO}, t_{setup_D}, $t_{setup_{WEN}}$, t_{max}) form the "external specification" or *datasheet* of the circuit provided by the manufacturer to the customer. They are determined by *electrical*

[2] There exists also a required delay of stability, called "hold" time, required *after* the rising edge of the signal, but we will not consider it in this paper.

[3] A similar delay exists for a read operation, but we will not consider it in this paper.

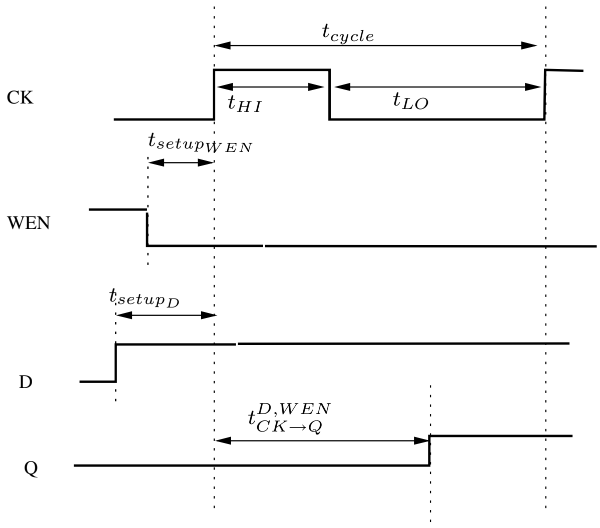


Fig. 2. A *write* operation corresponding to a rising edge of D ($D\uparrow$)

simulation. In order to perform such a simulation, each component of the memory is modelled at the transistor level as a set of differential equations, which represent the Kirshoff laws associated with the electric current traversing the component. These differential equations depend on the physical characteristics of the transistors and wires (capacitors, resistors, ...). The full memory is thus represented as a big system of differential equations. The values of the datasheet are computed by resolution of these differential equations, using, e.g., tool HSIM [1]. Actually, such a simulation process is much too long to be performed in a complete manner. Sensitive portions of the circuit, which are supposed to contain the longest paths of traversal, are therefore identified by hand. Electrical simulations are performed only for such limited portions of circuit, which are assumed to contain the critical paths. Such an assumption of 'criticality' is risky: it is very difficult to identify by hand relevant sensitive portions of the circuit (especially when the complexity of the circuit increases). The need for formal methods to *verify* the timings of the datasheet is therefore widely recognized.

2.2 Verified Property

We will focus in this paper on the following "response time" property, expressing an important aspect of the timing correctness of the memory's behavior.

The result of a *write* command is produced on output port Q within t_{max}. This will be expressed as: $t_{CK\to Q}^{D,WEN} \leq t_{max}$, where $t_{CK\to Q}^{D,WEN}$ represents the time (with respect to the beginning of clock cycle CK) after which signal Q reproduces the rising edge of D. Besides, our analysis will allow us to infer an optimal value for parameter t_{setup_D}.

Other properties regarding the writing into the internal memory point and read operation have been proven similarly, but will not be presented here, due to the lack of space.

Note that, as we focus here on the propagation of D through the whole memory (and disregard the writing in the internal memory point), the circuit parts involving the memory point and the address decoder are omitted here.

3 Method

Roughly speaking, the process consists to:

– construct a model of the memory under the form of a timed automaton,
– generate the set of reachable states (within two cycles),
– infer (by stepwise refinement) a set of timing constraints ensuring the response time property,
– verify that, for a given implementation, the response time property holds by checking the instantiated constraints.

3.1 General Scheme for Modelling a Circuit with Timed Automata

In contrast with electrical simulation, the verification process requires the modelling of the internal components of the memory, not from an electrical point of view (using differential equations), but at a *symbolic* abstract level. A circuit component is characterized by a Boolean function f (mapping its inputs to its outputs) and a propagation delay, given under the form of two intervals $[l^{\downarrow}, u^{\downarrow}]$ and $[l^{\uparrow}, u^{\uparrow}]$ (model of *bi-bounded inertial delays* distinguishing the propagation times of *rising* and *falling* edges [6]). The model of timed automata [3] is especially well-suited to represent asynchronous circuits (see, e.g., [5,17]); however, these models use a bi-bounded delay model without distinguishing the propagation delays of rising and falling edges. This unique propagation delay interval is too coarse for our circuit, thus we introduce two interval delays, $[l^{\uparrow}, u^{\uparrow}]$ for the propagation of a rising edge, and $[l^{\downarrow}, u^{\downarrow}]$ for the propagation of a falling edge.

Roughly speaking, a timed automaton is a finite state automata enriched with *(symbolic) clocks* that evolve at the same uniform rate, and can be reset to zero. A *state* is a pair (ℓ, v) where ℓ is a *location* (or “control state”), and v a clock valuation. Each location is associated with a conjunction of linear constraints over clocks, called *invariant.* A state (ℓ, v) has a *discrete transition*, labelled e, to (ℓ', v') if v satisfies a constraint, called *guard*, associated to e, and v' is obtained from v by resetting certain clocks to 0. The state (ℓ, v) has a *time transition* of duration t to (ℓ, v') if $v' = v + t$ and for all t' $(0 \leq t' \leq t)$, $v + t'$ satisfies the invariant associated to ℓ. States can be expressed under the symbolic form of conjunctions of linear constraints. Such states are classically represented as convex polyhedra (see, e.g., [14]). Sets of states correspond to union of polyhedra. The *(forward) reachability analysis* consists in generating iteratively the "successors" of sets of states via the system transitions, using elementary manipulation of polyhedra.

In order to model the circuit, each component is represented as a timed automaton combining its functionality f and delay intervals $[l^{\downarrow}, u^{\downarrow}]$ and $[l^{\uparrow}, u^{\uparrow}]$ (see [16,17] for a model with a unique interval). Input signals CK, D and WEN are themselves represented as timed automata. The global circuit is modelled as a composition of timed automata where synchronization is used to model the transmission of an internal signal between two components. A central clock s is used to measure the evolution of time. The states generated by reachability analysis correspond to linear constraints which relate the values of input and output signals, depending on timing values of the datasheet, the values of the internal delays, as time s evolves.

Let us note incidentally that our verification process still relies on results obtained by simulation, as it makes use of values of $[l^{\downarrow}, u^{\downarrow}]$ and $[l^{\uparrow}, u^{\uparrow}]$ for internal delays. However simulation is performed here at a lower scale (component), and is hopefully more reliable than for giving end-to-end values.

3.2 Modelling SPSMALL

Level of Modelling. A memory can be modelled at many different levels of complexity, e.g., in a increasing order: at the functional block level, at the "latch" level, at the gate level, or at the transistor level.

For the SPSMALL memory, the model can thus be implemented using: 3 main components, at the block level (cf [4]), 30 components at the latch level, 70 components at the gate level, or 200 components at the transistor level.

There is a tradeoff in finding the appropriate level of modelling. The lower the level of modelling is, the more faithful to the reality the model is, but the more difficult the verification process is. In particular, the task of finding by simulation the delays of each component becomes impracticable at the lowest levels (gate, transistors).

In this work, we chose to represent the memory at the latch level. The advantage is to limit the number of components (around 30) at a reasonable size, and to have a "schematics" describing the architecture of the memory at this level, which closely corresponds to (VHDL) code automatically produced with commercial tool TLL [2]. The interesting portion of this schematics for the property on which we focus here, is described in Fig. 3. It contains 12 components: 7 "wire" components, which transmits an input; one "not" component, which transmits the negation of its input ; one "or" component which computes the logical "or" of its two inputs[4]; and two "latches", $latch_D$ and $latch_{WEN}$. Two timing intervals are associated to each component, one representing the propagation delay of a rising edge (e.g. $[l_0^{\uparrow}, u_0^{\uparrow}]$ for component $wire_0$), and the other representing the propagation of a falling edge (e.g. $[l_0^{\downarrow}, u_0^{\downarrow}]$ for component $wire_0$). The precise behavior of these components is explained below.

[4] The associated delay has been actually incorporated into the input wires, and will not appear in the following.

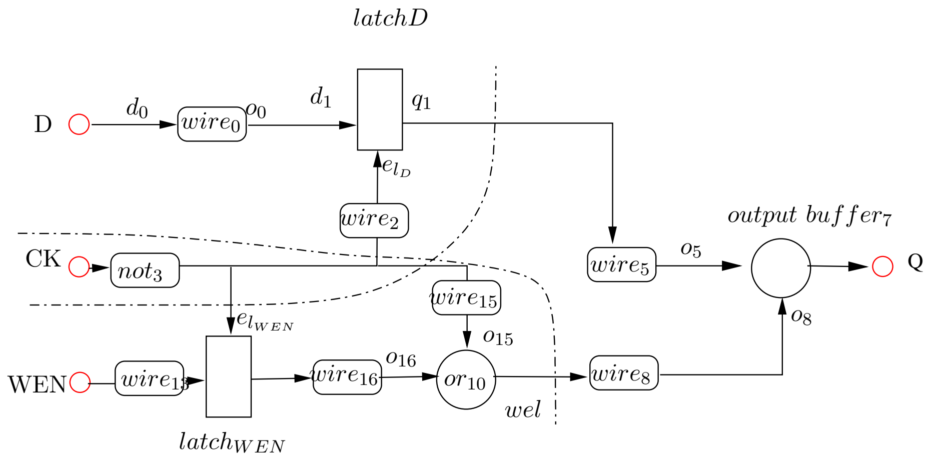


Fig. 3. Schematics'excerpt of SPSMALL

Modelling SPSMALL as a timed automaton. The SPSMALL memory is modelled as a timed automaton, resulting from the composition of all the timed automata corresponding to the input signals and the components. More precisely:

– Each input signal of the memory (synchronous signal CK, input signal D, read/write command WEN) is represented as a timed automaton.
– Each component of the memory (latch, wire, ...) is also represented as a timed automaton,
– The transmission of an internal signal between two components is modelled by synchronizing the two corresponding timed automata via a shared discrete transition.
– The emission of a signal at the output port Q of the memory corresponds to the updating of an internal variable, denoted by q, of the model.

We will focus on two main kinds of components: wires and latches (the "not","or" or "output buffer" components are similar). The simplest kind of component is a "wire" component, which transmits an input signal after a certain delay: the wire component is enabled $l^{\uparrow}$ (pico)secs after the rising edge of the input signal, and the output signal is fired after a delay lying in $[l^{\uparrow}, u^{\uparrow}]$ (parameters $l^{\uparrow}$ and $u^{\uparrow}$ are used to propagate a falling edge). We adopt the "inertial delay" interpretation (see [6]): changes that do not persist for l time are filtered out. Such a wire component is naturally modelled as the timed automaton depicted in Fig. 4. It makes use of 1 internal clock c and 5 locations. The symbol $d \uparrow$ (resp. $d \downarrow$) corresponds to a rising edge (resp. falling edge) of input internal signal d, and similarly for symbol $o \uparrow$ (resp. $o \downarrow$) with output signal o. Each edge corresponds to a discrete transition labelled with the name of input signals ($d \uparrow$ or $d \downarrow$) or output signals ($o \uparrow$ or $o \downarrow$). The medium-level locations are associated

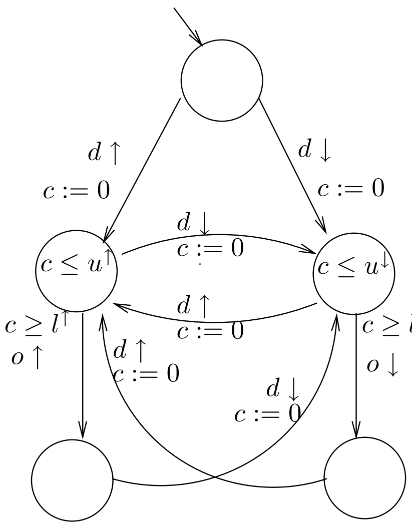


Fig. 4. Timed automaton of a wire component with delays $[l^{\uparrow}, u^{\uparrow}]$ and $[l^{\downarrow}, u^{\downarrow}]$ to propagate an edge from d to o

with invariants $c \leq u^{\uparrow}$ or $c \leq u^{\downarrow}$. The guard associated with edges outgoing downwards from these locations is $c \geq l^{\uparrow}$ or $c \geq l^{\downarrow}$.

A latch is a component that can store one bit of data. It has two inputs d (*data to be latched*) and e (*enable*), and one ouput q. It propagates the input data d to its output o (with a delay in $[l^{\uparrow}, u^{\uparrow}]$ or $[l^{\downarrow}, u^{\downarrow}]$) as long as e is high. While e is low, q keeps its value (even if d changes). A latch corresponds to the timed automaton depicted on Fig. 5 (using the same conventions as for the wire automaton). There are 6 locations and 1 clock c. The name of the locations $e_i d_j$ reflects the values i and j of e and d respectively; besides $e_1 d_j B$ indicates that an output $q\uparrow$ (resp. $q\downarrow$) has already been output if $j = 1$ (resp. $j = 0$).

3.3 Reachability Analysis

In order to verify property: $t^{D,WEN}_{CK\to Q} \leq t_{max}$, we model the behavior of the memory along two cycles:

– a 1st cycle where the values of D and WEN are set t_{setup_D} and $t_{setup_{WEN}}$ time before the 2nd rising edge of CK (corresponding to the write operation); input signal WEN is modelled as a falling edge, followed by a low level (selection of a write command), and input signal D is modelled as a rising edge, followed by a high level (in keeping with the stabilization requirement $setup_D$).
– a 2nd cycle where the write operation is performed (the D value is propagated on Q port).

Accordingly, the observation of the generated states is done along two cycles (see Fig. 6).

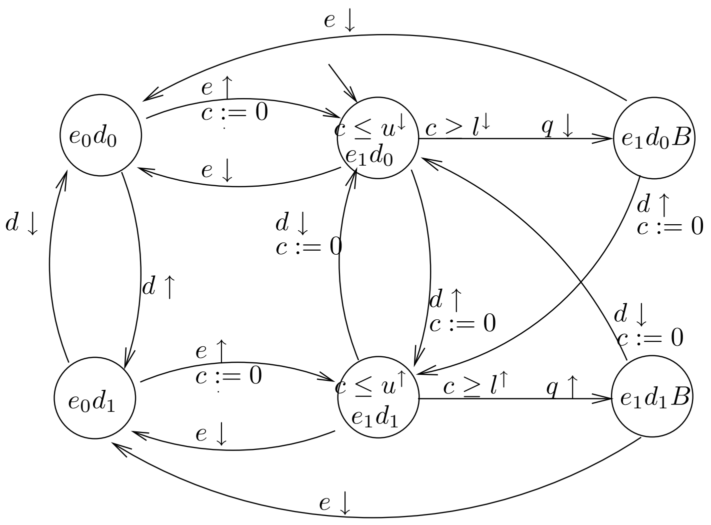


Fig. 5. Timed automaton of a latch component with delays $[l^{\uparrow}, u^{\uparrow}]$ and $[l^{\downarrow}, u^{\downarrow}]$ to propagate an edge from d to q

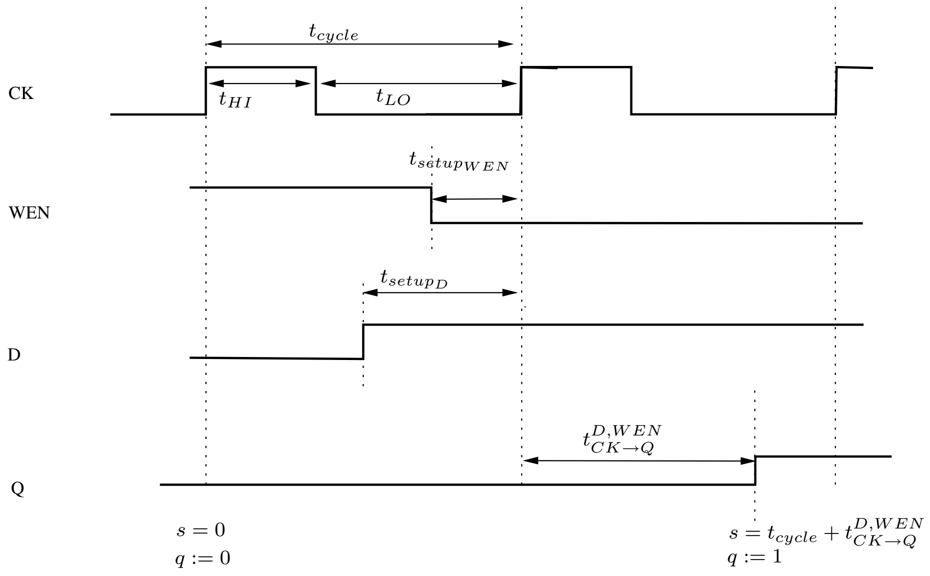


Fig. 6. The *write* operation used in our experiment

In the next subsection, we will explain how the verification method applies without instantiation of the parameters. Let us first explain how the method works given a specific implementation of the memory: in the rest of this section, we assume all the parameters to be instantiated with the values of the datasheet and those given by simulation.

A central clock s (initialized to 0, and never reset) is used to measure the evolution of time, during 2 cycles. We also use a flag q (initialized to 0), which stores the fact that the rising edge of input signal D has reached Q port. The value of s when flag q is set to 1, corresponds to the sought value $t_{CK\to Q}^{D,WEN}$. Starting from $s = 0 \wedge q = 0$, the set of reachable states is iteratively computed until, either:

- the switch of flag q occurs before 2 cycles, or
- 2 cycles are run out without any switch of q.

This yields a set of final states, denoted by $Post^{2t_{cycle}}$, which can be decomposed into:

- *good* states, i.e, states, for which the switch of q has occurred ($q = 1 \wedge s = t_{cycle} + t_{CK\to Q}^{D,WEN} \leq 2t_{cycle}$).
- *bad* states, i.e., states, for which 2 cycles has run out without any switch of q ($q = 0 \wedge s = 2t_{cycle}$).

The property holds iff:

- all the final states of $Post^{2t_{cycle}}$ are good (no bad state), and
- for each final state, the value $t_{CK\to Q}^{D,WEN}$ of s is at most equal to t_{cycle} (i.e.: $t_{CK\to Q}^{D,WEN} \leq t_{cycle}$).

3.4 Timing Constraints Extraction by Stepwise Refinement

Let us now explain how to perform the reachability analysis at a *generic* level, without setting the parameters to some specific values of a given implementation. We denote by $Assumption$, the set of symbolic constraints, relating the parameters t_{HI}, t_{LO}, t_{setup_D} and $\{[l_i, u_i]\}$ together. At the beginning of the process, $Assumption$ is $True$, which means that we start with the most general parameters (l_i is just assumed to be less than or equal to u_i). Given an initial set of states characterized by $Assumption$, we perform reachability analysis on 2 cycles. We thus generate a set of constraints, denoted by $Post^{2t_{cycle}}(Assumption)$. The first run of $Post^{2t_{cycle}}(Assumption)$, with $Assumption = True$, usually contains bad (final) states. The *refinement* process consists to eliminate such bad states, by restricting the possible values of the parameters, as follows:

1. select a bad state of $Post^{2t_{cycle}}(Assumption)$;
2. detect a "suspect" constraint;[5]
3. add the negation of this subconstraint to $Assumption$.
4. Recompute $Post^{2t_{cycle}}(Assumption)$ after having reset s and q to 0.

And so on, perform iteratively 1-2-3-4 until no bad state is generated.

At each run, $Assumption$ is a conjunction of linear constraints with an increasing number of conjuncts. Accordingly, $Post^{2t_{cycle}}(Assumption)$ decreases at

[5] In our context, a *suspect* constraint is a constraint violated by the values of the parameters of an SPSMALL available instance, *viz.* $SP1$ (see Sect. 4.3).

each run. At the end, the refinement process outcomes two formulas, $Assumption$ and $Final \equiv Post^{2t_{cycle}}(Assumption)$:

- $Assumption$ is a conjunctive constraint binding parameters t_{HI}, t_{LO}, t_{setup_D}, $t_{setup_{WEN}}$, $\{[l_i, u_i]\}$.
- $Final$ is a set of conjunctive constraints relating $t^{D,WEN}_{CK\rightarrow Q}$ to t_{HI}, t_{LO}, t_{setup_D}, $t_{setup_{WEN}}$, $\{[l_i, u_i]\}$.

Moreover, by construction, we have:

- $Final$ is the set of states reachable from $Assumption$ for which signal D has reached Q before 2 cycles (i.e., such that $q = 1 \wedge s = t_{cycle} + t^{D,WEN}_{CK\rightarrow Q} \leq 2t_{cycle}$).
- the propagation of D on Q *always* occurs *before* 2 cycles. ($Final$ contains *all* the states reachable from $Assumption$ such that $q = 1$).[6]

This set of constraints can be used for different purposes:

1. The validation of a specific implementation of SPSMALL. It consists in checking that:
   (a) the specific values of the datasheet $(t_{HI}, t_{LO}, t_{setup_D},\ t_{setup_{WEN}})$ and those of the internal delays $\{[l_i, u_i]\}$ are *compatible*, i.e: they satisfy $Assumption$ all together;
   (b) ensure that $t^{D,WEN}_{CK\rightarrow Q} \leq t_{max}$.

2. The optimisation of some parameters of the memory: it can be an external parameter (such as t_{setup} of an input signal) or an internal timing (such as $[l_i, u_i]$ of an internal component). For instance, we can find the minimal value of t_{setup_D} such that all constraints remain satisfiable.

4 Verification of SPSMALL

We now apply the above method for deriving the set $Assumption \cup Final$ of constraints associated with memory SPSMALL, and checking the correctness of two of its instances $SP1$, a “high-speed” implementation, and $SP2$, a “low-power” implementation.

4.1 Decomposition

In practice, we cannot apply directly the method described above, because we cannot perform reachability analysis with HYTECH due to the high number (34) of parameters. The model is therefore decomposed into three parts (see the dashed lines on Fig. 3):

- The 1st part represents the D's edge propagation through $latch_D$: The input signals are D and CK. The output is the output of $latch_D$, denoted by q_1. The goal is to compute the constraints on lower and upper bounds on $t^{D}_{CK\rightarrow q_1}$.

[6] This comes from the fact that all the bad states have been eliminated.

– The 2nd part represents the propagation of the WEN edge through $latch_{WEN}$ (among other components). The input signals are WEN and CK. The output is denoted by wel. The goal is to compute the constraints on the lower and upper bounds of $t^{WEN}_{CK\to wel}$.
– The 3rd part represents the propagation of q_1's edge through Q. The input signals are q_1 and wel. The output is Q. The goal is to compute the constraints on the lower and upper bounds of $t^{q_1,wel}_{CK\to Q}$. Using the bounds $t^{WEN}_{CK\to wel}$ and $t^{D}_{CK\to q_1}$ found for q_1 and wel respectively in the two first parts, this will allow us finally to determine the constraints on $t^{D,WEN}_{CK\to Q}$.

4.2 Generic Constraints

We analyze separately each part, thus obtaining constraints binding intermediate input and output parameters (see [8]). The "suspect" constraints discarded during the refinement process (see Sect. 3.4) are those incompatible with the values of the delay intervals $[l^{\uparrow}, u^{\uparrow}]$ and $[l^{\downarrow}, u^{\downarrow}]$ of instance $SP1$ (see Sect. 4.3). By recombination of these separate sets of constraints, we obtain constraints relating the inputs and outputs of the whole memory, that are given below. For conciseness, we consider only the case of a rising edge of D ($D\uparrow$):

$Assumption$:

$$t_{setup_D}+u^{\downarrow}_2+u^{\downarrow}_3 < l^{\uparrow}_0+t_{LO} \wedge\ t_{HI}+t_{LO} < l^{\downarrow}_2+l^{\downarrow}_3+t_{setup_D} \wedge\ u^{\downarrow}_2+u^{\downarrow}_3+u^{\uparrow}_1 < t_{LO}$$
$$\wedge\ u^{\downarrow}_3 + t_{setup_{WEN}} < t_{LO} + u^{\downarrow}_{13} \wedge\ u^{\downarrow}_{13} + u^{\downarrow}_{14} < t_{setup_{WEN}} + l^{\downarrow}_3 \wedge\ u^{\downarrow}_{14} < t_{HI}$$
$$\wedge\ u^{\downarrow}_{13} + u^{\downarrow}_{14} + u^{\downarrow}_{16} < t_{setup_{WEN}} + l^{\downarrow}_3 + u^{\downarrow}_{15} \wedge\ t_{setup_D} + u^{\downarrow}_3 + u^{\downarrow}_{15} \le l^{\uparrow}_5 + l^{\uparrow}_0 + l^{\uparrow}_1$$
$$\wedge\ u^{\uparrow}_5 + u^{\uparrow}_0 + u^{\uparrow}_1 \le l^{\downarrow}_8 + l^{\downarrow}_3 + l^{\downarrow}_{15} + t_{setup_D}$$

$Final$:

$$l^{\downarrow}_3 + l^{\downarrow}_{15} + l^{\downarrow}_8 + l^{\uparrow}_7 \le t^{D,WEN}_{CK\to Q} \le u^{\downarrow}_3 + u^{\downarrow}_{15} + u^{\downarrow}_8 + u^{\uparrow}_7$$

The constraints are symmetrical in the case of a falling edge of D ($D\downarrow$): more precisely, each $l^{\uparrow}_i$ (resp. $u^{\uparrow}_i$) should be changed into $l^{\downarrow}_i$ (resp. $u^{\downarrow}_i$).

From $Final$ (and its symmetrical counterpart for $D^{\downarrow}$), we infer the following constraint, guaranteeing property $t^{D,WEN}_{CK\to Q} \le t_{max}$:

$$\boxed{u^{\downarrow}_3 + u^{\downarrow}_{15} + u^{\downarrow}_8 + \max\{u^{\uparrow}_7, u^{\downarrow}_7\} \le t_{max}.} \qquad (*)$$

Constraints of $Assumption$ (and its symmetrical counterpart for $D^{\downarrow}$) can be used to determine lower and upper bounds for t_{setup_D}:

$$\max\{u^{\uparrow}_0+u^{\uparrow}_1+u^{\uparrow}_5-l^{\downarrow}_8-l^{\downarrow}_3-l^{\downarrow}_{15}, u^{\downarrow}_0+u^{\downarrow}_1+u^{\downarrow}_5-l^{\downarrow}_8-l^{\downarrow}_3-l^{\downarrow}_{15}, t_{HI}+t_{LO}-l^{\downarrow}_2-l^{\downarrow}_3\} \le t_{setup_D}$$
$$\wedge\ t_{setup_D} \le \min\{l^{\uparrow}_0+l^{\uparrow}_1+l^{\uparrow}_5-u^{\downarrow}_3-u^{\downarrow}_{15}, l^{\downarrow}_0+l^{\downarrow}_1+l^{\downarrow}_5-u^{\downarrow}_3-u^{\downarrow}_{15}, t_{LO}+l^{\uparrow}_0-u^{\downarrow}_2-u^{\downarrow}_3\}$$

We thus infer an *optimal* value of t_{setup_D}, denoted by $t^{opt}_{setup_D}$, which corresponds to its lower bound[7]:

$$\boxed{t^{opt}_{setup_D} = \max\{u^{\uparrow}_0 + u^{\uparrow}_1 + u^{\uparrow}_5 - l^{\downarrow}_8 - l^{\downarrow}_3 - l^{\downarrow}_{15}, u^{\downarrow}_0 + u^{\downarrow}_1 + u^{\downarrow}_5 - l^{\downarrow}_8 - l^{\downarrow}_3 - l^{\downarrow}_{15}, t_{HI} + t_{LO} - l^{\downarrow}_2 - l^{\downarrow}_3\}}$$

A similar expression can be obtained for the optimal value of $t_{setup_{WEN}}$.

4.3 Application to Instance *SP*1

The above sets of constraints now allow us to give a formal justification of the correctness of the instance $SP1$. The values of the datasheet are (in tens of picoseconds):

$t_{HI} = 36, t_{LO} = 74, t_{setup_D} = 108, t_{setup_{WEN}} = 48, t_{max} = 56.$

The internal delays are (in tens of picoseconds):

	$[l^{\uparrow}_i, u^{\uparrow}_i]$	$[l^{\downarrow}_i, u^{\downarrow}_i]$
(l_0, u_0)	$(95, 95)$	$(66, 66)$
(l_1, u_1)	$(14, 14)$	$(18, 18)$
(l_2, u_2)	$(23, 30)$	$(23, 30)$
(l_3, u_3)	$(5, 5)$	$(2, 2)$
(l_5, u_5)	$(22, 22)$	$(45, 45)$
(l_7, u_7)	$(21, 21)$	$(20, 20)$
(l_8, u_8)	$(0, 0)$	$(22, 22)$
(l_{13}, u_{13})	$(11, 11)$	$(8, 8)$
(l_{14}, u_{14})	$(21, 22)$	$(21, 22)$
(l_{15}, u_{15})	$(14, 14)$	$(11, 11)$
(l_{16}, u_{16})	$(24, 24)$	$(0, 0)$

We check that all the constraints of *Assumption* (and those of the symmetrical counterpart for $D\downarrow$) are satisfied. We also check constraint $(*)$. This shows that $SP1$ satisfies $t^{D,WEN}_{CK\rightarrow Q} \leq t_{max}$. Furthermore, we find the value 96 for $t^{opt}_{setup_D}$, which matches with the optimal value found by simulation by the designer.

4.4 Application to Instance *SP*2

As mentioned above, the values of the datasheet and internal delays of $SP1$ have been used in the refinement process in order to derive the appropriate set of generic constraints. Therefore, the derived set of constraints $Assumption \cup Final$ of Sect. 4.2 has not been produced independently from $SP1$, and the correctness of $SP1$ has been checked *a posteriori*. The constraints are however available now once for all, and can be reused to check immediately any other instance of SPSMALL. This is what has been done with instance $SP2$. The values of the datasheet are (in tens of picoseconds):

$t_{HI} = 72, t_{LO} = 170, t_{setup_D} = 241, t_{setup_{WEN}} = 109, t_{max} = 142.$

The internal delays are (in tens of picoseconds):

[7] Actually, we checked that $t^{opt}_{setup_D}$ also satisfies constraints, coming from other parts of the circuit, which are not limitative in the case of instances $SP1$ and $SP2$.

	$[l_i^{\uparrow}, u_i^{\uparrow}]$	$[l_i^{\downarrow}, u_i^{\downarrow}]$
(l_0, u_0)	$(197, 197)$	$(140, 140)$
(l_1, u_1)	$(60, 60)$	$(58, 58)$
(l_2, u_2)	$(66, 66)$	$(43, 43)$
(l_3, u_3)	$(8, 8)$	$(4, 4)$
(l_5, u_5)	$(61, 61)$	$(63, 63)$
(l_7, u_7)	$(47, 47)$	$(52, 52)$
(l_8, u_8)	$(0, 0)$	$(42, 42)$
(l_{13}, u_{13})	$(23, 23)$	$(23, 23)$
(l_{14}, u_{14})	$(35, 35)$	$(36, 36)$
(l_{15}, u_{15})	$(56, 56)$	$(43, 43)$
(l_{16}, u_{16})	$(24, 24)$	$(0, 0)$

As in the case of $SP1$, we check that $SP2$ satisfies $t_{CK\rightarrow Q}^{D,WEN} \leq t_{max}$. Besides, we find the value 229 for $t_{setup_D}^{opt}$, which matches with the optimal value found by electrical simulation by the designer.

5 Final Remarks

We have shown in this paper how to apply parametrized methods to verify timed properties of the generic architecture of a memory. We have thus found certain sufficient conditions (under the form of linear inequalities between parameters) that ensure that the response time lies between certain bounds, and check this property on an instance $SP1$ of the memory. These linear inequalities have been also used to derive the optimal values of setup timings of input signals (*viz.*, setup timing for D). This analysis can be immediately applied to the verification of other instances of the SPSMALL memory, as examplified here on instance $SP2$.

Our method requires from the user a certain knowledge of the circuit, especially in the stepwise refinement process when taking the refutation of a suspect constraint. By negating such constraints, we focus on a certain class of the circuit, disregarding other possible circuit implementations. In the future, we plan to improve this phase of constraint selection, in order to make the method more complete and more automatic.

References

1. HSIM Simulator Description. In `http://www.synopsys.com/products/`.
2. TLL Transistor Abstraction Tool description. In `http://www.transeda.com/products/`.
3. R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science 126*, pages 183–235, 1994.
4. M. Baclet and R. Chevallier. Timed verification of the SPSMALL memory. In *1th International Conference Memory Technology and Development*, pages 1–2, Giens France, 2005.
5. M. Bozga, H. Jianmin, O. Maler, and S. Yovine. Verification of asynchronous circuits using timed automata. In *TPTS'02, ENTCS vol 65*, 2002.

6. J.A. Brzozowski and C-J.H. Seger. *Asynchronous Circuits.* Springer, 1994.
7. R. Chevallier, E. Encrenaz-Tiphène, L. Fribourg, and W. Xu. Timing analysis of an embedded memory: SPSMALL. In *10th WSEAS International Conference on Circuits, Athens, Greece*, 2006.
8. R. Chevallier, E. Encrenaz-Tiphène, L. Fribourg, and W. Xu. Verification of the Generic Architecture of a Memory Circuit Using Parametric Timed Automata. Technical Report LSV-06-??, Laboratory Specification and Verification, 2006.
9. R. Clariso and J. Cortadella. The octahedron abstract domain. In *Proc. 11th Static Analysis Symposium (SAS), LNCS 3148, Springer, pp. 312-327*, 2004.
10. R. Clariso and J. Cortadella. Verification of timed circuits with symbolic delays. In *Proc. Asia and South Pacific Design Automation Conference (ASP-DAC), pp. 628-633*, 2004.
11. P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL, pp. 238-252*, 1977.
12. D. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Automatic Verification Methods for Finite State Systems, LNCS 407, Springer*, 1989.
13. N. Halbwachs. Delay analysis in synchronous programs. In *CAV'93, LNCS 697, Springer, pp. 333-346*, 1993.
14. T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. A User Guide to HYTECH. In *TACAS'95, LNCS 1019, Springer, pp.41-71*, 1995.
15. K. Larsen, P. Pettersson, and W. Yi. UPPAAL in a Nutshell. *International Journal on Software Tools for Technology Transfer*, 1:134–152, 1997.
16. O. Maler and A. Pnueli. Timing analysis of asynchronous circuits using timed automata. In *CHARME'95, LNCS 987, Springer, pp.189-205*, 1995.
17. R. Ben Salah, M. Bozga, and O. Maler. On timing analysis of combinational circuits. In *FORMATS'03, LNCS 2791, Springer, pp.204-219*, 2003.
18. S.Yovine. KRONOS: A verification tool for real-time systems. *International Journal on Software Tools for Technology Transfer*, 1:123–133, 1997.

Model Checking Timed Automata with Priorities Using DBM Subtraction

Alexandre David[1], John Håkansson[2], Kim G. Larsen[1], and Paul Pettersson[2]

[1] Department of Computer Science, Aalborg University, Denmark
{adavid, kgl}@cs.aau.dk
[2] Department of Information Technology, Uppsala University, Sweden
{johnh, paupet}@it.uu.se

Abstract. In this paper we describe an extension of timed automata with priorities, and efficient algorithms to compute subtraction on DBMs (difference bounded matrices), needed in symbolic model-checking of timed automata with priorities. The subtraction is one of the few operations on DBMs that result in a non-convex set needing *sets* of DBMs for representation. Our subtraction algorithms are efficient in the sense that the number of generated DBMs is significantly reduced compared to a naive algorithm. The overhead in time is compensated by the gain from reducing the number of resulting DBMs since this number affects the performance of symbolic model-checking. The uses of the DBM subtraction operation extend beyond timed automata with priorities. It is also useful for allowing guards on transitions with urgent actions, deadlock checking, and timed games.

1 Introduction

Since the introduction of timed automata [2] in 1990, the theory has proven its capability of specifying and analysing timed systems in many case studies, e.g., [4,23]. To support such studies, tools as Kronos [7], Uppaal [18], and RED [24] have been developed to offer means for modelling, simulation, model-checking, and also testing, of real-time systems specified as timed automata.

In the implementation of real-time systems, the concept of *priorities* is often used as a way to structure and control the usage of shared resources. Priorities are often associated with processes (or tasks) to control their usage of shared resources such as CPU or shared memory areas. As a consequence, programming languages such as Ada [3,12], and scheduling policies used in real-time operating system, such as *rate-monotonic scheduling* [9], are often based on a notion of priorities on tasks. In lower levels, closer to the hardware, priorities are often associated with interrupts to hardware devices and access to e.g., shared communication buses.

Priorities have been studied in process algebras, e.g., [11,8], and can be modelled using timed automata [12,14]. However, it can be cumbersome and error-prone to do so. Consider the simple example shown in Figure 1 and assume that the location l can be reached with any time assignment satisfying the constraint

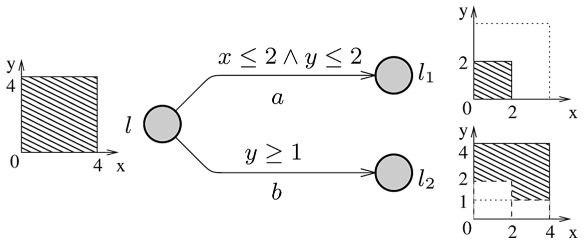


Fig. 1. A timed automaton with priorities on actions

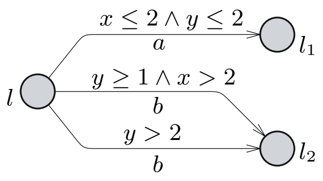


Fig. 2. Encoding of the priorities in Fig. 1

$x \leq 4 \wedge y \leq 4$. Further assume that the edge labelled with a has priority over the edge labelled b. We see that l_1 can be reached with any time assignment satisfying the constraint $x \leq 2 \wedge y \leq 2$. The location l_2 is reachable under the constraint $(y \geq 1 \wedge x \leq 4 \wedge y \leq 4) \wedge \neg(x \leq 2 \wedge y \leq 2)$, which is a non-convex set of clock valuations and thus not representable as a conjunction of simple constraints. This fact will make (symbolic) state-space exploration potentially costly, since the set of clock valuations reachable in one step over a low priority transition, such as transitions derived from the b-edge, generally will have to be represented by *a set* of convex constraint systems. In Figure 2 we show a timed automaton in which the priorities of the automaton in Figure 1 have been encoded. Note that the b-edge has been split to two edges to encode the disjunctive constraints on the clock valuations reaching l_2.

Model-checking tools for timed automata typically uses DBMs (difference bound matrices) [13,22] to represent convex constraints on clock variables. However, as illustrated above, analysis of timed automata with priorities will require the model-checking engine to efficiently handle disjunctive constraints. As a second contribution of this paper, we present a variety of techniques for performing *subtractions* on DBMs. That is, how to compute $D - D'$ defined as $D \wedge \neg D'$, for two DBMs D and D'. Guided by the goal to minimise the set of DBMs resulting from subtraction, and to keep them disjoint, we give a heuristic algorithm with good performance. To back up this statement, we present experimental evidence from applying a version of the UPPAAL tool extended with priorities, on a set of examples. We note that DBM subtraction is already needed for backward model-checking of full TCTL or scheduler synthesis [23], controller synthesis [10], and to support urgent guards.

The rest of this paper is organised as follow: Timed automata with priorities are described in Section 3, and the required DBM subtraction operation in

Section 4. In Section 5 we present subtraction algorithms that reduce the set of resulting DBMs. We show with experiments in Section 6 that our algorithm improves DBM subtractions significantly.

Related work: Priorities in process algebras are described in [11], where priorities on actions are defined in two levels. A process algebra of communicating shared processes is described in [8], where priorities are described as real numbers on events and timed actions. Preservation of congruence is a major concern in these papers.

In [6] priorities are introduced for *live* systems (time-lock and deadlock free with no indefinite waiting), with the purpose to preserve liveness in the composition of such systems. In our work we have focused on introducing priorities for existing timed automata models, and developing efficient algorithms for DBM subtraction.

In [16] a notion of priorities for timed automata based on total orderings and an algorithm for computing DBM subtractions have been proposed. In this paper, we introduce a more general notion of priorities for timed automata where the priority ordering is allowed to be partial, and allowing the priority ordering to be defined both on the level of synchronisation actions as well as the individual automata. We believe that both these two suggestions will be useful when modelling real-time systems with priorities. The subtraction of [16] is claimed to be optimal, i.e., it generates the fewest possible number of DBMs as a result. However, the ordering of constraining operations needed for subtraction is not mentioned. We argue in this paper that ordering is important and optimality of subtraction w.r.t. reachability is more difficult than just having the minimal number of DBMs from subtractions.

2 Preliminaries

2.1 Clock Constraints

Model-checking of timed automata involves exploring a state-space of symbolic states, where each symbolic state represents a set of clock valuations. For a set C of n clocks, a clock valuation is a map $v : C \mapsto \mathbb{R}_{\geq 0}$. We denote by $\mathcal{B}(C)$ the set of conjunctions of atomic constraints in the form $x_i \sim m$ or $x_i - x_j \sim m$, where m is a natural number, x_i and x_j are clock valuations of clocks i and j, and $\sim \in \{<, \leq, =, \geq, >\}$. Although it is possible to represent sets of clock valuations as regions [2], using zones is much more efficient in practice [5]. A *zone* corresponds to the set of clock valuations that satisfies a conjunction of constraints in $\mathcal{B}(C)$. A zone is convex by definition, and we represent it as a difference bound matrix (DBM).

2.2 Difference Bound Matrices

A DBM is a conjunction $D = \bigwedge_{1 \leq i,j \leq n}(x_i - x_j \sim b_{ij})$ for $\sim \in \{<, \leq\}$, written as $D = \bigwedge d_{ij}$. We use d_{ij} (or e_{ij}) to denote the constraints of a DBM D (or E).

The bound of a constraint d_{ij} is denoted $|d_{ij}|$. We define a *complement* operation over $\sim$ so that $\bar{\leq} =<$ and $\bar{<} =\leq$.

A DBM is *canonical* if it is closed under entailment, e.g. by Floyd's shortest path algorithm [15]. We consider all DBMs to be canonical.

Definition 1 (Operations on constraints). *For constraints d_{ij} and e_{ij}:*

- $d_{ij} \leq e_{ij} \Leftrightarrow d_{ij} \Rightarrow e_{ij}$
- $d_{ij} < e_{ij} \Leftrightarrow d_{ij} \neq e_{ij} \wedge d_{ij} \leq e_{ij}$
- $\neg d_{ij} = \neg(x_i - x_j \sim b_{ij}) = x_j - x_i \bar{\sim} - b_{ij}$. *Note that $\neg d_{ij}$ is a new d'_{ji} comparable with other constraints e_{ji}.*
- $d_{ik} + d_{kj} = (x_i - x_k \sim b_{ik}) + (x_k - x_j \sim' b_{kj}) = x_i - x_j \sim'' b_{ik} + b_{kj}$ *where $\sim''=<$ if $\sim=<$ or $\sim'=<$, otherwise $\sim''=\leq$.*
- $d_{ik} - d_{jk} = d_{ik} + \neg d_{jk}$

We write $v \models g$ to denote that a constraint $g \in \mathcal{B}(C)$ is satisfied by a clock valuation v. The notation $v \oplus d$ represents a valuation where all clocks have advanced by the real valued delay d from their value in v. For a set of clocks r we denote by $[r \mapsto 0]v$ the valuation that maps clocks in r to zero, and agrees with v for all other clocks. We write $D = \{v \mid v \models x_i - x_j \sim b_{ij}\}$ for the set of clock valuations that satisfy the constraints of D.

Definition 2 (Operations on zones). *For a zone D and a clock constraint g:*

- conjunction*:* $D \wedge g = \{v \,|\, v \in D, v \models g\}$,
- delay*:* $D^{\uparrow} = \{v \oplus t \,|\, v \in D, t \in \mathbb{R}_{\geq 0}\}$,
- reset*:* $r(D) = \{[r \mapsto 0]v \,|\, v \in D\}$,
- free*:* $free(D, r) = \{[r \mapsto t]v \,|\, v \in D, t \in \mathbb{R}_{\geq 0}\}$, *and*
- negation*:* $\neg D = \{v \,|\, v \notin D\}$.

Subtraction. Given two DBMs D and E, we want to subtract E from D. The resulting set S is defined as the set satisfying the constraints of D and $\neg E$. The set is not necessarily a zone. The result $S = D \wedge \neg E$, denoted $D - E$, is written as:

$$S = D \wedge \neg(\bigwedge_{1 \leq i,j \leq n} e_{ij}) = \bigvee_{1 \leq i,j \leq n} (D \wedge \neg e_{ij}), \quad \text{(De Morgan law)},$$

which is a union of D constrained by each of the negated constraints of E. This gives us a straight-forward *basic algorithm* to compute subtractions. Figure 3 illustrates this basic algorithm with two clocks. The result $S = D - E$ is represented by the union of the six smaller zones on the right in the figure. The number of zones in S is bounded by n^2, and creating each of these zones is $O(n^2)$, so the complexity of the operation is $O(n^4)$ (in both time and space), with n being the number of clocks (we assume this every time we discuss complexity).

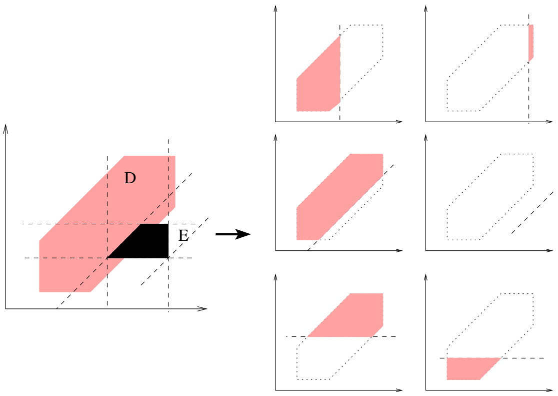


Fig. 3. Basic subtraction algorithm. The result of $D - E$ is the union of all the six zones on the right.

3 Timed Automata with Priorities

We denote by *Act* a set of actions, including the *internal action* τ and *synchronising* actions a. A synchronising action a has a complement $\bar{a} \in Act$ such that $\bar{\bar{a}} = a$. A *timed automaton* $A^i = \langle N^i, l_0^i, E^i, I^i \rangle$ is a finite state automaton with a set N^i of locations l^i, a set E^i of edges, and an initial location l_0^i. The function $I^i : N^i \mapsto \mathcal{B}(C)$ maps to each location an invariant condition. An edge of automata A^i from location l^i to $l^{i'}$ is denoted $l^i \xrightarrow{g,a,r} l^{i'}$. The edges are labelled with clock guards $g \in \mathcal{B}(C)$, actions $a \in Act$, and a set of clocks $r \subseteq C$.

We define a *network of timed automata* as the parallel composition of timed automata $A^1 | \cdots | A^n$ communicating on a set of actions *Act*, and extend these with priority orders on actions or automata. A priority order on actions is a partial order $\prec_a$. We write $a \prec_a a'$ to denote that a' has higher priority than a. Similarly for automata we write $A^i \prec_A A^j$ to denote that an automaton A^j has higher priority than A^i.

3.1 Semantics

A *state* of a network of timed automata is a pair $\langle l, v \rangle$, where l is a vector of locations l^i for each automaton, and v is a clock valuation. The initial state $\langle l_0, v_0 \rangle$ puts all automata in their initial locations l_0^i, and maps all clocks to zero.

The invariant $I(l)$ is defined as the conjunction of terms $I^i(l^i)$ for each automaton A^i. An update of the location for automata A^i is denoted by $l[l^{i'}/l^i]$ as the location vector originating from l where l^i is replaced by $l^{i'}$. For a transition t we denote by g^t the conjunction of the guards on the edges participating in that transition. Similarly we denote by r^t the union of clock sets r on the edges, and by l^t the location vector of the state generated by t.

Using a priority order $\prec$ on transitions, a transition can *block* another if it has a higher priority. From a state $\langle l, v\rangle$ a transition t is blocked according to the predicate $\mathit{block}(t) = \exists t'.t \prec t' \wedge v \models g^{t'} \wedge [r^{t'} \mapsto 0]v \models I(l^{t'})$.

The transitions between states can be either delay transitions, internal transitions, or synchronising transitions. The following rules define all possible transitions t:

– *Delay transition*: $\langle l, v\rangle \xrightarrow{d} \langle l, v \oplus d\rangle$ if $v \oplus d' \models I(l)$ for $0 \leq d' \leq d$.
– *Internal transition*: $\langle l, v\rangle \xrightarrow{a} \langle l[l^{i'}/l^i], v'\rangle$ if there is an edge $l^i \xrightarrow{g,a,r} l^{i'}$ with a local action a such that $v' = [r \mapsto 0]v$, $v \models g$, $v' \models I^i(l^{i'})$, and $\neg \mathit{block}(t)$.
– *Synchronising transition*: $\langle l, v\rangle \xrightarrow{a} \langle l[l^{i'}/l^i, l^{j'}/l^j], v'\rangle$ if there are two edges $l^i \xrightarrow{g^i,a,r^i} l^{i'}$ and $l^j \xrightarrow{g^j,\bar{a},r^j} l^{j'}$ such that $i \neq j$, $v' = [r^i \mapsto 0, r^j \mapsto 0]v$, $v \models g^i$, $v \models g^j$, $v' \models I^i(l^{i'})$, $v' \models I^j(l^{j'})$, and $\neg \mathit{block}(t)$.

With these semantics, a delay transitions can never be blocked, and no transition can be blocked by a delay transition. In Section 3.3 we show how a priority order $\prec$ over internal and synchronising transitions can be derived from the orders $\prec_a$ or $\prec_A$.

3.2 Symbolic Semantics

We use zones to define a symbolic, finite semantics for networks of timed automata with priorities. A *symbolic state* is a pair $\langle l, D\rangle$ with a location vector l and a zone D. A *symbolic transition* is denoted $\langle l, D\rangle \Longrightarrow \langle l', \hat{D}'\rangle$, where $\hat{D}'$ is a disjunction of a set of zones and $\langle l', \hat{D}'\rangle$ are all symbolic states $\langle l', D'\rangle$ such that $D' \in \hat{D}'$. The set of zones that block a transition t are describe by the predicate $\mathit{Block}(t) = \bigvee_{t \prec t'} \mathit{free}(I(l^{t'}), r^{t'}) \wedge g^{t'}$. The rules for symbolic transitions t are:

– *Symbolic delay transition*: $\langle l, D\rangle \overset{\delta}{\Longrightarrow} \langle l, D^{\uparrow} \wedge I(l)\rangle$.
– *Symbolic internal transition*: $\langle l, D\rangle \overset{a}{\Longrightarrow} \langle l[l^{i'}/l^i], \hat{D}'\rangle$ if there is an edge $l^i \xrightarrow{g,a,r} l^{i'}$ with a local action a, and $\hat{D}' = r(D \wedge g - \mathit{Block}(t)) \wedge I^i(l^{i'})$.
– *Symbolic synchronising transition*: $\langle l, D\rangle \overset{a}{\Longrightarrow} \langle l[l^{i'}/l^i, l^{j'}/l^j], \hat{D}'\rangle$ if there are two edges $l^i \xrightarrow{g^i,a,r^i} l^{i'}$ and $l^j \xrightarrow{g^j,\bar{a},r^j} l^{j'}$ such that $i \neq j$, and:
$\hat{D}' = (r^i \cup r^j)(D \wedge g^i \wedge g^j - \mathit{Block}(t)) \wedge I^i(l^{i'}) \wedge I^j(l^{j'})$.

Theorem 1 (Correctness of Symbolic Semantics). *Assume location vectors l_0, l_f, clock assignments u_0, u_f, and a set of zones $\hat{D}_f$. Let $\{u_0\}$ denote the clock constraint with a single solution u_0.*

– *(*Soundness*) whenever $\langle l_0, \{u_0\}\rangle \Longrightarrow^* \langle l_f, \hat{D}_f\rangle$ then $\langle l_0, u_0\rangle \longrightarrow^* \langle l_f, u_f\rangle$ for all $u_f \in \hat{D}^f$.*
– *(*Completeness*) whenever $\langle l_0, u_0\rangle \longrightarrow^* \langle l_f, u_f\rangle$ then $\langle l_0, \{u_0\}\rangle \Longrightarrow^* \langle l_f, \hat{D}_f\rangle$ for some $\hat{D}_f$ such that $u_f \in \hat{D}_f$.*

Proof: *By induction on the length of transition sequences. Using the zone operations of Definition 2 it can be shown that $\mathit{block}(t)$ and $\mathit{Block}(t)$ characterizes the same sets of clock valuations.* □

3.3 Priorities in UPPAAL

The priority order $\prec$ over transitions can be derived from the priority orders $\prec_a$ on actions and $\prec_A$ on automata. We describe here the order used in the UPPAAL tool [18]. For transitions t and t' with actions a and a' we derive a priority order from $\prec_a$ by defining $t \prec t'$ as $a \prec_a a'$.

Deriving a priority order on transitions from $\prec_A$ is less straightforward, as two automata with different priorities may be involved in a synchronising transition. For two transitions t and t', where t is a synchronisation between A^i and A^j such that $\neg(A^j \prec_A A^i)$, and t' is a synchronisation between $A^{i'}$ and $A^{j'}$ such that $\neg(A^{j'} \prec_A A^{i'})$, we define $t \prec t'$ to hold when:

$$(A^j \prec_A A^{j'}) \vee ((A^i \prec_A A^{i'}) \wedge \neg(A^j \prec_A A^{j'}) \wedge \neg(A^j \succ_A A^{j'}))$$

Intuitively the (weakly) higher priority processes A^j and $A^{j'}$ are compared first. If they are related they define the relation between t and t', otherwise the relation is defined by the relation between A^i and $A^{i'}$.

In a model with priorities on both actions and automata, priorities are resolved by comparing priorities on actions first. Only if they are the same, the priority order on automata is used.

4 DBM Subtraction

4.1 Improved Subtraction

The first observation from the basic algorithm given in Section 2.2 is that some splits may be avoided by taking into account only the constraints that are not redundant in the DBM. It is possible to compute the set of minimal constraints of a DBM [19] in $O(n^3)$ (the set being unique w.r.t. a given clock ordering). As this minimal set is semantically equivalent to the original set E, we use this set E_m instead: $D - E = D - E_m$. In the experiments this algorithm is the base for comparing with our other improvements since it obviously reduces splitting. Figure 4 shows the reduced subtraction by using the minimal set of constraints. We show it is worth spending this extra time because it is more important to reduce the number of DBMs in the result. The global complexity is still $O(n^4)$.

4.2 Disjoint Subtraction

The improved algorithm gives $D - E$ as a union of DBMs that overlap each other, which means there are redundant points. These points will duplicate later operations needlessly, so a second improvement is to ensure that the result is a union of disjoint DBMs. The downside of it is that inclusion checking may become worse for later generated DBMs. The problem exists even without subtraction and it is not obvious to conclude if we are improving or not on this point. The ordering of the splits affects the result but it is still guaranteed to be disjoint. The complexity is still $O(n^4)$.

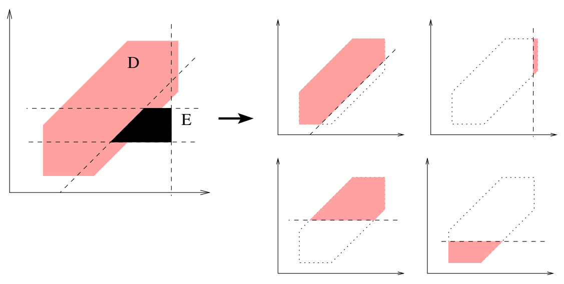


Fig. 4. Subtraction using the minimal set of constraints

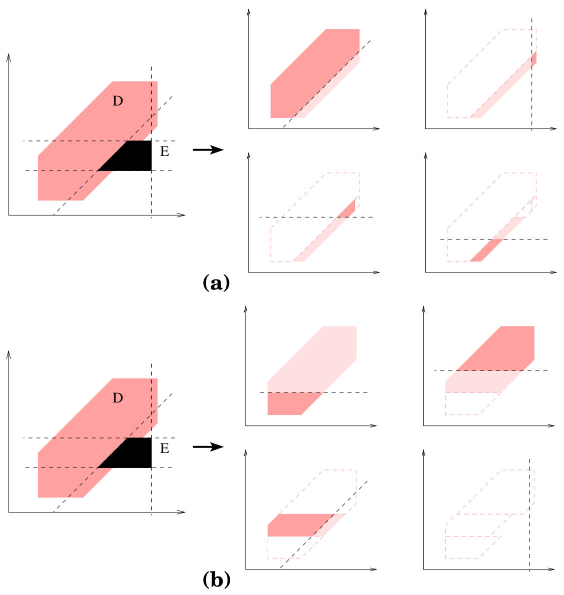


Fig. 5. Subtraction with disjoint result with two orderings (a) and (b) for splitting

Disjoint Subtraction Algorithm. We compute the subtraction $D - E$ with the minimal set of constraints $E_m \subseteq E$ as follows:

```
1. Compute $E_m$.
2. $S = false$, $R = D$
3. $\forall e_{ij} \in E_m,\ i \neq j$ :
4.     $S = S \vee (R \wedge \neg e_{ij})$
5.     $R = R \wedge e_{ij}$.
6. Return $S$.
```

R is the remainder of the subtraction and serves to compute consecutive splits. The ordering of the splits has an impact on the resulting number of DBMs as shown in Figure 5. The resulting DBMs are disjoint and the result is correct in both cases but in (a) we have 4 splits and a remainder that we will discard and in (b) we have 3 splits and no remainder (the last case discards the remainder trivially and is not a real split).

Lemma 1 (Soundness and completeness of disjoint subtraction). *The algorithm still computes the same subtraction $D-E$ and S is a union of disjoint DBMs: $\forall s_1, s_2 \in S.\ s_1 \neq s_2 \Rightarrow s_1 \cap s_2 = \emptyset$.*

4.3 Simple Improvements

There are two obvious cases illustrated in Figure 6 that we can detect before starting to compute $D-E$:

1. The negated constraint $\neg e_{ij}$ reduces D to an empty zone, which corresponds to a disjunct with false: we ignore *the constraint* e_{ij}.
2. The negated constraint $\neg e_{ij}$ has no effect on D, which means that $E \wedge D = false$ because DBMs are convex, therefor we stop and the result is D. We ignore the *whole subtraction*.

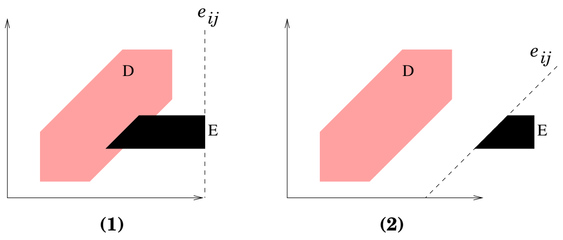


Fig. 6. Particular cases to consider to simplify subtractions: (1) ignore e_{ij} and (2) $D-E=D$

5 Reducing DBM Subtractions

Reducing subtractions means to reduce the number of splits of the operation but it is not obvious to define what an optimal split is. For a subtraction $D-E$, there may be different combinations that give the same minimal number of splits but the resulting DBMs will be used in further computations and the different combinations will give varying future splits. The problem of computing the minimal split is interesting but it is not obvious if it is possible to do it without worsening the original complexity $O(n^4)$ of the subtraction. In this section we propose a heuristic that tackles both problems: It tries to choose a good ordering to reduce the number of splits overall.

5.1 Efficient Heuristic

The idea is to use a good ordering of the constraints of E to compute the splits such that the first splits will cut the original DBM D into as large as possible DBMs to cancel the upcoming splits as soon as possible (when there is nothing left to do). We use the values $|e_{ij}| - |d_{ij}|$ to order the constraints e_{ij}, taking the smallest values first. This measures on one dimension how much the corresponding facet of E is "inside" D. The *important* trick of the algorithm is to always take the constraint with the smallest value after *every* split because the DBM changes after every split. Complexity-wise, this is equivalent to sorting in $O(n^4)$ (n^2 constraints) instead of $O(n^2 logn^2)$ but it gives better results and it makes more sense since at every iteration the previous values lose their meaning.

Algorithm. The algorithm splits D by choosing the current best e_{ij} as having the smallest $H_{E,R}(i,j) = |e_{ij}| - |r_{ij}|$.

```
1. If $\exists i, j.\ i \neq j,\ d_{ij} \leq \neg e_{ji}$ then return $D$.
2. Compute the minimal set of constraints $E_m$.
3. Initialise $R = D$ and $S = false$.
4. While $R \neq false$ do
5.    Choose $e_{ij} \in E_m,\ i \neq j$ with $min(H_{E,R}(i,j))$.
6.       if $r_{ij} \leq \neg e_{ji}$ return $S \vee R$
7.       else if $e_{ij} \geq r_{ij}$ skip
8.       else $S = S \vee (R \wedge \neg e_{ij})$
9.            $R = R \wedge e_{ij}$.
10. Return $S$.
```

Step 1 corresponds to case 2 in the preliminaries. It may seem redundant with step 6 but it is to avoid computing the minimal set of constraints in step 2.

Lemma 2 (Soundness of heuristic subtraction). *The algorithm computes the subtraction $D - E$ correctly.*

Proof: The algorithm is equivalent to the disjoint subtraction in Section 4.2 except for the ordering of the constraints and two improvements to detect the trivial cases mentioned in Section 4.3. □

5.2 Expensive Heuristic

The idea is to ignore (in addition to the previous heuristic) *facets* of E that do not intersect D. A facet of E corresponding to a constraint e_{ij} (of the form $x_i - x_j \sim b_{ij}$) is the hyper-plane $x_i - x_j = b_{ij}$ bounded by the other constraints of E. The intuition is to use the convexity of our DBMs and the fact $D - E = D - (D \cap E)$: If $D \cap E \neq \emptyset$ and a facet of E is not in $D \cap E$, i.e., it does not intersect D, then we can ignore it.

In practice, there are different cases to consider: If the constraints are strict or not and different configurations of the intersections on the corner. In addition, the exact detection of the intersection is $O(n^3)$ and it is not obvious for us if this idea is compatible with the minimal set of constraints, which is, the simple idea poses problems in practice.

To simplify, we define a new heuristic function $H_{E,R}$ that returns ∞ if $\bar{E} \wedge (x_i - x_j = b_{ij}) \cap D = \emptyset$ or the previous value $|e_{ij} - r_{ij}|$ otherwise. The condition means that we make the constraints of E non strict ($\bar{E}$) and we constrain it to be a facet that we use for testing intersection. The intersection detection is partial and is based on case 2 of Section 4.3. The new heuristic function is:

```
**function** $H'_{E,R}(i,j)$:
1.   $\forall k.\ k \neq i, k \neq j$ :
2.      if $|e_{ij} - e_{kj} - r_{ik}| \geq 0$ or $|e_{ij} - e_{ik} - r_{kj}| \geq 0$
3.      then return $+\infty$
4.   return $|e_{ij}| - |r_{ij}|$
**end**
```

We use two tricks: First we tighten e_{ji} with $\neg e_{ij}$ to compute the facet as a DBM. A specialisation of Floyd's shortest path [15] can do this in $O(n^2)$ [22]. The second trick is that we do not need all constraints but only the e_{ki} and e_{jk}, which is what the expression in the condition is doing. If the function returns ∞ then there is not intersection with the facet corresponding to the constraint e_{ij}, otherwise we do not know and we use the former heuristic value.

The function has complexity $O(n)$ which is worse than before, but it may reduce splitting further, which we investigate. The global complexity is unfortunately $O(n^5)$. We can get the detection out of the loop and get back to $O(n^4)$ but this is more complex than it seems in practice and the point is to see if it is worth the effort.

6 Experiments

We experiment[1] first with the impact of priorities in our model-checker. In practice, moderate splitting occurs so we focus on subtractions separately to answer the question of what happens on applications that cause much more splitting.

6.1 Experimenting with Priorities

We describe an experiment where we compare and evaluate models using priorities and models where priorities are manually encoded using guards and possibly extra edges. For every edge of the original automaton, the encoding is done by restricting the existing guards by removing all parts overlapping with higher priority transitions.

Experiment 1. We introduce priorities on actions in a model of the Fischer protocol [17] for mutual exclusion (Figure 7). The action a_i of the model is used to introduce priorities so that $a_i \prec a_j$ when $i < j$, and $a_i \prec \tau$ for all processes $P_1 \cdots P_N$. Since we give τ-actions the highest priority all N processes will enter location B_i, in contrast to the original model where *at most* N are in location B_i simultaneously. Also, the same process P_1 will always enter the critical section because it will be the last process to reach location C_i. The *Encoded* model is

[1] All experiments are carried on a dual-Xeon 2.8GHz with 4GB of RAM running Linux 2.6.9.

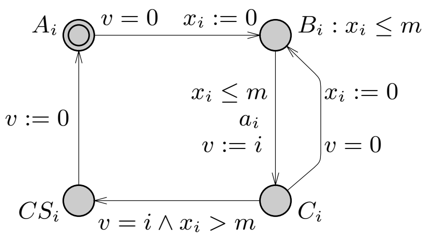


Fig. 7. A process P_i in Fischer's mutual exclusion protocol

Table 1. Measurements when model-checking N processes of the Fischer protocol

	Original		Priority		Encoded	
N	(s)	(Mb)	(s)	(Mb)	(s)	(Mb)
5	0.85	6.9	0.21	6.6	0.21	6.6
6	17.90	10.2	3.02	6.8	3.46	7.0
7	780.60	40.6	131.57	9.1	144.90	9.8

created by adding guards to edges from B_i to C_i that evaluate to false when a higher priority transition τ or $a_j \succ a_i$ is enabled.

Table 1 shows measurements of model-checking various models of the Fischer protocol, to verify that there are no deadlocks and that mutual exclusion holds. The column N is the number of processes, *Original* are the time and memory requirements for model-checking the original model without priorities, *Priority* are the corresponding numbers for the model with priorities, and *Encoded* are the numbers for a model encoding the same behaviour as the model with priorities. The results for the *Priority* and *Encoded* models are comparable, and the overhead of the priority extension is small at worst. This is encouraging since tool support for priorities makes modelling easier.

6.2 Experimenting with DBM Subtractions

Experiment 2. We have implemented a timed game reachability engine whose purpose is to find winning strategies [10] and we made a variant of it to solve jobhsop scheduling problems states as games. The timed game test example ("tgame") is the production cell [20,21] with 12 plates. The jobshop example ("jobshop") is modelled from [1] where we find a schedule for 4 jobs using 6 resources. We run variants of the prototype where we store (+strategy) and we do not store the strategy. In addition, for both experiments the main loop can reduce (+reduce) or not federations based on an inclusion checking using subtractions. Table 2 shows the results of these experiments. We give the total number of split operations (first number) as in the previous experiments with time (in seconds) and memory consumption (in megabytes).

Comparing "basic" and "reorder" shows that it is not easy to find a good ordering. Results from "disjoint" show as we claimed that reducing the size of the symbolic states may actually interfere with inclusion checking. The "expensive"

Table 2. Results of the timed game experiments (tgame) with 12 plates with and without *expensiveReduce* (reduce), and the jobshop experiments (jobshop) with or without strategy, with or without *expensiveReduce*

Algorithms	tgame12	tgame12 reduce	jobshop	jobshop strategy	jobshop reduce	jobshop reduce strategy
basic	343314 124s 319M	283334 10.9s 39.6M	55514 9.9s 35.2M	91849 10.8s 38.8M	45970 3.0s 18.7M	73018 3.6s 22.6M
reorder	351615 165s 320M	291627 10.9s 39.7M	54996 12.3s 31.9M	89763 13.4s 35.6M	47073 3.0s 18.7M	73730 3.5s 22.6M
disjoint	396053 120s 320M	356322 12.0s 40.6M	39776 11.7s 37.1M	64195 12.8s 40.8M	31592 3.2s 18.6M	51531 3.7s 22.6M
efficient	323097 **121s** 319M	275991 **10.7s** 39.7M	24105 **5.3s** 45.1M	47705 **6.8s** 48.6M	20360 2.2s 18.5M	37854 **2.5s** 22.0M
expensive	**320668** 121s 319M	**272788** 11.0s 39.7M	**23905** 7.5s 45.4M	**45428** 8.9s 48.8M	**20248** **2.1s** 18.5M	**37653** 2.6s 22.0M

heuristic gives a marginal gain considering its cost. The "efficient" heuristic is the best choice. The reduction of federations gives significant gains both in time and memory, which means it is possible to contain the splits to some extent. Still this reduction operation needs a good subtraction. Indeed, experiment 2 shows how bad it may go. Concerning memory consumption it is difficult to draw conclusions. The behaviour of the "efficient" implementation may seem an anomaly but it is explained by the fact that the prototype is using sharing of DBMs between states. The effect here is that DBMs are less shared since we know we have fewer of them.

These experiments confirm that our heuristic has an overhead but it is compensated by the reduction in splits, in particular for our "efficient" heuristic. We speculate that our priority implementation will behave reasonably well with models that generate more splitting thanks to our subtraction algorithm. Furthermore, we have implemented different reduction algorithms to merge DBMs based on subtraction. Our implementation scales well with good merging algorithms thanks to efficient subtractions. However, this is out of scope of this paper.

7 Conclusion

We have shown that our priority extension is useful for modelling and can also be used to reduce the search state-space. Furthermore, its overhead in our model-checker is reasonable. We have also shown that it is worth the extra effort for a DBM subtraction algorithm to produce fewer zones and to avoid redundancy by making the zones disjoint. The priority extension opens the door to more compact models and the support for subtraction allows us to add support for wanted features in UPPAAL such as urgent transitions with clock guards. In addition, we are improving on reduction techniques to make our model-checker more robust against splitting of DBMs.

References

1. Yasmina Abdeddaïm, Eugene Asarin, and Oded Maler. Scheduling with timed automata. *Theoretical Computer Science*, 354(2):272–300, march 2006.
2. Rajeev Alur and David L. Dill. Automata for modeling real-time systems. In *Proc. of Int. Colloquium on Algorithms, Languages, and Programming*, volume 443 of *LNCS*, pages 322–335, 1990.
3. J.G.P Barnes. *Programming in Ada, Plus and Overview of Ada 9X*. Addison–Wesley, 1994.
4. Johan Bengtsson, W. O. David Griffioen, Kre J. Kristoffersen, Kim G. Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. Automated analysis of an audio control protocol using UPPAAL. *Journal of Logic and Algebraic Programming*, 52–53:163–181, July-August 2002.
5. Johan Bengtsson and Wang Yi. Timed automata: Semantics, algorithms and tools. In W. Reisig and G. Rozenberg, editors, *Lecture Notes on Concurrency and Petri Nets*, volume 3098 of *Lecture Notes in Computer Science*. Springer-Verlag, 2004.
6. S. Bornot, G. Goessler, and J. Sifakis. On the construction of live timed systems. In S. Graf and M. Schwartzbach, editors, *Proc. of the 6*th *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 1785 of *LNCS*, pages 109–126. SPRINGER, 2000.
7. Marius Bozga, Conrado Daws, Oded Maler, Alfredo Olivero, Stavros Tripakis, and Sergio Yovine. Kronos: A Model-Checking Tool for Real-Time Systems. In *Proc. of the 10th Int. Conf. on Computer Aided Verification*, number 1427 in Lecture Notes in Computer Science, pages 546–550. Springer–Verlag, 1998.
8. Patrice Brémond-Grégoire and Insup Lee. A process algebra of communicating shared resources with dense time and priorities. *Theoretical Computer Science*, 189(1–2):179–219, 1997.
9. G. C. Buttazzo. *Hard Real-Time Computing Systems. Predictable Scheduling Algorithms and Applications*. Kulwer Academic Publishers, 1997.
10. Franck Cassez, Alexandre David, Emmanuel Fleury, Kim G. Larsen, and Didier Lime. Efficient on-the-fly algorithms for the analysis of timed games. In *To appear in CONCUR'05*, LNCS, 2005.
11. Rance Cleaveland and Matthew Hennessy. Priorities in process algebras. *Inf. Comput.*, 87(1-2):58–77, 1990.
12. J. Corbett. Modeling and analysis of real-time ada tasking programs. In *Proceedings of 15th IEEE Real-Time Systems Symposium, San Juan, P uerto Rico, USA*, pages 132–141. IEEE Computer Society Press, 1994.
13. David L. Dill. Timing assumptions and verification of finite-state concurrent systems. volume 407 of *LNCS*, pages 197–212. Springer, 1989.
14. Elena Fersman, Paul Pettersson, and Wang Yi. Timed automata with asynchronous processes: Schedulability and decidability. In J.-P. Katoen and P. Stevens, editors, *Proc. of the 8*th *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, number 2280 in Lecture Notes in Computer Science, pages 67–82. Springer–Verlag, 2002.

15. Robert W. Floyd. Acm algorithm 97: Shortest path. *Communications of the ACM*, 5(6):345, 1962.
16. Pao-Ann Hsiung and Shang-Wei Lin. Model checking timed systems with priorities. In *RTCSA*, pages 539–544, 2005.
17. Leslie Lamport. A fast mutual exclusion algorithm. *ACM Transactions on Computer Systems*, 5(1):1–11, 1987.
18. Kim G. Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a Nutshell. *Int. Journal on Software Tools for Technology Transfer*, 1(1–2):134–152, October 1997.
19. Fredrik Larsson, Kim G. Larsen, Paul Pettersson, and Wang Yi. Efficient verification of real-time systems: Compact data structures and state-space reduction. In *Proc. of the 18*th *IEEE Real-Time Systems Symposium*, pages 14–24. IEEE Computer Society Press, December 1997.
20. Claus Lewerentz and Thomas Lindner. "production cell": A comparative study in formal specification and verification. In *KORSO – Methods, Languages, and Tools for the Construction of Correct Software*, volume 1009 of *LNCS*, pages 388–416. Springer-Verlag, 1995.
21. Helmut Melcher and Klaus Winkelmann. Controller synthesis for the "production cell" case study. In *Proceedings of the second workshop on Formal methods in software practice*, pages 24–36. ACM Press, 1998.
22. Tomas Gerhard Rokicki. *Representing and Modeling Digital Circuits*. PhD thesis, Stanford University, 1993.
23. Stavros Tripakis and Sergio Yovine. Verification of the Fast Reservation Protocol with Delayed Transmission using the tool Kronos. In *Proc. of the 4*th *IEEE Real-Time Technology and Applications Symposium*. IEEE Computer Society Press, June 1998.
24. Farn Wang. RED: Model-checker for timed automata with clock-restriction diagram. In Paul Pettersson and Sergio Yovine, editors, *Workshop on Real-Time Tools, Aalborg University Denmark*, number 2001-014 in Technical Report. Uppsala University, 2001.

Symbolic Robustness Analysis of Timed Automata

Conrado Daws[1,2,⋆] and Piotr Kordy[2]

[1] Department of Applied Mathematics
[2] Formal Methods Group
Faculty of Electrical Engineering, Mathematics and Computer Science,
University of Twente, The Netherlands
conrado.daws, piotr.kordy at ewi.utwente.nl

Abstract. We propose a symbolic algorithm for the analysis of the robustness of timed automata, that is the correctness of the model in presence of small drifts on the clocks or imprecision in testing guards. This problem is known to be decidable with an algorithm based on detecting strongly connected components on the region graph, which, for complexity reasons, is not effective in practice.

Our symbolic algorithm is based on the standard algorithm for symbolic reachability analysis using zones to represent symbolic states and can then be easily integrated within tools for the verification of timed automata models. It relies on the computation of the stable zone of each cycle in a timed automaton. The stable zone is the largest set of states that can reach and be reached from itself through the cycle. To compute the robust reachable set, each stable zone that intersects the set of explored states has to be added to the set of states to be explored.

1 Introduction

Timed automata [2] are an important formal model for the specification and analysis of real-time systems. They are a simple extension of automata with real-valued variables, called clocks, whose values increase at the same rate in the control locations, and can be reset to 0 when a discrete transition is taken. By adding a certain type of constraint on clocks to the locations and edges of the automaton, one can respectively specify the time a system is allowed to remain in a control location, and when a discrete transition can be taken. Many real-time systems have been modeled using timed automata and analyzed automatically with tools like UPPAAL [10] and KRONOS [6].

A fundamental form of system analysis is the verification of safety properties, which consists in checking whether any unsafe state is reachable. This kind of analysis is performed efficiently by the tools mentioned above with well known algorithms manipulating timed constraints, called *zones*, that can be represented as a square matrix of difference bounds (DBM). The reachability analysis is based on the *idealized* semantic assumption that all clocks advance with the same speed. However, in a real implementation of a system, clocks will be likely to drift and measure time only up to some precision.

Puri first addressed this concern in [12] where he considered drifting clocks and showed that timed automata models are not robust with respect to safety properties,

⋆ This work was originally carried out at the Nijmegen Institute for Computing and Information Sciences, Radboud University Nijmegen, and supported by the European Community Project IST-2001-35304 AMETIST

meaning that a model proven to be safe under the standard ideal semantics might not be safe even if clocks drift by an arbitrarily small amount. De Wulf et al. consider a semantics, called the almost ASAP semantics [8], capturing certain notion of clock imprecision that can be translated into a syntactical enlargement of the guards. They later showed in [7] that the implementability of a model under their semantics can be decided with Puri's algorithm for robustness analysis.

Both results rely on an enlarged semantics of timed automata with either drift or imprecision of clocks. They consider the set of states that are reachable for *any* drift or imprecision. If this set contains some unsafe state, then the model is considered not to be robust or implementable. The robust reachability set can be computed with the algorithm from [12] in both cases, which are thus equivalent. The algorithm is based on the structure of the limit cycles of a timed automaton, i.e. the cyclic trajectories in the underlying timed transition system. The algorithm considers the strongly connected components of the region graph because they contain the limit cycles of the timed automaton. It adds every strongly connected component that intersects the reachability set, and its successors, to the reachability set. However, because the size of the region graph is exponential in the number of clock variables and the largest constant in the constraints, the algorithm is not effective in practice.

We propose a symbolic algorithm for computing the enlarged reachability set of a timed automaton based on the standard algorithm for symbolic reachability analysis using zones to represent symbolic states. Our algorithm relies on the computation of a *stable zone* W_σ of every progress cycle σ in the timed automaton, defined as the maximal set of clock values that have successors and predecessors through any number of iterations of σ in the timed automaton. That is, $W_\sigma = \bigcap_{i \geq 0} \mathrm{post}_\sigma^i(\mathsf{True}) \cap \bigcap_{i \geq 0} \mathrm{pre}_\sigma^i(\mathsf{True})$. The stable zone has the property of reaching and being reached from any cycle in the region graph, and hence any states in a limit cycle. We modify the standard reachability algorithm such that whenever the stable zone of a cycle intersects the standard reachable set, the whole stable zone is added to the set of states to be explored.

Related work. The robustness analysis has been extended to more general type of properties, like Büchi and LTL in [4]. Other notions of robustness have been considered in the literature, like [9,11] which impose a restriction to the type of accepted traces, as opposed to the enlargement we consider here. A different modelling based approach to implementability can be found in [1].

The remaining of the paper is organized as follows. Section 2 recalls the basic standard definitions of timed automata. The robustness problem arising from an enlarged semantics is presented in Section 3. Our contribution is the subject of Section 4, where we define the stable zone of a cycle and study its main properties, which we then use in our symbolic algorithm for robustness analysis. Finally, Section 5 concludes our presentation with a summary of the main results and a discussion on future work.

2 Timed Automata

This section briefly recalls the definitions of timed automata, their semantics, reachability analysis, and region graph.

2.1 Definitions

Definition 1 (Closed Zones). *A* closed zone *over the set of clocks $\mathcal{C}$ is a conjunction of simple constraints giving a positive lower and upper bound to the value of each clock, and a lower and upper bound to the difference between any pair of clocks. Formally,*

$$\mathcal{Z}(\mathcal{C}) = \bigwedge_{x \in \mathcal{C}} l_x \leq x \leq u_x \wedge \bigwedge_{x,y \in \mathcal{C}} x - y \leq d_{xy}$$

with $l_x, u_x \in \mathbb{N} \cup \{\infty\}$ and $d_{xy} \in \mathbb{Z} \cup \{-\infty, \infty\}$.

Zones without bounds for clock differences are called *rectangular* and denoted $\mathcal{Z}_R(\mathcal{C})$. Rectangular zones without lower bounds are called *upper* zones, and denoted $\mathcal{Z}_U(\mathcal{C})$. A *clock valuation* is a function v mapping $\mathcal{C}$ to non-negative real numbers. True denotes the zone $\bigwedge_{x \in \mathcal{C}} 0 \leq x$ satisfied by any valuation of the clocks. By $v \models z$ we will understand that if v is a clock valuation and z is a zone, then the clock values denoted by v satisfy the constraints of z.

Definition 2 (Timed Automaton). *A* timed automaton *is a tuple* $\mathsf{A} = \langle \mathcal{L}, \mathcal{C}, \mathcal{I}, \mathcal{T} \rangle$ *where*

1. *$\mathcal{L}$ is a finite set of locations representing the discrete control structure of the system*
2. *$\mathcal{C}$ is a finite set of nonnegative real-valued variables called clocks*
3. *$\mathcal{I} : \mathcal{L} \rightarrow \mathcal{Z}_U(\mathcal{C})$ are the location invariants*
4. *$\mathcal{T} \subseteq \mathcal{L} \times \mathcal{Z}_R(\mathcal{C}) \times 2^{\mathcal{C}} \times \mathcal{L}$ is a sef of edges. An edge (l, g, R, l') represents a transition from l to l' with a guard g and a set R of clocks to reset.*

2.2 Semantics

Definition 3 (Standard semantics). *Given a timed automaton $\mathsf{A} = \langle \mathcal{L}, \mathcal{C}, \mathcal{I}, \mathcal{T} \rangle$ its semantics is defined as the timed transition system $[\![\mathsf{A}]\!] = (Q, \rightarrow_t \cup \rightarrow_e)$ such that:*

- *$Q \subseteq \mathcal{L} \times \mathbb{R}_+^n$: $(l, v) \in Q$ iff $v \models \mathcal{I}(l)$*
- *$(l, v) \rightarrow_t (l, v + t)$ if $t \in \mathbb{R}_{\geq 0}$ and $v + t \models \mathcal{I}(l)$*
- *$(l, v) \rightarrow_e (l', v')$ if $e = (l, g, R, l') \in \mathcal{T}$ such that:*
  - *$v \models g$ and $v' \models I(l')$*
  - *$v'(x) = 0$ if $x \in R$, $v'(x) = v(x)$ otherwise.*

In the remaining of the paper we will use the following notations. Let $e \in \mathcal{T}$ be an edge of A, then $x \overset{e}{\Longrightarrow} y$ if and only if there exists $z, z' \in Q$ such that $x \rightarrow_t z \rightarrow_e z' \rightarrow_t y$. Let $\pi = e_1 e_2 \ldots e_n$ be a sequence of edges, then $x \overset{\pi}{\Longrightarrow} y$, meaning that there is a trajectory from x to y through π, if and only if there exist $z_1 \ldots z_{n-1} \in Q$ such that $x \overset{e_1}{\Longrightarrow} z_1 \ldots z_{n-1} \overset{e_n}{\Longrightarrow} y$.

The following lemma states that if we follow the same sequence of edges reaching two states then any linear combination of the states is also reachable. Lemma 1 follows from the convexity of the guards.

Lemma 1. *Let $\pi = e_1 e_2 \ldots e_n$ be be a sequence of edges. If $x \overset{\pi}{\Longrightarrow} x'$ and $y \overset{\pi}{\Longrightarrow} y'$, then for all $\lambda \in [0, 1]$, $\lambda x + (1 - \lambda) y \overset{\pi}{\Longrightarrow} \lambda x' + (1 - \lambda) y'$.*

2.3 Reachability Analysis

The most fundamental form of analysis of a timed automaton is the computation of its reachable state space from an initial state. The reachable state space of A from $q_0 \in Q$ under semantics$[\![\mathsf{A}]\!]$, denoted $\mathsf{Reach}([\![\mathsf{A}]\!], q_0)$, is the set of states $q \in Q$ such that $(q_0, q) \in (\rightarrow_t \cup \rightarrow_e)^\star$. We denote $y \in \mathsf{Reach}([\![\mathsf{A}]\!], x)$ by $x \Longrightarrow y$.

Safety properties can then be verified by checking if an undesired set of states Bad is reachable, i.e. if $\mathsf{Reach}([\![\mathsf{A}]\!], q_0) \cap \mathsf{Bad} = \emptyset$, which can be decided even when it is not possible to compute the reachable state space exactly. An interesting question first tackled by Puri in [12] is how robust a timed automaton model is if we relax the assumption that all clocks advance at the same speed. He showed that in some cases the verification results do not hold even for small drifts.

2.4 Region Graph

Given a timed automaton A, let k be a function, called a clock ceiling, mapping each clock $x \in \mathcal{C}$ to $k(x)$ - the largest integer c such that $(x \leq c)$ or $(c \leq x)$ is a subformula for some clock constraint appearing in A. We assume that every clock $x \in \mathcal{C}$ appears in some constraint. For a real number d, let $\langle d \rangle$ denote the fractional part of d, and $\lfloor d \rfloor$ denote its integral part. So $d = \lfloor d \rfloor + \langle d \rangle$.

Definition 4 (Clock regions). *A* clock region *is an equivalence class of the relation $\sim_k$. The equivalence relation $\sim_k$ is defined over the set of clock valuations. Two clock valuations are region equivalent denoted $v \sim_k v'$ iff all following conditions hold:*

1. *for all $x \in \mathcal{C}$ either $\lfloor v(x) \rfloor = \lfloor v'(x) \rfloor$ or $v(x) > k(x)$ and $v'(x) > k(x)$.*
2. *for all $x, y \in \mathcal{C}$ if $v(x) \leq k(x)$ and $v(y) \leq k(y)$ then $\langle v(x) \rangle \leq \langle v(y) \rangle$ iff $\langle v'(x) \rangle \leq \langle v'(y) \rangle$.*
3. *for all $x \in \mathcal{C}$ if $v(x) \leq k(x)$ then $\langle v(x) \rangle = 0$ iff $\langle v'(x) \rangle = 0$.*

We denote $[v]$ the smallest closure of the set of clock assignments region-equivalent to v. Such set is called a *closed region.*

Definition 5 (Region graph). *Given the timed transition system $[\![\mathsf{A}]\!] = (Q, \rightarrow)$ of a timed automaton A we define the corresponding (closed)* region graph *$G = (\mathsf{R}, \rightarrow_G)$ of A:*

– $\mathsf{R} = \{(l, [v]) \mid (l, v) \in Q\}$ *is a set of closed regions.*
– $\rightarrow_G \subseteq \mathsf{R} \times \mathsf{R}$*: $((l, [v]), (l', [v'])) \in \rightarrow_G$ if $(l, [v]) \neq (l', [v'])$ and either $(l, v) \rightarrow_t (l', v')$ or $(l, v) \rightarrow_e (l', v')$.*

If $q = (l, v)$ then by $[q]$ we understand closed region containing q, i.e. $[q] = (l, [v])$

3 Robustness Problem

In this section we consider the robustness or implementability of timed automata as studied in [12] and [7]. We first define a family of enlarged semantics parameterized by the drift in clocks ϵ and the imprecision in guards Δ. Following [12] we require that every cycle is a progress cycle, i.e. a cycle where each clock is reset at least once.

3.1 Enlarged Semantics

Given a timed automaton $\mathsf{A} = \langle \mathcal{L}, \mathcal{C}, \mathcal{I}, \mathcal{T} \rangle$ its enlarged semantics parameterized by $\epsilon, \Delta \in [0,1)$ is defined as the timed transition system $[\![\mathsf{A}]\!]^{\epsilon}_{\Delta} = (Q, \rightarrow_t \cup \rightarrow_e)$ such that:

Definition 6 (Enlarged semantics)

- $Q \subseteq \mathcal{L} \times \mathbb{R}^n_+$: $(l,v) \in Q$ *iff* $v \models \mathcal{I}(l)$
- $(l,v) \rightarrow_t (l,v')$ *if* $v' \models \mathcal{I}(l')$ *and* $v'(x_i) - v(x_i) \in [(1-\epsilon)t, (1+\epsilon)t]$ *for* $i = 1, \ldots, n$
- $(l,v) \rightarrow_e (l',v')$ *if* $e = (l,g,R,l') \in \mathcal{T}$ *such that:*
  - $v \models g_\Delta$[1] *and* $v' \models \mathcal{I}(l')$
  - $v'(x) = 0$ *if* $x \in R$, *otherwise* $v(x)$

Clearly, the enlarged semantics $[\![\mathsf{A}]\!]^{\epsilon}_{\Delta}$ has more reachable states than the standard semantics $[\![\mathsf{A}]\!] = [\![\mathsf{A}]\!]^{0}_{0}$. We are not interested in the reachable states for some particular ϵ or Δ, but in those states that are reachable for any ϵ or any Δ but are not reachable in the standard semantics.

Definition 7 (Robust reachability)
The robust set of states reachable from $q_0 \in Q$ with some clock drift R^*_ϵ or guard imprecision R^*_Δ are given by

$$R^*_\epsilon(\mathsf{A}, q_0) = \bigcap_{\epsilon > 0} \mathsf{Reach}([\![\mathsf{A}]\!]^{\epsilon}_{0}, q_0) \qquad R^*_\Delta(\mathsf{A}, q_0) = \bigcap_{\Delta > 0} \mathsf{Reach}([\![\mathsf{A}]\!]^{0}_{\Delta}, q_0).$$

The question is whether unsafe states Bad can be unreachable under the standard semantics but reachable under the robust semantics, i.e. $\mathsf{Reach}([\![\mathsf{A}]\!], q_0) \cap \mathsf{Bad} = \emptyset$ and $R^*_\epsilon(\mathsf{A}, q_0) \cap \mathsf{Bad} \neq \emptyset$. The following example, from [12], shows that this is indeed possible.

3.2 Example

We consider the timed automaton in Figure 1 and the initial state $(\mathsf{L1}, a = 1 \wedge b = 0)$. The parameter K is an integer taking the value 2 or 3. We want to check if the model is not safe, i.e. if location Err is reachable, which is only possible if the value of b can be larger or equal than K when entering location $\mathsf{L2}$.

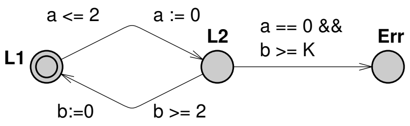


Fig. 1. A (robust?) timed automaton

Clearly, this is not possible under the standard semantics, regardless of the value of K. As can be seen from the reachable state space depicted in dark gray in Figure 2,

[1] g_Δ is the rectangular guard g extended by Δ, i.e. replacing $a \leq x \leq b$ by $a - \Delta \leq x \leq b + \Delta$.

upon entering location $\mathsf{L2}$ we must have $a = 0$ and $b \leq 1$. However, lets consider the case of clock b advancing at speed $1 + \epsilon$ in location $\mathsf{L1}$, and 1 in location $\mathsf{L2}$, whilst a advances with speed 1 in both locations. Then there is a trajectory such that upon entering location $\mathsf{L2}$ for the k-th time, $a = 0$ and $b = (1 + \epsilon)^k$. Moreover, because $a \geq 0$ in $\mathsf{L1}$, we have that $b \leq 2(1 + \epsilon)/(1 - \epsilon)$ when entering $\mathsf{L2}$. So for any ϵ and sufficiently many iterations, location Err becomes reachable if $K = 2$, but not if $K = 3$ and $\epsilon < 1/5$.

So, although the model for $K = 2$ is considered safe under the standard idealized semantics, it is not under the robust semantics because the unsafe location will be reachable for *any* drift or imprecision in the measurement of clocks. In other words, the model is not robust or implementable. On the other hand, the same model is robust or implementable for $K = 3$ provided that ϵ is small enough.

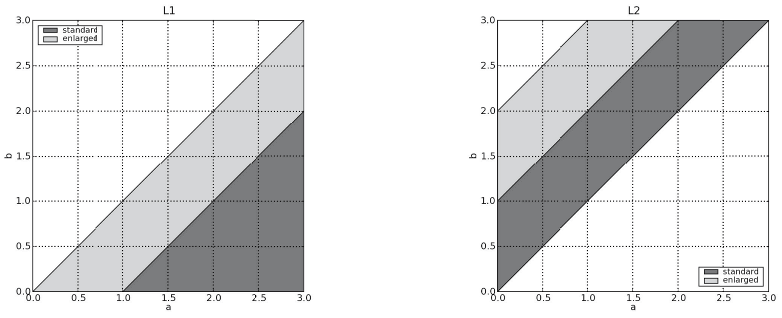


Fig. 2. Reachable state space from $(\mathsf{L1}, a = 1 \wedge b = 0)$ with the standard (dark) and enlarged (light) semantics

3.3 Algorithm

Algorithm 1 is a slightly modified version of the algorithm in [12] to compute iteratively the set J^* on the region graph of a timed automaton A. We first compute the reachable regions from a given initial state $[q_0]$. Then, for every region s in the strongly connected components of the region graph that intersects the current J^*, the regions reachable from s are added to J^*.

Theorem 1 (from [12] and [7]). *Algorithm 1 computes the robust reachable state space of* A *from* r_0 *under the enlarged semantics, with either drifting clocks [12] or enlarged guards [7].*

$$J^* = R^*_\epsilon(\mathsf{A}, r_0) = R^*_\Delta(\mathsf{A}, r_0)$$

The proof of this theorem (shown in [12] and [7]) relies on the structure of the limit cycles of the timed automaton.

4 Symbolic Robustness Analysis

The robust semantics of a timed automaton depends on the structure of its limit cycles. Algorithm 1 computes the robust reachable state space by adding the reachable cycles

Algorithm 1. Algorithm computing $R^*_\epsilon(\mathsf{A}, r_0)$

```
Input: A timed automaton $\mathsf{A}$ and initial region $r_0$
Output: The set $J^* = R^*_\epsilon(\mathsf{A}, r_0)$
EnlargedReach($\mathsf{A}, r_0$)
(1)    Construct the region graph $G$ of timed automaton $\mathsf{A}$
(2)    $S \leftarrow$ StronglyConnectedComponents($G$)
(3)    $J^* \leftarrow$ Reach($G, r_0$)
(4)    while there exists $s \in S$ such that $s \not\subseteq J^*$ and $s \cap J^* \neq \emptyset$
(5)        $J^* \leftarrow J^* \cup$ Reach($G, s$)
(6)        $S \leftarrow S \setminus s$
(7)    return $J^*$
```

in the region graph (which contain the limit cycles). But this algorithm is not effective in practice because of the size of the region graph.

We propose a symbolic algorithm to compute the enlarged reachable state space using zones to represent symbolic states. For that, we consider in this section a cycle $\sigma = e_1 \dots e_n$ of a timed automaton, with $e_i = (l_i, g_i, r_i, l'_i)$, as a sequence of edges such that $l'_i = l_{i+1}$ for $i = 1 \dots n-1$ and $l_1 = l'_n$. We assume that the cycle is a progress cycle, meaning that each clock is reset at least once.

4.1 Limit Cycles

Limit cycles are cyclic trajectories in the underlying timed transition system of a timed automaton, without superfluous 0-time self loops. A state in a limit cycle can return to itself after one or more iterations of a cycle in the timed automaton.

Definition 8 (Return Map). *Let σ be a cycle in timed automaton A. The return map of a state q is the set of states reachable from q after one cycle σ*

$$R_\sigma(q) = \{q' \mid q \overset{\sigma}{\Longrightarrow} q'\}.$$

Definition 9 (Limit Cycles). *The set of states which can return back to themselves after $i > 0$ iterations of the cycle σ is $L^i_\sigma = \{q \mid q \in R^i_\sigma(q)\}$, and the set of states with limit cycles through σ is $L_\sigma = \bigcup_{i>0} L^i_\sigma$.*

Lemma 2 shows that the set of states in limit cycles is convex because the convex combination of two limit cycles is also a limit cycle. Let $\overset{\sigma^+}{\Longrightarrow}$ be the transitive closure of $\overset{\sigma}{\Longrightarrow}$.

Lemma 2. *Let σ be a cycle in A. If $x \overset{\sigma^+}{\Longrightarrow} x$ and $y \overset{\sigma^+}{\Longrightarrow} y$, then for all $\lambda \in [0,1]$, $\lambda x + (1-\lambda) y \overset{\sigma^+}{\Longrightarrow} \lambda x + (1-\lambda) y$.*

Proof. If $x \overset{\sigma^+}{\Longrightarrow} x$ then there exists $m \geq 1$ such that $x \overset{\sigma^m}{\Longrightarrow} x$. Similarly, there exists $n \geq 1$ such that $y \overset{\sigma^n}{\Longrightarrow} y$. So $x \overset{\sigma^{mn}}{\Longrightarrow} x$ and $y \overset{\sigma^{mn}}{\Longrightarrow} y$. Therefore, from Lemma 1, for all $\lambda \in [0,1]$ $\lambda x + (1-\lambda) y \overset{\sigma^{mn}}{\Longrightarrow} \lambda x + (1-\lambda) y$.

Several properties of limit cycles of timed automata are given in [12]. For instance, the set of states with limit cycles in a cycle of the region automaton is a non-empty region. It follows, by convexity of the limit cycles, that L_σ is a zone. An important property of limit cycles is that for any state in a cycle of the region graph, there is a state in the limit cycles of its region that can reach it, and a state that can be reached from it.

Lemma 3 (Lemma 11 in [12]). *Consider a cycle $c = c_0 c_1 \dots c_N$ in the region graph. Then, for any $x \in c_0$, there exist $u, v \in L_c$ such that $x \Longrightarrow u$ and $v \Longrightarrow x$.*

4.2 Region Graph Cycles

Definition 10 (Regions with Cycles). *Let σ be a cycle in timed automaton A. We define the set of regions with cycles through σ in the region graph of A as $C_\sigma = \{r \in \mathsf{R} \mid \exists q, q'. \, r = [q] = [q'] \,\mathrm{and}\, q \overset{\sigma^+}{\Longrightarrow} q'\}$.*

Lemma 4 says that for any two cycles in the region graph following a cycle in the timed automaton, we can find a sequence of regions touching each other and each of these regions is in a cycle of the region graph.

Lemma 4. *Given a cycle σ in the timed automaton, for any two regions $c, c' \in C_\sigma$ there exists a finite sequence of regions $c_0 \dots c_n \in C_\sigma$ such that for all $0 \le i < n$, $c_i \cap c_{i+1} \neq \emptyset$, $c_0 \cap c \neq \emptyset$ and $c_n \cap c' \neq \emptyset$.*

Proof. We know that each region in a cycle of the region graph contains a state with a limit cycle. Let $x \in c$ and $y \in c'$ have limit cycles. Thus, state $\lambda x + (1-\lambda) y$ is in a limit cycle and its region in a cycle in the region graph, for any $\lambda \in [0,1]$, which implies the existence of regions with the required property.

Remark 1. The set of states in C_σ is not necessarily convex, as illustrated in the example in Section 4.5.

4.3 Stable Zone

Let $X \subseteq Q$ be a set of states and σ a cycle of A. We denote $\mathrm{pre}_\sigma(X) = \{q \mid \exists q' \in X. q \overset{\sigma}{\Longrightarrow} q'\}$ the set of predecessors of X through σ, and $\mathrm{post}_\sigma(X) = \{q \mid \exists q' \in X. q' \overset{\sigma}{\Longrightarrow} q\}$) the set of successors of X through σ.

Definition 11 (Stable Zone). *The* stable zone *of a cycle σ in a timed automaton is the zone*

$$W_\sigma = \nu X. \, \mathrm{post}_\sigma(X) \cap \nu X. \, \mathrm{pre}_\sigma(X)$$

The stable zone characterizes those states that have a predecessor and a successor after any number of σ iterations. In the finite lattice of zones defined using constants smaller than the largest constant appearing in the timed automaton, the greatest fixed points can be computed by iteration from True. Hence the σ-stable zone can be computed as

$$W_\sigma = \bigcap_{i \ge 0} \mathrm{post}^i_\sigma(\mathsf{True}) \cap \bigcap_{i \ge 0} \mathrm{pre}^i_\sigma(\mathsf{True})$$

Example 1. The stable zone of the cycle σ =L1-L2-L1 in the timed automaton in Figure 1 is:

$$\begin{aligned} W_\sigma &= (0 \le a \le 2 \wedge 0 \le b \le 3 \wedge -2 \le b - a \le 3) \cap \\ &\quad (0 \le a \le 3 \wedge 0 \le b \le 3 \wedge -3 \le b - a \le 0) \\ &= (0 \le a \le 2 \wedge 0 \le b \le 2 \wedge -2 \le b - a \le 0) \end{aligned}$$

where the fixpoints are computed in 2 iterations. It can be checked that those are also the states with a limit cycle through σ. However this is not always the case.

Moreover, for any state in W_σ there exist a successor and a predecessor through any number of iterations of σ that also belong to W_σ. This property follows from the following lemma that characterizes the stable zone as the maximal set of states that can reach and be reached from itself.

Lemma 5. $W_\sigma = \nu X.(\mathrm{post}_\sigma(X) \cap \mathrm{pre}_\sigma(X))$

Proof. Let $Y_1 = \nu X.\,\mathrm{post}_\sigma(X)$, $Y_2 = \nu X.\,\mathrm{pre}_\sigma(X)$ and $Z = \nu X.(\mathrm{post}_\sigma(X) \cap \mathrm{pre}_\sigma(X))$. $Z \subseteq \mathrm{post}_\sigma(Z)$, therefore $Z \subseteq Y_1$. Similarly, $Z \subseteq Y_2$. Hence $Z \subseteq W_\sigma$. On the other hand, for all $y \in Y_1 \cap Y_2$, there exist $y_1 \in Y_1$ and $y_2 \in Y_2$ such that $y_1 \overset{\sigma}{\Longrightarrow} y \overset{\sigma}{\Longrightarrow} y_2$. Since $y \in Y_2$, y has suctcessors through any number of σ iterations, and so does y_1, hence $y_1 \in Y_2$ and $y \in \mathrm{post}_\sigma(Y_1 \cap Y_2)$. Similarly, $y_2 \in Y_1$ and $y \in \mathrm{pre}_\sigma(Y_1 \cap Y_2)$. So $Y_1 \cap Y_2 \subseteq \mathrm{post}_\sigma(Y_1 \cap Y_2) \cap \mathrm{pre}_\sigma(Y_1 \cap Y_2)$, hence $W_\sigma \subseteq Z$.

The following lemma states that the stable zone of a cycle σ of a timed automaton contains the cycles in the region graph through σ, which we know from [12] contain the limit cycles in σ. Moreover, the inclusions can be strict. An example where the first inclusion is strict is given in [13], and an example showing that both inclusions can be strict can be found in Section 4.5. We will abuse notation and consider C_σ as the set of states in the regions of C_σ.

Lemma 6. $L_\sigma \subseteq C_\sigma \subseteq W_\sigma$

Proof. Let $c \in C_\sigma$ be a region.Then, for any $q \in c$ there exist $q_1, q_2 \in c$ such that $q_1 \overset{\sigma^+}{\Longrightarrow} q \overset{\sigma^+}{\Longrightarrow} q_2$ so q has successors and predecessors through any number of σ iterations. Hence, $q \in W_\sigma$ and $c \subseteq W_\sigma$.

An important property in Puri's theory is that even though not every state in a cycle in the region graph is itself in a limit cycle, it can always reach a state in a limit cycle, and be reached by a state in a limit cycle. In a similar way, not every state in the stable zone is in a cycle in the region graph, but it can always reach a state in a cycle in the regiont graph, and be reached by a state in a cycle in the region graph. Lemma 7 formalizes this property.

Lemma 7. *For all $q \in W_\sigma$, there exist q_1, q_2 such that $q_1 \overset{\sigma^+}{\Longrightarrow} q \overset{\sigma^+}{\Longrightarrow} q_2$ and $[q_1], [q_2] \in C_\sigma$.*

Proof. Let $q \in W_\sigma$ be a state in the stable zone of σ. From Lemma 5 there exist $x_1 \dots x_n \in W_\sigma$ such that $x_n \overset{\sigma}{\Longrightarrow} \dots x_1 \overset{\sigma}{\Longrightarrow} q$ for any $n \geq 1$. Since q can be reached in any number of iterations of σ, and the number of regions is finite, then, for n sufficiently large, there exist x_i, x_j such that $[x_i] = [x_j]$ and then we take $q_1 = x_i$. A similar reasoning shows the existence of q_2.

4.4 Symbolic Robustness Algorithm

Algorithm 1 computes the robust reachability set by adding iteratively to the reachable state space all regions reachable from a strongly connected component of the region graph, when some region in the SCC intersects the current reachable state space.

We propose an algorithm to compute symbolically the robust set of reachable states of a timed automaton. Algorithm 2 is based on the standard symbolic algorithm [5,3] to compute the reachable state space of a timed automaton as implemented in the real-time model checkers KRONOS and UPPAAL. A symbolic state is a pair $\langle l, z\rangle$ of a location l and a zone z. First the algorithm computes the stable zone of each cycle of the timed automaton. Each time a new symbolic state representing states not visited yet is chosen, its successors are added to the waiting list, until the latter is empty. The difference with the standard algorithm resides in lines 9–11 where we check if the zone of the currently visited symbolic state intersects any of the stable zones of a cycle starting in its location, in which case we add the stable zone and its time-successors to the waiting list.

Algorithm 2. Symbolic algorithm computing $R^*_\epsilon(\mathsf{A}, r_0)$

Input: A timed automaton A and initial region $r_0 \in R_\mathsf{A}$
Output: The symbolic state set $\mathsf{Passed} = R^*_\epsilon(\mathsf{A}, r_0)$

```
SYMBENLARGEDREACH(A)
(1)  Compute $W_\sigma$ for every cycle $\sigma$ in A
(2)  $\mathsf{Wait} = \{r_0\}$
(3)  $\mathsf{Passed} = \emptyset$
(4)  while $\mathsf{Wait} \neq \emptyset$
(5)      pop $\langle l, D\rangle$ from Wait
(6)      if $D \not\subseteq D'$ for all $\langle l', D'\rangle \in \mathsf{Passed}$
(7)          $\mathsf{Passed} \leftarrow \mathsf{Passed} \cup \{\langle l, D\rangle\}$
(8)          foreach $\langle l', D'\rangle \in \textsc{Succ}(\langle l, D\rangle)$
(9)              $\mathsf{Wait} \leftarrow \mathsf{Wait} \cup \{\langle l', D'\rangle\}$
(10)         foreach $W_\sigma$ with $\sigma$ starting in $l$
(11)             if $D \cap W_\sigma \neq \emptyset$
(12)                 $\mathsf{Wait} \leftarrow \mathsf{Wait} \cup \{\textsc{TimeSucc}(\langle l, W_\sigma\rangle)\}$
(13) return Passed
```

Theorem 2 states that Algorithm 2 computes the set R^*_ϵ. We prove this result by showing that this algorithm computes the same set of states J^* as Algorithm 1 from Puri.

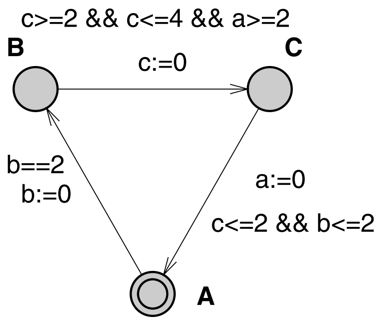
Theorem 2. *Let* A *be a timed automaton,* $r_0 \in R_\mathsf{A}$ *an initial region, and* W^* *the set of states returned by Algorithm 2. Then* $W^* = R^*_\epsilon(\mathsf{A}, r_0)$.

Proof. Algorithm 1 adds cycles in the region graph to the reachable state space. From Lemma 6 we know that all cycles in the region graph are included in the stable zone and hence will also be added by Algorithm 2, so $J^* \subseteq W^*$.

On the other hand, if some W_σ is added in Algorithm 2, then $W_\sigma \cap D \neq \emptyset$ for some reachable $\langle l, D\rangle$. From Lemma 7 we know that there exists some $c \in C_\sigma$ that is reachable from $W_\sigma \cap D$. So $[c]$ will be added by Algorithm 1. From Lemma 4, the whole C_σ must be added. Finally, W_σ is also added because it is reachable from C_σ by Lemma 7. Hence, $W^* \subseteq J^*$.

4.5 Example with Strict Inclusions

The timed automaton in Figure 3 (left) shows that the inclusions $L_\sigma \subseteq C_\sigma \subseteq W_\sigma$ can be strict, and that the set C_σ is not necessarily convex (right).



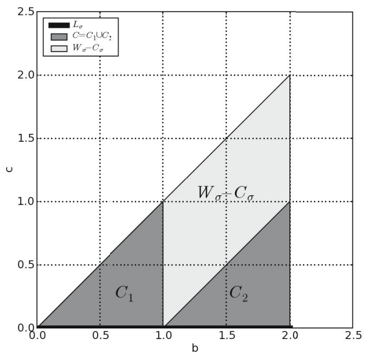


Fig. 3. Timed automaton (left) showing that $L_\sigma \subsetneq C_\sigma \subsetneq W_\sigma$ (right)

Strict inclusions and non convexity of C_σ. We only consider set of states when entering location A, that is, with $a = 0$ (the remaining reachable states can be reached from these states). The reader can check that the states with limit cycles L_σ, the states in cycles in the region graph C_σ, and the stable zone W_σ are given by the following equations, represented in Fig.3 (right):

$$\begin{aligned}
L_\sigma &= (A, a = 0 \wedge 0 \le b \le 2 \wedge c = 0)\\
C_1 &= (A, a = 0 \wedge 0 \le b \le 1 \wedge 0 \le c \le 1 \wedge 0 \le b - c \le 1)\\
C_2 &= (A, a = 0 \wedge 1 \le b \le 2 \wedge 0 \le c \le 1 \wedge 1 \le b - c \le 2)\\
C_\sigma &= C_1 \cup C_2\\
W_\sigma &= (A, a = 0 \wedge 0 \le b \le 2 \wedge 0 \le c \le 2 \wedge 0 \le b - c \le 2)
\end{aligned}$$

The fact that a set of states belongs or not to a cycle in the region graph can be observed computing the corresponding reachability set. In this case we have:

$$\begin{aligned}
\mathsf{Reach}(L_\sigma) &= W_\sigma & \mathsf{Reach}(C_1) &= W_\sigma\\
\mathsf{Reach}(W_\sigma) &= W_\sigma & \mathsf{Reach}(C_2) &= C_2\\
\mathsf{Reach}(C_\sigma) &= W_\sigma & \mathsf{Reach}(W_\sigma \setminus C_\sigma) &= C_2
\end{aligned}$$

Comparing both algorithms. Theorem 2 shows that both the original algorithm on the region graph, and our symbolic algorithm using the stable zone are equivalent. It is easy to see that any region added to the reachability set by Algorithm 1 will be also added by our algorithm because the stable zone contains all the cycles in the region graph. We illustrate with this example why the converse is also true.

Let's assume state $(A, (0, 2, 2))$ is reachable. Since it belongs to the stable zone, the whole stable zone will be added by our algorithm. Algorithm 1 will first only add the states reachable from $(A, (0, 2, 2))$, that is $(A, (0, 2, 0)) \in C_2$. So C_2 will be added next, then also C_1 because $C_1 \cap C_2 \neq \emptyset$ (a particular case of Lemma 4). Finally, $\mathsf{Reach}(C_1) = W_\sigma$, so the whole W_σ will be added.

5 Conclusions

We proposed a *symbolic algorithm* for computing the robust reachability set of a timed automaton based on the standard algorithm for symbolic reachability analysis of timed automata, using zones to represent symbolic states. Although the worst case complexity of symbolic algorithms is the same as for the region graph, symbolic algorithms are known to be more efficient in practice. Moreover, our symbolic algorithm is easy to implement and integrate within existing formal frameworks for the validation of real-time systems.

Our algorithm relies on the computation of the stable zone of each progress cycle in the timed automaton. The stable zone of a cycle is the maximal set of clock values that have a successor and a predecessor through any number of iterations of the cycle. All cycles in the region graph following the same cycle in the timed automaton are connected to each other through the limit cycles. Moreover, every point in the stable zone can reach a cycle in the region graph, and be reached from a cycle in the region graph. Based on these facts, we modified the standard reachability algorithm such that whenever the stable zone of a cycle intersects the set of computed reachable states, the stable zone is added to the set of states to be explored. We showed that our zone based algorithm is equivalent to the one from Puri operating on the region graph to compute the robust reachabilty set.

We implemented the computation of the stable zone of a cycle and applied it to the simple examples in [12] to validate our results. However, we need to implement the robust reachability algorithm in order to assess its effectiveness in handling complex timed automata models. The modifications to the standard reachability algorithm are quite straightforward to implement in tools like UPPAAL or KRONOS.

In the future we are interested in the study of sufficient conditions for a timed automaton model to be robust which can be checked more efficiently, as well as in devising meaningful model transformation techniques to make a model robust.

References

1. K. Altisen and S. Tripakis. Implementation of timed automata: an issue of semantics or modeling? Technical report, Verimag, Centre Équation, 38610 Gières, June 2005.
2. R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

3. J. Bengtsson and W. Yi. Timed automata: Semantics, algorithms and tools. In W. Reisig and G. Rozenberg, editors, *In Lecture Notes on Concurrency and Petri Nets*, Lecture Notes in Computer Science vol 3098. Springer–Verlag, 2004.
4. P. Bouyer, N. Markey, and P.-A. Reynier. Robust model-checking of linear-time properties in timed automata. In J. R. Correa, A. Hevia, and M. Kiwi, editors, *Proceedings of the 7th Latin American Symposium on Theoretical Informatics (LATIN'06)*, volume 3887 of *Lecture Notes in Computer Science*, pages 238–249, Valdivia, Chile, Mar. 2006. Springer.
5. C. Daws. *Vérification de systèmes temporisés: de la théorie à la pratique*. PhD thesis, Institut National Polytechnique de Grenoble, 20 Oct. 1998.
6. C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool Kronos. In R. Alur, T. A. Henzinger, and E. D. Sontag, editors, *Hybrid Systems III*, volume 1066 of *Lecture Notes in Computer Science*, pages 208–219. Springer-Verlag, 1996.
7. M. De Wulf, L. Doyen, N. Markey, and J.-F. Raskin. Robustness and implementability of timed automata. In *Formats - FTRTFT*, pages 118–133, 2004.
8. M. De Wulf, L. Doyen, and J. Raskin. Almost ASAP Semantics: from timed models to timed implementations, 2004.
9. V. Gupta, T. Henzinger, and R. Jagadeesan. Robust Timed Automata. In *Proc. Int. Work. Hybrid and Real-Time Systems (HART'97)*, volume 1201 of *LNCS*, pages 331–345. Springer, 1997.
10. K. Larsen, P. Pettersson, and W. Yi. Uppaal in a nutshell. *Software Tools for Technology Transfer*, 1(1+2):134–152, 1997.
11. J. Ouaknine and J. Worrell. Revisiting digitization, robustness, and decidability for timed automata. In *LICS '03: Proceedings of the 18th Annual IEEE Symposium on Logic in Computer Science*, page 198, Washington, DC, USA, 2003. IEEE Computer Society.
12. A. Puri. Dynamical Properties of Timed Automata. In *FTRTFT '98: Proceedings of the 5th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 210–227, London, UK, 1998. Springer-Verlag.
13. A. Puri. Dynamical Properties of Timed Automata. *Discrete Event Dynamic Systems-Theory and Applications*, 10(1-2):87–113, Jan 2000.

Coping with the Parallelism of BitTorrent: Conversion of PEPA to ODEs in Dealing with State Space Explosion

Adam Duguid

Laboratory of Computer Science, The University of Edinburgh
a.j.duguid@sms.ed.ac.uk

Abstract. The Performance Evaluation Process Algebra (PEPA) language is a stochastic process algebra, generating Continuous Time Markov Chains (CTMC) to allow quantitative analysis. Protocols such as BitTorrent are highly parallel in nature, and represent one area where CTMC analysis is limited by the well-known state space problem. The number of unique states each client can exist in, and the number of clients required to accurately model a typical BitTorrent network preclude the use of CTMCs. Recent work has shown that PEPA models also allow the derivation of an activity matrix, from which ODE and stochastic simulation models, as alternative forms of analysis, are possible. Using this technique, a BitTorrent network is created, analysed, and the results compared against previous BitTorrent models.

1 Introduction

The PEPA[1] language originated, in part, from Calculus of Communicating Systems (CCS), allowing the generation of a labelled transition graph with rates based on the exponential distribution. From this graph a CTMC can be obtained and the steady state gained through standard numerical techniques. CTMCs produce exact results, in the sense that every possible state of the system is accounted for and that all probabilities are correct for the given rates. The use of CTMCs assumes that the subsystems behaviour can, to some degree of accuracy, be described with exponential distributions and behaviour is independent of time.

One particular weakness of CTMCs is the size of the model which can be efficiently analysed while still remaining tractable. As the number of components increases, especially components that act independently from one another, the size of the state space can rapidly expand beyond tractable limits. Techniques such as model simplification and state aggregation can allow the analysis of larger models to some extent but the limitations still remain.

Recent work has introduced process algebra (in this instance π calculus) to the area of systems biology[2,3], a field interested in the dynamic pathways of biological systems. However, with the desire to model large numbers of proteins or receptors, the dominant approach within systems biology has been to use

Ordinary Differential Equations (ODEs). ODEs are continuous-time, continuous-state and deterministic in nature. In addition, the area of ODEs is well researched and so supported by a range of solvers. The modelling of biological systems can also be conducted through stochastic simulation such as Gillespie's Stochastic Simulation Algorithm (SSA)[4]. Gillespie's argument for the use of a continuous-time, discrete-state stochastic simulation centres on physical accuracy with the real system, in this case chemical reactions. Both approaches scale differently to CTMCs, where the numbers of each component do not affect the complexity of the model in the same way.

This has led to work mapping PEPA to ODEs[5,6] to model the Extracellular signal Regulated Kinase (ERK) signalling pathway from within a process algebra. Two structures were defined, reagent-centric and pathway-centric, offering different views of the system. Both structures allow the derivation of ODEs from the underlying PEPA model, offering an alternative style of analysis from CTMCs. This same derivation process can be used to allow the use of SSA.

The PEPA language will be briefly covered, followed by a more complete description of the mapping from PEPA to ODEs. The salient parts of the BitTorrent protocol will be detailed, and a PEPA model constructed using the reagent-centric approach. The BitTorrent protocol is used due to its parallel nature of communication between entities, which resists analysis with CTMCs. Attempting to model even 20 peers can easily lead to a state space as large as 100^{20} with CTMCs while remaining tractable when using ODEs or SSAs. Finally, these results will be compared against an existing model of BitTorrent.

2 PEPA

A model defined in the PEPA language consists of a number of components representing different agents or entities in the real system. The components interact with each other through a small set of combinators as shown below.

$$P ::= (\alpha, r).P \mid P + Q \mid P \underset{L}{\bowtie} Q \mid P/L \mid A$$

Prefix $(\alpha, r).P$ represents a component that can perform an activity α at rate r (sampled from the negative exponential distribution) before it transitions to a component of type P.

Choice $P + Q$ represents a component which is either of type P or Q. Which is chosen is based on a race condition on the first activity of each component.

Co-operation $P \underset{L}{\bowtie} Q$ requires that if components P and Q can both perform an activity α (where $\alpha \in L$), then for either component to perform α, they must both perform it together. Where P or Q are capable of an activity not in the set L then these can occur independently.

$$S_1 \overset{def}{=} (u,\alpha).S_2 + (v,\beta).S_3$$
$$S_2 \overset{def}{=} (w,\gamma).S_1$$
$$S_3 \overset{def}{=} (x,\delta).S_1$$
$$T_1 \overset{def}{=} (v,\beta).T_2$$
$$T_2 \overset{def}{=} (x,\delta).T_1$$

$$S_1 \underset{\{v,x\}}{\bowtie} T_1$$

$$\begin{tikzcd}
S_3 \underset{\{v,x\}}{\bowtie} T_2 \arrow[d, shift left, "{(x,\delta)}"] \\
S_1 \underset{\{v,x\}}{\bowtie} T_1 \arrow[u, shift left, "{(v,\beta)}"] \arrow[d, shift left, "{(u,\alpha)}"] \\
S_2 \underset{\{v,x\}}{\bowtie} T_1 \arrow[u, shift left, "{(w,\gamma)}"]
\end{tikzcd}$$

Fig. 1. Toy example highlighting the PEPA language

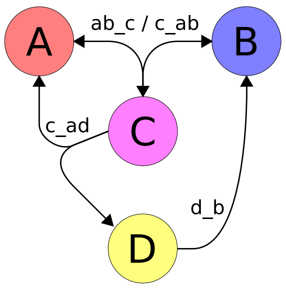
Hide P/L alters component P such that any activity in the set L is hidden from the rest of the model and cannot be synchronised on.
Constant $A \overset{def}{=} P$ assigns names to components.

Figure 1 shows an example in the PEPA language with the choice and co-operation in use. In this example, the component S_1 can freely change into S_2 via the activity u. Activity w can also occur independently. The co-operation over activities v and x mean that both the S_i and T_i components must be in the correct state and both must transition together i.e. $S_1 \underset{\{v,x\}}{\bowtie} T_1 \xrightarrow{(v,\beta)} S_3 \underset{\{v,x\}}{\bowtie} T_2$ is the only transition possible with activity v. A more complete description of the language can be found in [1], but for the purposes of describing the reagent-centric approach, understanding of the prefix and choice combinators is enough.

2.1 The Reagent-Centric Approach

The reagent-centric approach in its coarsest form defines two states for each component, those being *high* and *low*. Through activities that *consume* the resource or component, the component transitions from a high to low state. Conversely, activities that replenish move a component from a low to high state. This approach is in keeping with the component view of PEPA, where the focus is on the component and the activities possible in that state. The Prefix combinator records the reaction that causes this change, and the rate that the reaction occurs. The Choice combinator allows any one component to be associated with any number of reactions.

Figure 2 shows a small network, and the PEPA reagent-centric model that describes the graphical representation. In this example the PEPA components are A,B and C, and are tagged with H and L to designate the high and low concentrations. By stipulating unique activity names for each reaction, the direction of change (high to low or vice versa) can be used to create a list of reactions with components either being consumed or created through an activity. The PEPA definitions in Fig. 2 give rise to four reactions shown here in the Chemical Model Definition Language (CMDL) $W, X \rightarrow Y, Z$. W is the name for the reaction, $X = \{x_1 + ... + x_n\}$ lists all the components that are consumed in this reaction. Y is a list in the same format as X representing those



$$
\begin{aligned}
\mathrm{A_H} &\stackrel{\mathit{def}}{=} (ab\_c, \alpha).\mathrm{A_L}\\
\mathrm{A_L} &\stackrel{\mathit{def}}{=} (c\_ab, \beta).\mathrm{A_H} + (c\_ad, \gamma).\mathrm{A_H}\\
\mathrm{B_H} &\stackrel{\mathit{def}}{=} (ab\_c, \alpha).\mathrm{B_L}\\
\mathrm{B_L} &\stackrel{\mathit{def}}{=} (c\_ab, \beta).\mathrm{B_H} + (d\_b, \delta).\mathrm{B_H}\\
\mathrm{C_H} &\stackrel{\mathit{def}}{=} (c\_ab, \beta).\mathrm{C_L} + (c\_ad, \gamma).\mathrm{C_L}\\
\mathrm{C_L} &\stackrel{\mathit{def}}{=} (ab\_c, \alpha).\mathrm{C_H}\\
\mathrm{D_H} &\stackrel{\mathit{def}}{=} (d\_b, \delta).\mathrm{D_L}\\
\mathrm{D_L} &\stackrel{\mathit{def}}{=} (c\_ad, \gamma).\mathrm{D_H}
\end{aligned}
$$

$$((\mathrm{A_H} \underset{\{ab\_c, c\_ab\}}{\bowtie} \mathrm{B_H}) \underset{\{ab\_c, c\_ab, c\_ad\}}{\bowtie} \mathrm{C_L}) \underset{\{c\_ad, d\_b\}}{\bowtie} \mathrm{D_L}$$

Fig. 2. PEPA reagent-centric example

components increased by this reaction. The last part of the reaction, Z, defines the rate constant from which the reaction rate is derived.

$$
\begin{array}{ll}
ab\_c,\ \mathrm{A} + \mathrm{B} \rightarrow \mathrm{C}\ , \alpha & \quad c\_ab,\ \mathrm{C} \rightarrow \mathrm{A} + \mathrm{B}\ , \beta\\
c\_ad,\ \mathrm{C} \rightarrow \mathrm{A} + \mathrm{D}\ , \gamma & \quad d\_b,\ \ \mathrm{D} \rightarrow \mathrm{B} \qquad , \delta
\end{array}
$$

The reaction *ab_c* consists of two reactants and one product. From the PEPA definition in Fig. 2, it can be seen that both the components A and B can transition from a high to low state via the activity/reaction *ab_c*, thus placing them as the two reactants of reaction *ab_c*. Similarly, component C can transition from a low to high state by reaction *ab_c* placing it as a product of this particular reaction. By iterating through all of the prefixes for each definition, the reactions can be constructed.

The rate at which any one reaction can happen is not simply the defined constant. Where previously the reaction *ab_c* was defined as $\mathrm{A} + \mathrm{B} \rightarrow \mathrm{C}, \alpha$, we take the constant α and multiply it by the number of molecules in both the A and B components (to allow for permutations of all A molecules interacting with all B molecules) to give a reaction rate of $\alpha\mathrm{AB}$ which is known as the mass action rate. The reactions can also take the form of ODEs (as seen below) by a linear transform on the reaction definitions.

$$
\begin{aligned}
\frac{\mathrm{dA}}{\mathrm{d}t} &= -\alpha\mathrm{A}(t)\mathrm{B}(t) + (\beta + \gamma)\mathrm{C}(t) & \qquad \frac{\mathrm{dC}}{\mathrm{d}t} &= \alpha\mathrm{A}(t)\mathrm{B}(t) - (\beta + \gamma)\mathrm{C}(t)\\
\frac{\mathrm{dB}}{\mathrm{d}t} &= -\alpha\mathrm{A}(t)\mathrm{B}(t) + \beta\mathrm{C}(t) + \delta\mathrm{D}(t) & \qquad \frac{\mathrm{dD}}{\mathrm{d}t} &= \gamma\mathrm{C}(t) - \delta\mathrm{D}(t)
\end{aligned}
$$

Through these two formats, both stochastic simulation and ODE analysis are available. ODEs derived from PEPA in this manner will always respect the rules of conservation, as PEPA works on a static number of components. The inclusion of stoichiometric information (the quantitative relationship between reactants and products) outside of the PEPA model does however allow for a more powerful representation. Now the numbers of each components required in each reaction are any valid integer i.e. *ab_c* requires 3 units of component A instead of 1.

3 Modelling a BitTorrent Network

Before a BitTorrent network can be modelled, the salient parts of the protocol must be covered. Prior work by Qui and Srikant shall be examined and contrasted against the intended model before a more in-depth description of the PEPA model.

3.1 The BitTorrent Protocol

Developed in 2001 by Bram Cohen, BitTorrent was designed as a way of distributing the load of hosting a resource, making use of the bandwidth of the users. The BitTorrent protocol encodes information regarding the resource within a torrent file, including SHA-1 hashing of the files and the location of the tracker, where peer discovery occurs. When starting a torrent, the client (at this initial point known as a downloader) must contact the tracker in order to join the 'swarm' of active peers. Each contact with the tracker will typically return a randomised peer list of size fifty (where swarm size > 50) and so over time (contact occurring every three to thirty minutes) will represent a well-connected graph.

Once knowledge of other peers is obtained, connections can be made and transfer started. The entire content is split into a number of pieces (one Linux distribution supplies a 2.83GB DVD over 2906 pieces) and the parallelism is in the ability to download these pieces in any order. Using a combination of tit-for-tat, and a set of behaviours for dealing with previously snubbed peers (peers you are currently ignoring), each client attempts to maximise its own downloading speed by uploading to those peers that offer the highest transfer speeds. After each piece is downloaded, it can be offered to other peers instantly.

The splitting of the content into multiple pieces also allows the downloading to happen over greater periods of time. Izal *et al.* had to account for multi-session downloading (where peers disconnect from the swarm and reconnect at a later point, ready to continue where they left off), a feature that is advantageous when dealing with large downloads i.e. 2.83GB operating systems.

The protocol does not enforce a system for peers that are only uploading (known as seeds). The current implementation of BitTorrent by Bram Cohen [7] uses a seeding policy based on uploading to those peers that can download the fastest, the motivation being to create another seed as quickly as possible. As has been noted, this current policy means there is little incentive [8,9] for a client to upload once the entire content has been downloaded.

Lastly, while not part of the protocol, the recommended strategy for piece selection is *rarest-first*. With the exception of the initial piece, rarest-first strategy is used to ensure an even availability of all pieces. Although this only applies within the local group (each peer cannot see availability of pieces beyond those it is connected to) the well-connected property of random graphs will help create an even spread of pieces over time.

3.2 Simple Fluid Model

The fluid model by Qiu and Srikant [10] adopts a high level view of a BitTorrent network, representing peers in one of two states, downloaders or seeders. Consisting of two differential equations (downloaders (1) and seeders (2)), six parameters are used to define the behaviour.

$$\frac{\mathrm{d}x}{\mathrm{d}t} = \lambda - \theta x(t) - min\{cx(t), \mu(\eta x(t) + y(t))\} \ . \tag{1}$$

$$\frac{\mathrm{d}y}{\mathrm{d}t} = min\{cx(t), \mu(\eta x(t) + y(t))\} - \gamma y(t) \ . \tag{2}$$

The main fragment (3), present in both reactions, defines the rate of completion where variable c is the downloading bandwidth, $x(t)$ the number of downloaders at time t, μ the uploading bandwidth, $\eta \in [0, 1]$ the effectiveness of file-sharing and $y(t)$ the number of seeds at time t. This fragment simply states that the rate peers can complete the download is defined by either the speed the peers can download at or the speed at which all peers are uploading, whichever is smaller. The other variables are γ for seed disconnect rate, λ for peer arrival rate and θ for termination of downloading.

$$\min\{cx(t), \mu(\eta x(t) + y(t))\} \ . \tag{3}$$

In the paper, η is defined as $\eta \approx 1 - \left(\frac{\log N}{N}\right)^k$ where N is the number of pieces and k is the number of other downloaders currently connected to. If we let $N = 2906$, as seen for the DVD example, even where $k = 1$ $\eta \approx 0.9988$. Additionally, the upload capacity of the typical consumer is smaller than the download capacity available, thus $\frac{c}{\mu} \geq 1$ and with typical asynchronous connections (2Mbps download, 512Kbps upload) $\frac{c}{\mu} \geq 4$. This can be used to approximate (3) to $\mu(x(t) + y(t))$ when $x(t)(\frac{c}{\mu} - 1) > y(t)$, a condition satisfied for the majority of the time in Qui and Srikant's own results. Unsurprisingly it can be seen from this approximation that the rate of change from downloader to seeder within the model will follow the exponential distribution with rate μ but more importantly, with $\sigma^2 = \frac{1}{\mu^2}$.

Instead, the PEPA model will compartmentalise the downloading action over one hundred steps, acting as a percentage complete indicator. To accurately model a BitTorrent network all permutations of the pieces should be recorded. As this would require 2^N states (and so intractable for any but the most trivial values of N) the approximation to one hundred states for percentage complete is deemed adequate. This action can be represented by the Erlang distribution (Erlang CDF F_e shown in (4)) and the effect of splitting the download into k compartments seen in Fig. 3.

$$F_e = 1 - e^{-\lambda x} \sum_{n=0}^{k-1} \frac{(\lambda x)^n}{n!} \qquad \begin{array}{l}\text{where } k = \text{number of compartments,} \\ \text{and } \lambda = \text{rate for each compartment} \ .\end{array} \tag{4}$$

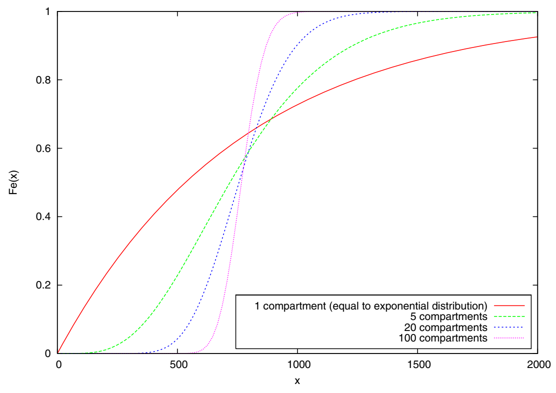


Fig. 3. CDF for the Erlang distribution. The effect of splitting a single state change ($\lambda = 0.0013$) to k compartments.

With the Erlang distribution, $\bar{x} = \frac{k}{\lambda}$ and $\sigma^2 = \frac{k}{\lambda^2}$, so if we define $\lambda' = k\lambda$ the means of the original distribution and the Erlang distribution are identical ($\frac{1}{\lambda} = \frac{k}{\lambda'}$) but the variance is reduced by a factor of k ($\frac{1}{\lambda^2}$ compared to $\frac{k}{(\lambda')^2} = \frac{1}{k\lambda^2}$), an important consideration when dealing with events with such large delays.

While the fluid model defined by Qui and Srikant could undoubtedly be modified to follow the Erlang distribution, the advantage of explicitly defining the one hundred steps is to accommodate other observable peer behaviour. Besides allowing a peer to disconnect, the PEPA model will incorporate the concept of multi-session downloads [8], where a peer can temporarily disconnect from the network, before rejoining and continuing from where it left off.

3.3 The PEPA Model

For the PEPA reagent-centric model, the following assumptions are made:

1. The network shall consist of only homogeneous, asynchronous connections.
2. The source for the content (original seeder) will have a persistent online presence. The torrent in this instance cannot die (less than 100% of the content being available).
3. Through the use of randomised peer lists and tit-for-tat the upload of the peer can be fully utilised if there is demand.
4. Through the use of randomised peer lists, rarest first piece selection and assumption 2, all pieces are available to all peers.
5. Peer behaviour is independent of time.

Assumptions 1 and 5 are gross simplifications of the real system but allow for a tractable model. Modelling different types of connections can have a multiplicative effect on the number of states, as every action may be influenced by different types of peer. Peer behaviour is unlikely to be independent of time or even consistent across peers. Pouwelse *et al.* [9] found that new torrents were often accompanied by a huge surge of downloaders very early on, labelling this the *flashcrowd effect*.

Assumption 2 is justified by the expected use of the BitTorrent protocol. Where content providers previously relied on HTTP or FTP, the content was required to be permanently accessible. This would not change with the use of the BitTorrent protocol. The content owners may optimise such that with enough seeds the original is not active, but to maintain availability it would be required to seed again if the number of seeds dropped below some threshold.

Assumptions 3 and 4 are outwith the scope of this paper. They rely on the properties of random graphs, the small world assumption and epidemiology. Without these assumptions, modelling the BitTorrent protocol becomes increasingly more complex. Of these assumptions, only 2 is not also made by Qui and Srikant.

Using these assumptions, the model will incorporate certain behaviours, these being:

1. Each peer within the swarm shall be either a downloader or a seed.
2. A downloader will be split into one hundred states, to record the level of completion.
3. Both downloaders and seeds can quit or go offline at any stage, returning online will return them to their previous state.

These properties can be translated into four groups of definitions within the PEPA reagent-centric approach, with one group for each of the peer states (online, offline, downloading and seeding). Two additional components, upload_pool and peer_deac, are used to control behaviour of the model. The existence of the downloading state is to enforce correct behaviour regarding use of the available upload bandwidth.

$$
\begin{aligned}
\text{Online\_Peer-0}_{\text{H}} &\overset{def}{=} (connect_0, connect\_rate).\text{Online\_Peer-0}_{\text{L}} + \\
&\quad (offline_0, offline\_rate).\text{Online\_Peer-0}_{\text{L}} + \\
&\quad (quit_0, quit\_rate).\text{Online\_Peer-0}_{\text{L}} \\
\text{Online\_Peer-0}_{\text{L}} &\overset{def}{=} (torrent, torrent\_rate).\text{Online\_Peer-0}_{\text{H}} + \\
&\quad (online_0, online\_rate).\text{Online\_Peer-0}_{\text{H}} \\
\text{Online\_Peer-}n_{\text{H}} &\overset{def}{=} (connect_n, connect\_rate).\text{Online\_Peer-}n_{\text{L}} + \\
&\quad (offline_n, offline\_rate).\text{Online\_Peer-}n_{\text{L}} + \\
&\quad (quit_n, quit\_rate).\text{Online\_Peer-}n_{\text{L}} \\
\text{Online\_Peer-}n_{\text{L}} &\overset{def}{=} (downloaded_n, d\_rate).\text{Online\_Peer-}n_{\text{H}} + \\
&\quad (online_n, online\_rate).\text{Online\_Peer-}n_{\text{H}} \\
&\qquad n \in \{1 \ldots 99\}
\end{aligned}
$$

Here we have the online peer PEPA definitions. The activities $quit_n$ and $offline_n$ cause the levels of an online peer to decrease, seen as activities that change a resource from high to low. Conversely the activity $online_n$ increases the numbers of that particular peer. New peers enter the system at Online_Peer-0, through the *torrent* activity and enter the ready state for downloading via the $connect_n$ activity. As entering the downloading state is a change of state, the $connect_n$

activity reduces the number of online peers for that given state. Lastly, the act of completing another part of the download via the *downloaded*$_n$ activity increases the number of online peers for that percentage.

$$\begin{aligned}
\text{Downloading\_Peer-}n_\text{H} &\overset{def}{=} (\mathit{downloaded}_{n+1}, \mathit{d\_rate}).\text{Downloading\_Peer-}n_\text{L} + \\
&\quad (\mathit{downloading\_offline}_n, \mathit{offline\_rate}).\text{Downloading\_Peer-}n_\text{L} + \\
&\quad (\mathit{downloading\_quit}_n, \mathit{quit\_rate}).\text{Downloading\_Peer-}n_\text{L} \\
\text{Downloading\_Peer-}n_\text{L} &\overset{def}{=} (\mathit{connect}_n, \mathit{connect\_rate}).\text{Downloading\_Peer-}n_\text{H} \\
\text{Offline\_Peer-}n_\text{H} &\overset{def}{=} (\mathit{online}_n, \mathit{online\_rate}).\text{Offline\_Peer-}n_\text{L} \\
\text{Offline\_Peer-}n_\text{L} &\overset{def}{=} (\mathit{offline}_n, \mathit{offline\_rate}).\text{Offline\_Peer-}n_\text{H} + \\
&\quad (\mathit{downloading\_offline}_n, \mathit{offline\_rate}).\text{Offline\_Peer-}n_\text{H} \\
&\qquad n \in \{0 \ldots 99\}
\end{aligned}$$

Next are the PEPA definitions for the downloading and offline states. The first defined activity for a Downloading_Peer$_n$ is *downloaded*$_{n+1}$. Here we can see how progression through the states happens. The number of downloading peers with $x\%$ complete decreases when they manage to complete another percent, taking them to $x+1\%$. As already seen in the online peer definition, the number of peers with $x\%$ complete increases through the *downloaded*$_x$ activity, and so already some of the behaviour of the model can be seen.

The downloading peer state like the online peer can quit or go offline. The *d_rate* represents the length of time to download 1% of the torrent and can easily be several minutes in duration, and so this is the state where peers will spend the most time. Without the ability to terminate in this state the rates for quitting and going offline are skewed as $x(t) = \sum_i \text{Online\_Peer}_\text{i} + \text{Downloading\_Peer}_\text{i}$.

$$\begin{aligned}
\text{Seed}_\text{H} &\overset{def}{=} (\mathit{seed\_quit}, \mathit{seed\_quit\_rate}).\text{Seed}_\text{L} + (\mathit{seed\_offline}, \mathit{seed\_offline\_rate}).\text{Seed}_\text{L} \\
\text{Seed}_\text{L} &\overset{def}{=} (\mathit{downloaded}_{100}, \mathit{downloaded\_rate}).\text{Seed}_\text{H} + \\
&\quad (\mathit{seed\_online}, \mathit{seed\_online\_rate}).\text{Seed}_\text{H}
\end{aligned}$$

$$\begin{aligned}
\text{Offline\_Seed}_\text{H} &\overset{def}{=} (\mathit{seed\_online}, \mathit{seed\_online\_rate}).\text{Offline\_Seed}_\text{L} \\
\text{Offline\_Seed}_\text{H} &\overset{def}{=} (\mathit{seed\_offline}, \mathit{seed\_offline\_rate}).\text{Offline\_Seed}_\text{H}
\end{aligned}$$

As can be seen, the seed plays a passive role, where simply their presence (or lack of) is the only information required.

The control states Upload_Pool and Peer_deac are designed to maintain the upload capacity within the network. Upload_Pool acts as a counter for unused upload bandwidth while Peer_deac was created to prevent skew to the rates at which seeds and peers would quit or go offline. For each seed or online peer that disconnects from the swarm, the upload pool will be decremented. The rate at which seeds and peers disconnect though should not depend on the current level of the upload pool while conservation tells us we can not have populations of negative values. Instead the Peer_deac acts as a reservoir, holding all the disconnect requests and reducing the upload pool as and when possible. This will become clearer when the PEPA is converted.

$$\begin{aligned}
\text{Upload\_Pool}_{\text{H}} &\overset{def}{=} (\mathit{connect}_n, \mathit{connect\_rate}).\text{Upload\_Pool}_{\text{L}} + \\
&\quad (\mathit{deallocation}, \mathit{deallocation\_rate}).\text{Upload\_Pool}_{\text{L}} \\
\text{Upload\_Pool}_{\text{L}} &\overset{def}{=} (\mathit{downloaded}_p, \mathit{d\_rate}).\text{Uploaded\_Pool}_{\text{H}} + \\
&\quad (\mathit{online}_n, \mathit{online\_rate}).\text{Upload\_Pool}_{\text{H}} + \\
&\quad (\mathit{downloading\_offline}_n, \mathit{offline\_rate}).\text{Uploaded\_Pool}_{\text{H}} \\
&\quad (\mathit{downloading\_quit}_n, \mathit{quit\_rate}).\text{Uploaded\_Pool}_{\text{H}} \\
&\quad (\mathit{seed\_online}, \mathit{seed\_online\_rate}).\text{Upload\_Pool}_{\text{H}}
\end{aligned}$$

$$n \in \{0 \ldots 99\}, p \in \{1 \ldots 100\}$$

$$\begin{aligned}
\text{Peer\_deac}_{\text{H}} &\overset{def}{=} (\mathit{deallocation}, \mathit{deallocation\_rate}).\text{Peer\_deac}_{\text{L}} \\
\text{Peer\_deac}_{\text{L}} &\overset{def}{=} (\mathit{offline}_n, \mathit{offline\_rate}).\text{Peer\_deac}_{\text{H}} + \\
&\quad (\mathit{quit}_n, \mathit{quit\_rate}).\text{Peer\_deac}_{\text{H}} + \\
&\quad (\mathit{seed\_offline}, \mathit{seed\_offline\_rate}).\text{Peer\_deac}_{\text{H}} + \\
&\quad (\mathit{seed\_quit}, \mathit{seed\_quit\_rate}).\text{Peer\_deac}_{\text{H}}
\end{aligned}$$

$$n \in \{0 \ldots 99\}$$

The PEPA model defined allows the following behaviour. Peers join the swarm with 0% of the content complete and without the ability to upload. They can enter a state ready to download (resources committed) and increase the percentage complete at which point the allocated resources are released. Peers with 1 to 100% complete contribute to the upload capacity. All peers (online, downloading and seeds) can enter an offline state or quit where those that had contributed to the upload (1 to 100%) reduce the upload capacity.

3.4 Reaction Based System

With BitTorrent modelled in the reagent-centric style, it can be translated into a set of reactions suitable for either stochastic simulation or analysis through ODEs. Based on the PEPA definitions, the $\mathit{connect}_n$ activity allocates the required upload bandwidth and changes a online peer to a downloading peer. As previously stated, the use of stoichiometric information is external to the PEPA model, but can be clearly seen here.

$$\mathit{connect}_n,\ \text{Online\_Peer}_{\text{n}} + \text{Upload\_Pool} \times 4 \rightarrow \text{Downloading\_Peer}_{\text{n}},\ \mathit{connect\_rate}$$
$$n \in \{0 \ldots 99\}$$

Here the use of the Upload_Pool can be clearly seen. Once a peer has part of the content, they return the previously allocated resources, plus an additional unit that they now contribute to the swarm. Once the last part of the content is downloaded a downloading peer can be seen to transition to a seed

$$\begin{aligned}
&\mathit{downloaded}_1,\ \text{Downloading\_Peer}_0 \rightarrow \text{Online\_Peer}_1 + \\
&\qquad \text{Upload\_Pool} \times 5,\ \mathit{downloaded\_rate} \\
&\mathit{downloaded}_n,\ \text{Downloading\_Peer}_{\text{n-1}} \rightarrow \text{Online\_Peer}_{\text{n}} + \\
&\qquad \text{Upload\_Pool} \times 4,\ \mathit{downloaded\_rate} \\
&\mathit{downloaded}_{100},\ \text{Downloading\_Peer}_{99} \rightarrow \text{Seed} + \text{Upload\_Pool} \times 4,\ \mathit{download\_rate}
\end{aligned}$$
$$n \in \{2 \ldots 99\}$$

Within the *offline* activity, the influence of the Peer_deac is evident. Here, and in the definitions for seeds going offline, the delay in updating the upload pool allows the activity to happen at the correct rate. At a later point the *deallocation* activity will decrement the upload pool as required.

$\mathit{offline}_0$, Online_Peer$_0$→Offline_Peer$_0$+Peer_deac, *offline_rate*
$\mathit{offline}_n$, Online_Peer$_n$→Offline_Peer$_n$+Peer_deac, *offline_rate*
$n \in \{1 \ldots 99\}$

deallocation, Upload_Pool+Peer_deac→ , *deallocation_rate*

The behaviour of a downloading peer that goes offline or quits is different to that of an online peer. The reason is the use of Peer_deac is not required. The previous securing of resources accounts for more than any one peer contributes to the system. Thus a downloading peer only has to relinquish one unit less than it reserved. This allows the model to behave correctly without additional use (and minor delays) of the deallocation counter.

$\mathit{downloading\_offline}_0$, Downloading_Peer$_0$→Offline_Peer$_0$+Upload_Pool×4, *offline_rate*
$\mathit{downloading\_offline}_n$, Downloading_Peer$_n$→Offline_Peer$_0$+Upload_Pool×3, *offline_rate*
$n \in \{1 \ldots 99\}$

The online_n definitions allow peers that went offline to return to the swarm. As can be seen, if returning from a partial download state their return increments the upload capacity within the network.

online_0, Offline_Peer$_0$→Online_Peer$_0$, *online_rate*
online_n, Offline_Peer$_n$→Online_Peer$_n$+Upload_Pool, *online_rate*
$n \in \{1 \ldots 99\}$

Similar rules exist for the seeds. The only difference lies within the rate at which seeds can leave the system. To enforce the second assumption of the BitTorrent network, the rate is changed from *seed_online_rate*×Seed to *seed_online_rate*×(Seed-1) and so enforces the continued existence of one seed at all times.

4 Analysis

Of the three experiments perform by Qui and Srikant, the third compared the fluid model against log files from a real torrent file (the file in question was 530MB in size). Using these logs, they derived values for the six parameters with λ and γ found to vary with time. Simplified slightly, the parameters used in the comparison were $\eta = 1$, $\theta = 0.001$, $\mu = 0.0013$ and $c = 1$. As already stated λ and γ varied with time and so were set at $\lambda = 0.06$ and $\gamma = 0.001$ for $t \leq 800$, and $\lambda = 0.03$ and $\gamma = 0.0044$ for $t > 800$.

For the PEPA model to mirror the fluid model, the ability of the downloaders and seeds to switch between online and offline was turned off. To approximate the time dependent arrival rate, a linear function was fitted to the data and calculated with *torrent_rate* = 0.0368577. As the dynamic seed quit rate (γ) is connected to seed population it can not be fitted in the same way and hence fixed at 0.0044. Lastly, the download rate was normalised to account for the content being downloaded in 1% chunks and that enough upload capacity was allocated to maximise each peer, leaving the $downloaded\_rate = 0.52$.

The results for these two models can be seen in Fig. 4. The peaks connected to the fluid model are the effect of the dynamic parameters. The effect of cascade of exponential distributions can be easily seen in the delay before a new seed can appear within the PEPA model. Whilst the steady state value for the seeds is similar between the two models, the value for the downloaders is noticeably different, a potential cause for concern given the low values encountered. Figure 5 compares the same PEPA model against a fluid model with non dynamical parameters, λ and γ being fixed to those values used within the PEPA model. From Fig. 4 a difference in the number of downloaders within the model still exists but is now reduced, which can be explained by fixing the rate at which new peers enter the system. When using fixed parameters, the steady state values for downloaders and seeds are reasonably close between the two models.

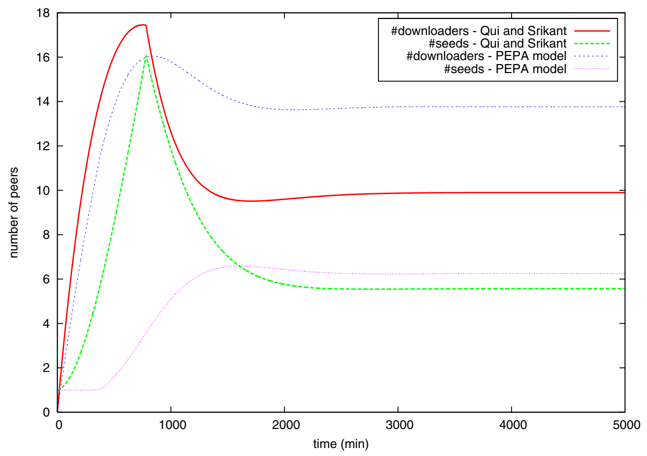


Fig. 4. Comparison of fluid and PEPA model (fluid model using time varying parameters)

To further contrast the two models, a second experiment was run, where the size of the file was increased. The size of the file was set to 2.83GB (the DVD previously mentioned) and the peers assumed to have a 2Mbps download capacity and 256Kbps upload. This capacity equates to $\mu = \frac{1}{1648}$ and $downloaded\_rate = \frac{1}{4}$, while leaving c alone does not affect the model. All other parameters were left at their original (fixed)values. Figure 6 shows the results for this. This experiment accentuates the differences seen within the first experiment. While the number of seeds in both models is approximately the same, the difference in the number of downloaders has increased.

The final graph, Fig. 7 highlights the flexibility built into both the use of PEPA as the modelling language and of this particular model. Gillespie's Stochastic

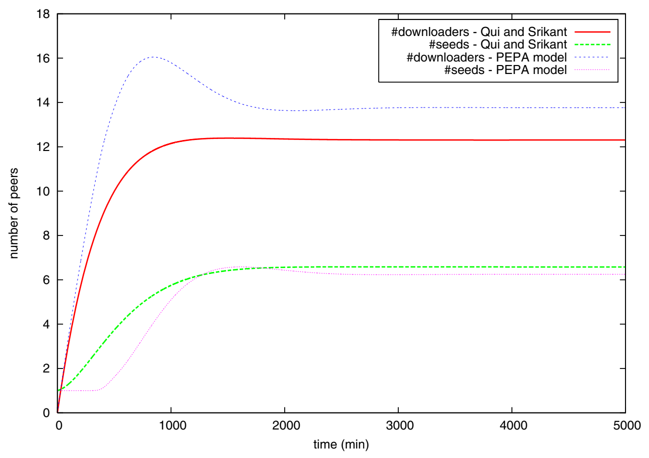


Fig. 5. Comparison of fluid and PEPA model (fluid model using fixed parameters)

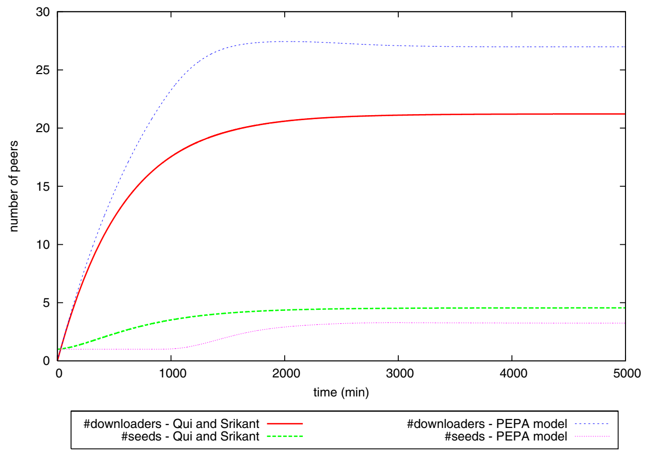


Fig. 6. PEPA and fluid model with larger files

Simulation Algorithm (SSA) was run over five independent replications as an alternative to use of ODEs. Where stochastic noise may be of interest, or the population sizes of the components are low, the use of stochastic simulations may prove advantageous. For the comparisons against the fluid model, the ability to change from online to offline was inhibited. Here, with exaggerated parameters, the effects of the offline state can be seen (more so with the downloaders). By shifting a certain percentage of the downloaders from the active state, the available bandwidth is reduced causing a more shallow gradient towards the steady state level.

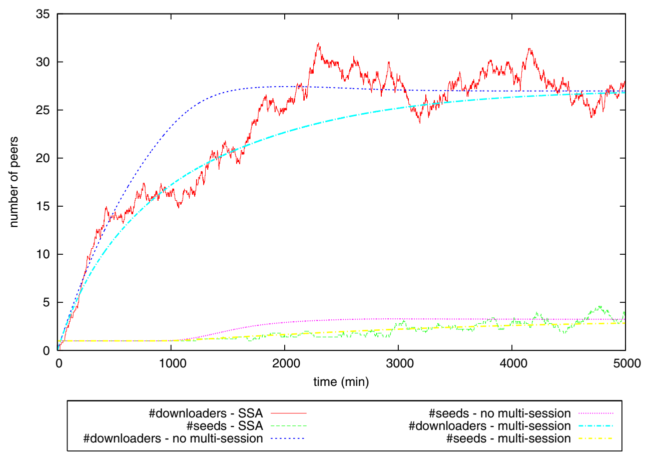


Fig. 7. PEPA model with and without multi-session ability and stochastic simulation

5 Conclusions

The ability to convert PEPA models to a reaction based system, allowing the use of ODE analysis or stochastic simulation now allows models previously too large for more traditional forms of analysis. Using the BitTorrent protocol to highlight this fact, a model where a peer can exist in any one of three hundred states was created and compared against a simple two state fluid model. Under CTMCs this would have created a potential state space of 300^x where x is the number of peers. The PEPA reagent-centric approach also affords a cleaner representation whilst providing access to both deterministic and stochastic solutions. As the size of the model increases, the complexity of the ODEs can hinder understanding of the system or alteration if required.

Whilst the translation process is now well defined, the mapping of CTMC analysis to time-series analysis is still work in progress. Regardless, the ability to convert to a reaction based system is a useful tool.

The BitTorrent model itself has shown that not only can a large system be defined within the PEPA language, but also offered a more detailed view of a BitTorrent network. It expanded on the simple fluid model, detailing many of the extra states. Many of the parameters require assigning based on real world data, the next step being to obtain records from a tracker. It should also be noted that the full flexibility of the model has not been covered. While one of the assumptions was that peer behaviour was independent of time, the alteration of rates for different levels of completion would allow a certain level of controllable behaviour, i.e. a peer is more likely to disconnect in the first five percent and extremely unlikely in the last ten percent. Again, this data can be gathered by obtaining records from a tracker.

References

1. Hillston, J.: A Compositional Approach to Performance Modelling. Cambridge University Press (1996)
2. Regev, A., Silverman, W., Shapiro, E.: Representation and simulation of biochemical processes using the pi-calculus process algebra. In: Proceedings of the Pacific Symposium of Biocomputing (PSB2001). (2001) 459–470
3. Priami, C., Regev, A., Silverman, W., Shapiro, E.: Application of a stochastic name passing calculus to representation and simulation of molecular processes. Information Processing Letters **80** (2001) 25–31
4. Gillespie, D.: Exact stochastic simulation of coupled chemical reactions. Journal of Physical Chemistry **81**(25) (1977) 2340–2361
5. Calder, M., Gilmore, S., Hillston, J.: Modelling the influence of RKIP on the ERK signaling pathway using the stochastic process algebra PEPA. In: Proceedings of BioConcur'04, London, England (2004) Extended version to appear in Transactions on Computational Systems Biology.
6. Calder, M., Gilmore, S., Hillston, J.: Automatically deriving ODEs from process algebra models of signalling pathways. In: Proceedings of Computational Methods in Systems Biology '05, Edinburgh, Scotland (2005)
7. Cohen, B.: Incentives Build Robustness in BitTorrent. `http://www.bittorrent.com/bittorrentecon.pdf` (2003)
8. Izal, M., Urvoy-Keller, G., Biersack, E.W., Felber, P., Hamra, A.A., Garcés-Erice, L.: Dissecting BitTorrent: Five Months in a Torrent's Lifetime. In Barakat, C., Pratt, I., eds.: PAM. Volume 3015 of Lecture Notes in Computer Science., Springer (2004) 1–11 `http://www.pam2004.org/papers/148.pdf`.
9. Pouwelse, J., Garbacki, P., Epema, D., Sips, H.: The BitTorrent P2P File-sharing System: Measurements and Analysis. In Castro, M., van Renesse, R., eds.: IPTPS. Volume 3640 of Lecture Notes in Computer Science., Springer (2005) `http://www.st.ewi.tudelft.nl/~pouwelse/Bittorrent_Measurements_6pages.pdf`.
10. Qiu, D., Srikant, R.: Modeling and performance analysis of BitTorrent-like peer-to-peer networks. In Yavatkar, R., Zegura, E.W., Rexford, J., eds.: SIGCOMM, ACM (2004) 367–378 `http://comm.csl.uiuc.edu/srikant/Papers/sigcomm04_revised.pdf`.

Temporal Logic Verification Using Simulation

Georgios E. Fainekos[1], Antoine Girard[2], and George J. Pappas[3]

[1] Department of Computer and Information Science, Univ. of Pennsylvania, USA
fainekos@cis.upenn.edu
[2] VERIMAG, 2 avenue de Vignate, 38610 Gières, France
Antoine.Girard@imag.fr
[3] Department of Electrical and Systems Engineering, Univ. of Pennsylvania, USA
pappasg@ee.upenn.edu

Abstract. In this paper, we consider a novel approach to the temporal logic verification problem of continuous dynamical systems. Our methodology has the distinctive feature that enables the verification of the temporal properties of a continuous system by verifying only a finite number of its (simulated) trajectories. The proposed framework comprises two main ideas. First, we take advantage of the fact that in metric spaces we can quantify how close are two different states. Based on that, we define robust, multi-valued semantics for MTL (and LTL) formulas. These capture not only the usual Boolean satisfiability of the formula, but also topological information regarding the distance from unsatisfiability. Second, we use the recently developed notion of bisimulation functions to infer the behavior of a set of trajectories that lie in the neighborhood of the simulated one. If the latter set of trajectories is bounded by the tube of robustness, then we can infer that all the trajectories in the neighborhood of the simulated one satisfy the same temporal specification as the simulated trajectory. The interesting and promising feature of our approach is that the more robust the system is with respect to the temporal logic specification, the less is the number of simulations that are required in order to verify the system.

1 Introduction

Software and hardware design has tremendously benefited from advances in algorithmic verification. Model checking [1] is now a widely used technology in various industrial settings. Thanks to the rapidly growing area of embedded systems with real-time specifications, a similar growth is also being experienced in the area of real-time systems [2]. As the complexity of the physical systems increases and captures continuous or hybrid systems, the verification problems quickly become hard, if not undecidable.

For the verification of hybrid systems, a variety of methods have been proposed [3,4,5,6,7,8] (not an inclusive list). The common characteristic of all these approaches is that they apply to either continuous systems with simple dynamics, or they are computationally expensive and, thus, they can only be used for low dimensional systems (for promising high-dimensional results see [9,10]). Beyond the scope of these techniques, the analysis of complex systems still relies heavily on simulation-based methods for monitoring [11]. Along these lines several authors have proposed simulation techniques that can provide guarantees for uniform coverage [12,13] or even completeness results [14].

This paper develops a simulation-based method for verifying temporal properties of complex continuous systems. In particular, given a continuous dynamical system, a set of initial conditions, a bounded time horizon, and a temporal logic specification expressed in Metric or Linear Temporal Logic [15], we develop a simulation-based algorithm that verifies whether all the system trajectories satisfy the desired temporal property. To achieve this, we build upon two recent notions : a definition of *robust satisfaction* for Metric Temporal Logic (MTL) specifications [16] and the notion of *bisimulation functions* [17]. The definition of robust satisfaction of an MTL specification is meaningful only when state sequences evolve in metric spaces, a very natural assumption for continuous systems. Our proposed robust semantics capture bounds on the magnitude of the state perturbations that can be tolerated without altering the Boolean truth value of the MTL or LTL property. Bisimulation functions, on the other hand, quantify the distance between two approximately bisimilar states and the trajectories initiating from them. Using a bisimulation function we can define a neighborhood of trajectories around a nominal one which have approximately the same behavior as the nominal trajectory. If this neighborhood of the simulated trajectory is contained in the tube of trajectories, which robustly satisfy the specification, then we can safely infer that the neighborhood of trajectories also satisfies the specification.

Based on this observation, we develop an algorithm that, first, samples points in the set of initial conditions of the system using guidance from the bisimulation function. Starting from this set of points, we simulate the system for a bounded horizon. For each of these trajectories we compute an under-approximation of its robustness degree. If the robustness degree bounds the distance computed by the bisimulation function then we are done, otherwise we repeat the procedure. The novelty in our framework is that the number of simulations, which are required for the verification of the system, decreases inversely to the robustness of the system with respect to the temporal property.

Finally, we would like to point out that in the past several authors have also studied the robustness of real time specifications with respect to timed or dense time traces of real time systems [18,19,20], but the robustness is considered with respect to the timing constraints, not state perturbations. The work which is the closest in spirit to this paper appears in [21] where the authors give quantitative semantics to the branching-time logic CTL (called Discounted CTL) in order to achieve robustness with respect to model perturbations.

2 Problem Formulation

Let $\mathbb{R}$ be the set of the real numbers, $\mathbb{Q}$ the set of the rational numbers and $\mathbb{N}$ the set of the natural numbers. We denote the extended real number line by

$\overline{\mathbb{R}} = \mathbb{R} \cup \{\pm\infty\}$. In addition, $\mathbb{R}_{\geq 0}$ denotes the subset of the reals whose elements are greater or equal to zero. Finally, $\mathcal{P}(C)$ denotes the powerset of a set C.

2.1 Continuous Time Dynamical Systems as Timed State Sequences

In this paper, we focus on the verification of temporal properties of continuous time dynamical systems.

Definition 1 (Continuous Time Dynamical System). *A continuous-time dynamical system is defined by a tuple $\Sigma = (N, P, f, g, I, AP, \mathcal{O})$ where: N and P are positive integers which respectively denote the dimension of the state-space and of the observation-space, $f : \mathbb{R}^N \to \mathbb{R}^N$ and $g : \mathbb{R}^N \to \mathbb{R}^P$ are continuous maps, I is a compact subset of $\mathbb{R}^N$ which denotes the set of initial states, AP is a set of atomic propositions and $\mathcal{O} : AP \to \mathcal{P}(\mathbb{R}^P)$ is a predicate mapping.*

A *trajectory* of the continuous-time dynamical system Σ is a pair of functions $(x(t), y(t))$ such that $x : \mathbb{R}_{\geq 0} \to \mathbb{R}^N$ and $y : \mathbb{R}_{\geq 0} \to \mathbb{R}^P$ satisfy $x(0) \in I$ and $\forall t \in \mathbb{R}_{\geq 0}$

$$\dot{x}(t) = f(x(t)) \text{ and } y(t) = g(x(t)) \tag{1}$$

Here, $\dot{x}$ denotes the first order derivative of the function x. We make the standard assumption on f so that for a given initial state $x_0 \in I$, the system Σ has a unique trajectory. Hence, given $x(0)$ the system is deterministic. In order to specify the properties of interest for the system Σ in any temporal logic, we must define a set of regions in its observation space. If AP is a finite set of atomic propositions, then the *predicate mapping* $\mathcal{O} : AP \to \mathcal{P}(\mathbb{R}^P)$ is a set valued function that assigns to each atomic proposition $\pi \in AP$ a set of states $\mathcal{O}(\pi) \subseteq \mathbb{R}^P$.

Beyond certain classes of continuous dynamical systems such as linear systems, system Σ does not always have an analytical solution. Typically, the behavior of such systems can be explored using numerical simulation [22]. Numerical simulation methods approximate the differential equations of the system Σ by algebraic equations which depend on the size of the integration (time) step. Furthermore, such simulations can only be of finite duration. Therefore, we can model such computations by finite timed state sequences.

Definition 2 (TSS). *A timed state sequence $\mathcal{T}$ in a space Q is a tuple $(\sigma, \tau, \mathcal{O})$ where for some $n \in \mathbb{N}$: $\sigma = \sigma_0, \sigma_1, \ldots, \sigma_n$ is a sequence of states, $\tau = \tau_0, \tau_1, \ldots, \tau_n$ is a sequence of time stamps and $\mathcal{O} : AP \to \mathcal{P}(Q)$ is a predicate mapping. The following conditions must be satisfied by $\mathcal{T}$: (i) for all $i \in \{0, 1, \ldots, n\}$ we have $\sigma_i \in Q$ and $\tau_i \in \mathbb{R}_{\geq 0}$ and (ii) τ is a strictly monotonically increasing sequence.*

By convention, we set $\tau_0 = 0$ (in the Metric Temporal Logic we care only about relative time). We define $\sigma\uparrow_i$ to be the suffix of a sequence, i.e. $\sigma\uparrow_i = \sigma_i, \sigma_{i+1}, \ldots, \sigma_n$. When the same operator $\uparrow_i$ is applied to the sequence τ, it is defined as $\tau\uparrow_i = 0, \tau_{i+1} - \tau_i, \ldots, \tau_n - \tau_i$. The length of $\sigma = \sigma_0, \sigma_1, \ldots, \sigma_n$ is defined to be $|\sigma| = n + 1$. For convenience, we let $|\mathcal{T}| = |\tau| = |\sigma|$ and $\mathcal{T}\uparrow_i = (\sigma\uparrow_i, \tau\uparrow_i, \mathcal{O})$.

We define TS to be the set of all possible finite timed state sequences in the space $\mathbb{R}^P$, that is $\text{TS} = \{(\sigma, \tau, \mathcal{O}) \mid n \in \mathbb{N}_{>0}, \sigma \in (\mathbb{R}^P)^n, \tau \in \mathbb{R}^n_{\geq 0}$ such that $\tau_i < \tau_{i+1}$ for $i < n$ and $\mathcal{O} : AP \to \mathcal{P}(\mathbb{R}^P)\}$. Note that by definition we do not consider empty timed state sequences and that essentially the sequence σ is isomorphic to a point in the product space $(\mathbb{R}^P)^{|\sigma|}$. In addition, given a timed state sequence $\mathcal{T} = (\sigma, \tau, \mathcal{O})$, then $\text{TS}_\mathcal{T}$ is the set of all timed state sequences with the same predicate mapping $\mathcal{O}$ and the same sequence of time stamps τ as $\mathcal{T}$, i.e. $\text{TS}_\mathcal{T} = \{(\sigma', \tau', \mathcal{O}') \in \text{TS} \mid \tau' = \tau, \mathcal{O}' = \mathcal{O}\}$.

Now, given a sequence of integration steps (which is equivalent to a sequence of time stamps τ) for the numerical simulation of the system Σ, we can model the resulting discrete trajectory as a timed state sequence, which we refer to as a trace.

Definition 3 (Trace). *Given a sequence of time stamps τ, a trace of a continuous dynamical system Σ is a timed state sequence $\mathcal{T} = (\sigma, \tau, \mathcal{O})$ such that there exists a trajectory (x, y) of Σ satisfying $\sigma_i = y(\tau_i) = g(x(\tau_i))$ for all $i = 0, 1, \ldots, |\tau| - 1$. The set of traces of Σ associated with the sequence of time stamps τ is denoted by $\mathcal{L}_\tau(\Sigma)$.*

We should point out that in this paper, we essentially consider the trace to be sampled from the continuous solution of the system Σ. In numerical methods for the integration of differential equations, though, there exists a quantifiable and bounded error between the continuous solution of the equations (1) and the result of the numerical simulation, which can be driven arbitrarily close to zero [22]. Therefore, we can safely ignore this issue for now in order to facilitate the presentation of the contributions in the current paper.

2.2 Metric Temporal Logic over Finite Timed State Sequences

We employ the Metric Temporal Logic (MTL) [15] in order to formally characterize the desired behavior of the system Σ. In MTL, the syntax of the logic is extended to include timing constraints on the usual temporal operators of the Linear Temporal Logic (LTL). Using LTL specifications we can check qualitative timing properties, while with MTL specifications quantitative timing properties. Recently, it was shown by Ouaknine and Worrell [23] that MTL is decidable over finite timed state sequences. In this section, we review the basics of MTL with point-based semantics over finite timed state sequences.

Definition 4 (Syntax of MTL). *An MTL formula ϕ is inductively defined according to the grammar*

$$\phi ::= \top \mid \pi \mid \neg\phi_1 \mid \phi_1 \vee \phi_2 \mid \phi_1 \mathcal{U}_\mathcal{I} \phi_2$$

where $\pi \in AP$, $\top$ is the symbol for the boolean constant true and $\mathcal{I}$ is an interval of $\mathbb{R}_{\geq 0}$ with rational endpoints.

The set of all well formed MTL formulas is denoted by $\Phi_{\rm MTL}$. Even though we can derive the constant true ($\top$) from the law of excluded middle ($\top = \pi \vee \neg\pi$), we chose to add it in the syntax of MTL for reasons that will be clear in Sect. 3. The constant *false* is denoted by $\bot = \neg\top$. We can also derive additional temporal operators such as *release* $\phi_1 \,\mathcal{R}_{\mathcal{I}}\phi_2 = \neg((\neg\phi_1)\,\mathcal{U}_{\mathcal{I}}\neg\phi_2)$ (which is the dual of the until operator), *eventually* $\Diamond_{\mathcal{I}}\phi = \top\,\mathcal{U}_{\mathcal{I}}\phi$ and *always* $\Box_{\mathcal{I}}\phi = \bot\,\mathcal{R}_{\mathcal{I}}\phi$.

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, or even a singleton. For any $t \in \mathbb{Q}$, we define $\mathcal{I} + t = \{t' + t \mid t' \in \mathcal{I}\}$. In the case where $\mathcal{I} = [0, +\infty)$, we remove the subscript $\mathcal{I}$ from the temporal operators, i.e. we just write $\mathcal{U}$, $\mathcal{R}$, $\Diamond$ and $\Box$. When all the subscripts of the temporal operators are of the form $[0, +\infty)$, then the MTL formula ϕ reduces to an LTL formula and we can ignore the time stamps.

Metric Temporal Logic (MTL) formulas are interpreted over timed state sequences $\mathcal{T}$ with $|\mathcal{T}| > 0$. In this paper, we denote formula satisfiability using a membership function $\langle\!\langle\phi\rangle\!\rangle : \mathrm{TS} \to \{\bot, \top\}$ instead of the usual notation $\mathcal{T} \models \phi$. We say that a timed state sequence $\mathcal{T}$ satisfies the formula ϕ when $\langle\!\langle\phi\rangle\!\rangle(\mathcal{T}) = \top$. In this case, $\mathcal{T}$ is a *model* of ϕ. The set of all models of ϕ is denoted by $\mathcal{L}(\phi)$, i.e. $\mathcal{L}(\phi) = \{\mathcal{T} \in \mathrm{TS} \mid \langle\!\langle\phi\rangle\!\rangle(\mathcal{T}) = \top\}$.

Definition 5 (Semantics of MTL). *Let $\mathcal{T} = (\sigma, \tau, \mathcal{O}) \in TS$, $\pi \in AP$, $i, j \in \mathbb{N}$ and $\psi, \phi_1, \phi_2 \in \Phi_{\rm MTL}$, then the semantics of any MTL formula ϕ are defined recursively as*

$$\begin{aligned}
\langle\!\langle\top\rangle\!\rangle(\mathcal{T}) &:= \top \\
\langle\!\langle\pi\rangle\!\rangle(\mathcal{T}) &:= \sigma_0 \in \mathcal{O}(\pi) \\
\langle\!\langle\neg\psi\rangle\!\rangle(\mathcal{T}) &:= \neg\langle\!\langle\psi\rangle\!\rangle(\mathcal{T}) \\
\langle\!\langle\phi_1 \vee \phi_2\rangle\!\rangle(\mathcal{T}) &:= \langle\!\langle\phi_1\rangle\!\rangle(\mathcal{T}) \vee \langle\!\langle\phi_2\rangle\!\rangle(\mathcal{T}) \\
\langle\!\langle\phi_1\,\mathcal{U}_{\mathcal{I}}\phi_2\rangle\!\rangle(\mathcal{T}) &:= \bigvee\nolimits_{i=0}^{|\mathcal{T}|-1}\big((\tau_i \in \mathcal{I}) \wedge \langle\!\langle\phi_2\rangle\!\rangle(\mathcal{T}{\uparrow}_i) \wedge \bigwedge\nolimits_{j=0}^{i-1}\langle\!\langle\phi_1\rangle\!\rangle(\mathcal{T}{\uparrow}_j)\big)
\end{aligned}$$

2.3 Problem Statement

Now that we have presented all the necessary mathematical objects we can formally state the verification problem that we answer in this paper.

Problem 6. Given an MTL formula ϕ, a continuous dynamical system Σ and a sequence of time stamps τ, verify that $\mathcal{L}_\tau(\Sigma) \subseteq \mathcal{L}(\phi)$. In other words, verify that all the traces $\mathcal{T}$ of Σ satisfy the specification ϕ.

The difficulty in solving Problem 6 is that in metric spaces there exists an infinite number of traces $\mathcal{T} = (\sigma, \tau, \mathcal{O})$ of Σ. Thus, the verification of Σ cannot be done by exhaustive simulation. In the following, we show that using the robust semantics of MTL (Sect. 3) and the notion of bisimulation function [17], the verification of Σ is possible by using only a finite number of simulations.

Example 7. In order to motivate the rest of the discussion, we present as an example the verification problem of a transmission line [10]. The goal is to check that the transient behavior of a long transmission line is acceptable both in terms of overshoot and of response time. Figure 1 shows a model of the transmission line, which consists of a number of RLC components (R: resistor, L: inductor and C: capacitor) modeling segments of the line. The left side is the sending end and the right side is the receiving end of the transmission line.

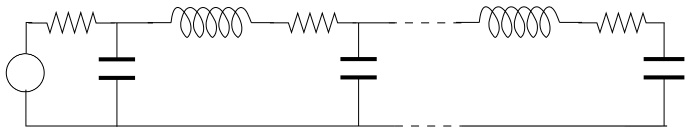


Fig. 1. RLC model of a transmission line

The dynamics of the system are given by the linear dynamical system

$$\dot{x}(t) = Ax(t) + bU_{in}(t) \text{ and } U_{out}(t) = Cx(t)$$

where $x(t) \in \mathbb{R}^N$ is the state vector containing the voltage of the capacitors and the current of the inductors and $U_{in}(t) \in \mathbb{R}$ is the voltage at the sending end. The output of the system is the voltage $U_{out}(t) \in \mathbb{R}$ at the receiving end. Here, A, b and c are matrices of appropriate dimensions. Initially, $U_{in}(0) \in [-0.2, 0.2]$ and the system is at its steady state $x(0) = -A^{-1}bU_{in}(0)$. Then, at time $t = 0$ the input is set to the value $U_{in}(t) = 1$. We use an 81st order RLC model of the transmission line (i.e. $N = 81$). An example of a trace is shown in Fig. 2.

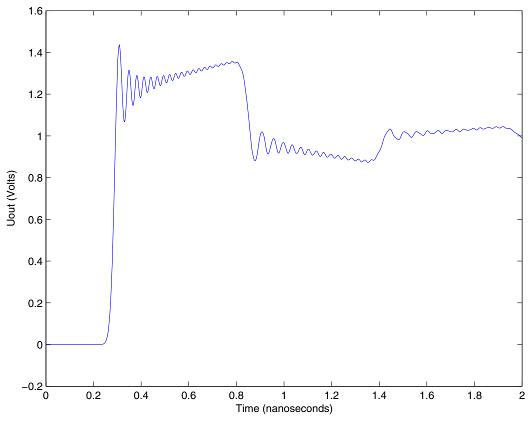


Fig. 2. An example trace of the RLC model of the transmission line

The goal of the verification is double. We want to check that the voltage at the receiving end stabilizes between 0.8 and 1.2 Volts within T nano-seconds (response time) and that its amplitude always remains bounded by θ Volts (overshoot) where $T \in [0, 2]$ and $\theta \geq 0$ are design parameters. The specification is expressed as the MTL property:

$$\phi = \Box\pi_1 \wedge \Diamond_{[0,T]}\Box\pi_2$$

where the predicates are mapped as follows: $\mathcal{O}(\pi_1) = [-\theta, \theta]$ and $\mathcal{O}(\pi_2) = [0.8, 1.2]$. We consider a time frame of 2 nanoseconds. The sequence of time stamps τ is uniformly generated with a time step of $\Delta t = 0.02$ nanoseconds.

3 Robust Satisfaction of MTL Specifications

In this section, we define what it means for a timed state sequence to satisfy a Metric Temporal Logic specification *robustly*. Our definition of robustness is built upon the fact that in metric spaces we can quantify how far apart are two points of the space.

3.1 Distance in Metric Spaces

Let (Q, d) be a metric space, that is a set Q with a metric d which gives the topology of Q. Given two points q_1, q_2 of Q, the number $d(q_1, q_2)$ is called the *distance* between q_1 and q_2 in the metric d. Using the metric d, we can define the distance of a point $q \in Q$ from a subset of $R \subseteq Q$.

Definition 8 (Distance, Signed Distance). *Let $q \in Q$ be a point, $R \subseteq Q$ be a set and d be a metric. Then we define the*

– distance *from q to R to be* $\mathbf{dist}_d(q, R) := \inf\{d(q, q') \mid q' \in R\}$
– signed distance *from q to R to be*

$$\mathbf{Dist}_d(q, R) := \begin{cases} -\mathbf{dist}_d(q, R) & \text{if } q \notin R \\ \mathbf{dist}_d(q, Q \backslash R) & \text{if } q \in R \end{cases}$$

We should point out that we use the extended definition of supremum and infimum, where $\sup \emptyset = -\infty$ and $\inf \emptyset = +\infty$. Also of importance is the notion of an open ball of radius ε centered at a point $q \in Q$.

Definition 9 (ε-Ball). *Given a metric d, a radius $\varepsilon \in \overline{\mathbb{R}}_{>0}$ and a point $q \in Q$, the open ε-ball centered at q is defined as $B_d(q, \varepsilon) = \{q' \in Q \mid d(q, q') < \varepsilon\}$.*

If the distance ($\mathbf{dist}_d$) of a point q from a set R is $\varepsilon > 0$, then $B_d(q, \varepsilon) \cap R = \emptyset$. And similarly, if $\mathbf{dist}_d(q, Q \backslash R) = \varepsilon > 0$, then $B_d(q, \varepsilon) \subseteq R$.

3.2 Defining Robust Semantics for the Metric Temporal Logic

In $\mathbb{R}^P$, we can quantify how close are two different observations $y_1, y_2 \in \mathbb{R}^P$ by using the metric $d(y_1, y_2) = \|y_1 - y_2\| = \sqrt{(y_1 - y_2)^T (y_1 - y_2)}$. Let $\mathcal{T} = (\sigma, \tau, \mathcal{O})$ be a timed state sequence and $(\sigma', \tau, \mathcal{O}) \in \mathrm{TS}_{\mathcal{T}}$, then

$$\rho(\sigma, \sigma') = \max\{\|\sigma_0 - \sigma'_0\|, \|\sigma_1 - \sigma'_1\|, \ldots, \|\sigma_{|\tau|-1} - \sigma'_{|\tau|-1}\|\} \tag{2}$$

is a metric on the set $(\mathbb{R}^P)^{|\mathcal{T}|}$, which is well defined since $|\mathcal{T}|$ is finite. Now that the space of state sequences is equipped with a metric, we can define a tube around a timed state sequence $\mathcal{T}$. Given an $\varepsilon > 0$, then

$$\mathrm{TS}^{\varepsilon}_{\mathcal{T}} = \{(\sigma', \tau, \mathcal{O}) \in \mathrm{TS}_{\mathcal{T}} \mid \sigma' \in B_\rho(\sigma, \varepsilon)\}$$

is the set of all timed state sequences that remain ε-close to $\mathcal{T}$.

Informally, we define the degree of robustness that a timed state sequence $\mathcal{T}$ satisfies an MTL formula ϕ to be a number $\varepsilon \in \overline{\mathbb{R}}$. Intuitively, a positive ε means that the formula ϕ is satisfiable and, moreover, that all the other timed state sequences that remain ε-close to the nominal one also satisfy ϕ. Accordingly, if ε is negative, then $\mathcal{T}$ does not satisfy ϕ and all the other timed state sequences that remain within the open tube of radius $|\varepsilon|$ also do not satisfy ϕ.

Definition 10 (Degree of Robustness). *Let $\phi \in \Phi_{\mathrm{MTL}}$, $\mathcal{T} = (\sigma, \tau, \mathcal{O}) \in TS$ and ρ be the metric (2). Define $P^{\phi}_{\mathcal{T}} := \{\sigma' \mid (\sigma', \tau, \mathcal{O}) \in TS_{\mathcal{T}} \cap \mathcal{L}(\phi)\}$, then the robustness degree $\varepsilon \in \overline{\mathbb{R}}$ of $\mathcal{T}$ with respect to ϕ is defined as $\varepsilon := \mathbf{Dist}_\rho(\sigma, P^{\phi}_{\mathcal{T}})$.*

The following proposition is derived directly from the definitions. It states that all the timed state sequences $\mathcal{S}$, which have distance from $\mathcal{T}$ less than robustness degree of $\mathcal{T}$, satisfy the same specification ϕ as $\mathcal{T}$.

Proposition 11. *Let $\phi \in \Phi_{\mathrm{MTL}}$, $\mathcal{T} = (\sigma, \tau, \mathcal{O}) \in TS$ and $\varepsilon = \mathbf{Dist}_\rho(\sigma, P^{\phi}_{\mathcal{T}})$. If $|\varepsilon| > 0$, then $\langle\langle\phi\rangle\rangle(\mathcal{S}) = \langle\langle\phi\rangle\rangle(\mathcal{T})$ for all $\mathcal{S} \in TS^{|\varepsilon|}_{\mathcal{T}}$.*

Remark 12. If $\varepsilon = 0$, then the truth value of ϕ with respect to $\mathcal{T}$ is not robust, i.e. any small perturbation of a critical state in the timed state sequence can change the satisfiability of the formula with respect to $\mathcal{T}$.

Theoretically, the set $P^{\phi}_{\mathcal{T}}$ can be computed since we have a finite number of atomic propositions, a finite trace and a known in advance sequence of time stamps. Implementation wise, though, the construction of the set $P^{\phi}_{\mathcal{T}}$ and the computation of the distance $\mathbf{Dist}_\rho(\sigma, P^{\phi}_{\mathcal{T}})$ are computationally expensive, if not infeasible (for a discussion see [16]). Therefore in this section, we develop an algorithm that computes a conservative approximation of the robustness degree ε by directly operating on the timed state sequence while avoiding set operations. As is usually the case in trade-offs, we gain computational efficiency at the expense of accuracy.

Similar to [21], we propose multi-valued semantics for the Metric Temporal Logic where the valuation function on the atomic propositions takes values over the totally ordered set $\mathfrak{R} = (\overline{\mathbb{R}}, \leq)$ according to the metric d operating on the state space $\mathbb{R}^P$ of the timed state sequence $\mathcal{T}$. For this purpose, we let the valuation function be the signed distance from the current point in the trace y to a set $\mathcal{O}(\pi)$. Intuitively, this distance represents how robustly is a point y within the set $\mathcal{O}(\pi)$. If this metric is zero, then even the smallest perturbation of the point can drive it inside or outside the set $\mathcal{O}(\pi)$, dramatically affecting membership.

We define the binary operators $\sqcup : \overline{\mathbb{R}} \times \overline{\mathbb{R}} \to \overline{\mathbb{R}}$ and $\sqcap : \overline{\mathbb{R}} \times \overline{\mathbb{R}} \to \overline{\mathbb{R}}$ using the maximum and minimum functions as $\alpha \sqcup \beta := \max\{\alpha, \beta\}$ and $\alpha \sqcap \beta := \min\{\alpha, \beta\}$. Also, for some $R \subseteq \overline{\mathbb{R}}$ we extend the above definitions as follows: $\bigsqcup R := \sup R$ and $\bigsqcap R := \inf R$. Recall that $\bigsqcup \overline{\mathbb{R}} = +\infty$ and $\bigsqcap \overline{\mathbb{R}} = -\infty$ and that any subset of $\overline{\mathbb{R}}$ has a supremum and infimum. Finally, because $\mathfrak{R}$ is a totally ordered set, it is distributive, i.e. for all $\alpha, \beta, \gamma \in \overline{\mathbb{R}}$ it is $\alpha \sqcap (\beta \sqcup \gamma) = (\alpha \sqcap \beta) \sqcup (\alpha \sqcap \gamma)$.

For the purposes of the following discussion, we use the notation $[\![\phi]\!](\mathcal{T})$ to denote the approximation to the degree of robustness with which the structure $\mathcal{T}$ satisfies the specification ϕ (formally $[\![\phi]\!] : \mathrm{TS} \to \overline{\mathbb{R}}$).

Definition 13 (Robust Semantics of MTL). *For $\phi \in \Phi_{\mathrm{MTL}}$ and $\mathcal{T} = (\sigma, \tau, \mathcal{O}) \in TS$, the robust semantics of ϕ with respect to $\mathcal{T}$ are defined as (let $\pi \in AP$ and $\psi, \phi_1, \phi_2 \in \Phi_{\mathrm{MTL}}$)*

$$\begin{aligned}
[\![\top]\!](\mathcal{T}) &:= +\infty \\
[\![\pi]\!](\mathcal{T}) &:= \mathbf{Dist}_d(\sigma_0, \mathcal{O}(\pi)) \\
[\![\neg\psi]\!](\mathcal{T}) &:= -[\![\psi]\!](\mathcal{T}) \\
[\![\phi_1 \vee \phi_2]\!](\mathcal{T}) &:= [\![\phi_1]\!](\mathcal{T}) \sqcup [\![\phi_2]\!](\mathcal{T}) \\
[\![\phi_1 \,\mathcal{U}_{\mathcal{I}} \phi_2]\!](\mathcal{T}) &:= \bigsqcup_{i=0}^{|\mathcal{T}|-1} \left([\![\tau_i \in \mathcal{I}]\!](\mathcal{T}) \sqcap [\![\phi_2]\!](\mathcal{T}\uparrow_i) \sqcap \bigsqcap_{j=0}^{i-1} [\![\phi_1]\!](\mathcal{T}\uparrow_j) \right)
\end{aligned}$$

where the unary operator $(-)$ is defined to be the negation over the reals.

Remark 14. 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 $a, b \in \overline{\mathbb{R}}$.

The following theorem states that robustness parameter that we compute using the robust semantics of MTL is an under-approximation of the actual degree of robustness (for the proof see the technical report [16]).

Theorem 15. *Let $\phi \in \Phi_{\mathrm{MTL}}$ and $\mathcal{T} \in TS$, then $|[\![\phi]\!](\mathcal{T})| \leq |\mathbf{Dist}_\rho(\sigma, P_{\mathcal{T}}^{\phi})|$. Moreover, if $[\![\phi]\!](\mathcal{T}) = \varepsilon \neq 0$, then for all $\mathcal{S} \in TS_{\mathcal{T}}^{|\varepsilon|}$ it is $\langle\!\langle \phi \rangle\!\rangle(\mathcal{S}) = \langle\!\langle \phi \rangle\!\rangle(\mathcal{T})$.*

Based on the robust semantics of MTL, we can derive a tableau formulation of the until temporal operator which is the basis for an on-the-fly monitoring algorithm. For the state of art monitoring algorithms for MTL using point-based semantics see [24] and the references therein. Using similar procedures, it is easy to derive an algorithm that returns the Boolean truth value of the formula and its robustness degree. Further details can be found in the technical report [16].

Remark 16. Consider the MTL fragment $\Phi_{\mathrm{MTL}}(\wedge, \Box)$ where the only allowed operators are the conjunction and always. In this fragment, the negation ($\neg$) can appear only in front of atomic propositions. If $\phi \in \Phi_{\mathrm{MTL}}(\wedge, \Box)$ and $\langle\langle\phi\rangle\rangle(\mathcal{T}) = \top$, then $[\![\phi]\!](\mathcal{T}) = \mathbf{Dist}_\rho(\sigma, P_{\mathcal{T}}^{\phi})$. For a discussion see [16].

4 Verification Using Robust Simulation

In this section, we show that Problem 6 can be solved in the framework of continuous-time dynamical systems. Our approach comprises three basic steps. First, we define a notion of neighborhood on the set of trajectories of the system Σ. This enables us to determine the sets of trajectories with approximately equivalent behaviors. Then, it is possible to verify that a property ϕ holds for all the traces of the dynamical system by simulating only a finite number of traces of the system Σ and evaluating their coefficients of robustness. A similar approach was proposed for the verification of safety properties in [14].

4.1 Bisimulation Function

The notion of bisimulation function has been introduced in [17] in the context of general non-deterministic metric transition systems. Intuitively, a bisimulation function evaluates how far are two states from being bisimilar: a bisimulation function bounds the distance between the observations associated with the states and is non-increasing during the evolution of the system. In the context of continuous-time dynamical systems considered in this paper, the formal definition of a bisimulation function is the following.

Definition 17. *A continuous function $V : \mathbb{R}^N \times \mathbb{R}^N \to \mathbb{R}_{\geq 0}$ is a bisimulation function for the dynamical system Σ if it satisfies the following properties*

1. *For all $x \in \mathbb{R}^N$, $V(x, x) = 0$*
2. *For all $x_1 \in \mathbb{R}^N$, $x_2 \in \mathbb{R}^N$, $V(x_1, x_2) \geq \|g(x_1) - g(x_2)\|$*
3. *For all $x_1 \in \mathbb{R}^N$, $x_2 \in \mathbb{R}^N$,*

$$\frac{\partial V}{\partial x_1}(x_1, x_2) \cdot f(x_1) + \frac{\partial V}{\partial x_2}(x_1, x_2) \cdot f(x_2) \leq 0.$$

Remark 18. Effective characterizations of bisimulation functions have been proposed for linear dynamical systems based on a set of linear matrix inequalities [25] and for nonlinear dynamical systems based on sum of squares programs [26]. Both characterizations can be interpreted in terms of convex optimization leading to efficient algorithms for the computation of bisimulation functions.

Theorem 19. *Let V be a bisimulation function, (x_1, y_1) and (x_2, y_2) be trajectories of Σ and $\mathcal{T}_1 = (\sigma^1, \tau, \mathcal{O}) \in \mathcal{L}_\tau(\Sigma)$ and $\mathcal{T}_2 = (\sigma^2, \tau, \mathcal{O}) \in \mathcal{L}_\tau(\Sigma)$ be the associated traces, then*

$$(\exists i \in \{1, 2\}. V(x_1(0), x_2(0)) < |[\![\phi]\!](\mathcal{T}_i)|) \implies (\langle\langle\phi\rangle\rangle(\mathcal{T}_1) = \langle\langle\phi\rangle\rangle(\mathcal{T}_2)) \qquad (3)$$

Proof. From the third property of Definition 17, it follows that for all $t \in \mathbb{R}_{\geq 0}$,

$$\frac{dV(x_1(t), x_2(t))}{dt} = \frac{\partial V}{\partial x_1}(x_1(t), x_2(t)) \cdot f(x_1(t)) + \frac{\partial V}{\partial x_2}(x_1(t), x_2(t)) \cdot f(x_2(t)) \leq 0.$$

Then, from the second property of Definition 17, for all $t \in \mathbb{R}_{\geq 0}$,

$$\|y_1(t) - y_2(t)\| = \|g(x_1(t)) - g(x_2(t))\| \leq V(x_1(t), x_2(t)) \leq V(x_1(0), x_2(0)).$$

Therefore, $\forall i \in \{0, 1, \ldots, |\tau| - 1\}$, it is $\|y_1(\tau_i) - y_2(\tau_i)\| \leq V(x_1(0), x_2(0))$ or

$$\rho(\sigma^1, \sigma^2) \leq V(x_1(0), x_2(0)) \tag{4}$$

Without loss of generality assume that $V(x_1(0), x_2(0)) < |[\![\phi]\!](\mathcal{T}_1)|$ and let $\varepsilon' = V(x_1(0), x_2(0))$ and $\varepsilon = |[\![\phi]\!](\mathcal{T}_1)|$. Equation (4) implies that $\mathcal{T}_2$ belongs in the closure of $TS^{\varepsilon'}_{\mathcal{T}_1}$. But $TS^{\varepsilon'}_{\mathcal{T}_1} \subset TS^{\varepsilon}_{\mathcal{T}_1}$ since $\varepsilon' < \varepsilon$. Therefore, $\mathcal{T}_2 \in TS^{\varepsilon}_{\mathcal{T}_1}$ and by applying Theorem 15 we can conclude that $\langle\!\langle\phi\rangle\!\rangle(\mathcal{T}_1) = \langle\!\langle\phi\rangle\!\rangle(\mathcal{T}_2)$. □

The previous result means that using the robust semantics of MTL and a bisimulation function, it is possible to infer the Boolean truth value of the MTL specification for an infinite number of traces. This property is exploited in the following section to verify all the traces of a system Σ using only a finite number of traces.

4.2 Sampling the Initial States

The challenge in developing a simulation-based verification algorithm is to sample the set of initial conditions in a way that ensures coverage. For this purpose, we define a discretization operator based on the bisimulation function.

Proposition 20. *Let V be a bisimulation function. For any compact set of initial conditions $I \subseteq \mathbb{R}^N$, for all $\delta > 0$, there exists a finite set of points $\{x_1, \ldots, x_r\} \subseteq I$ such that*

$$\textit{for all } x \in I, \textit{ there exists } x_i, \textit{ such that } V(x, x_i) \leq \delta. \tag{5}$$

Proof. V is continuous on $I \times I$ which is compact, therefore V is uniformly continuous on $I \times I$. Hence, for all δ, there exists ν such that

$$\forall x, x', z, z' \in I,\ \|x - x'\| \leq \nu \text{ and } \|z - z'\| \leq \nu \implies |V(x, z) - V(x', z')| \leq \delta.$$

Particularly, by setting $x' = z = z'$ and remarking that $V(x', x') = 0$, we have

$$\forall x, x' \in I,\ \|x - x'\| \leq \nu \implies V(x, x') \leq \delta.$$

Now, let us assume that for all finite set of points $\{x_1, \ldots, x_r\} \subseteq I$, there exists $x_{r+1} \in I$, such that for all x_i, $\|x - x_i\| \geq \nu$. Then, starting from a point $x_1 \in I$, we can construct a sequence $\{x_i\}_{i \in \mathbb{N}}$ such that for all $i, j \in \mathbb{N}$, $i \neq j$, we have $\|x_i - x_j\| \geq \nu$. Therefore, we cannot extract a converging subsequence of $\{x_i\}_{i \in \mathbb{N}}$ and I cannot be compact. Hence, we have proved by contradiction that there exists a finite set of points $\{x_1, \ldots, x_r\} \subseteq I$ such that for all $x \in I$, there exists x_i, such that $\|x - x_i\| \leq \nu$ which allows us to conclude (5). □

Let $Disc$ be the discretization operator which maps the compact set $I \subseteq \mathbb{R}^N$ and a strictly positive number δ to a list of points $Disc(I, \delta) = \{x_1, \ldots, x_r\}$ satisfying equation (5).

Theorem 21. *Let* $(x_1, y_1), \ldots, (x_r, y_r)$ *be trajectories of* Σ *such that* $Disc(I, \delta) = \{x_1(0), \ldots, x_r(0)\}$*. Let* $\mathcal{T}_1, \ldots, \mathcal{T}_r \in \mathcal{L}_\tau(\Sigma)$ *be the associated traces. Then,*

$$(\forall i = 1, \ldots, r.\ [\![\phi]\!](\mathcal{T}_i) > \delta) \implies (\forall \mathcal{T} \in \mathcal{L}_\tau(\Sigma).\ \langle\!\langle \phi \rangle\!\rangle(\mathcal{T}) = \top)$$

Proof. Let $\mathcal{T} \in \mathcal{L}_\tau(\Sigma)$, let (x, y) be the associated trajectory of Σ. From Proposition 20, there exists $x_i(0)$ such that $V(x(0), x_i(0)) \leq \delta$. Then, from Theorem 19 and Proposition 2 from [16], it follows that $\langle\!\langle \phi \rangle\!\rangle(\mathcal{T}) = \langle\!\langle \phi \rangle\!\rangle(\mathcal{T}_i) = \top$. □

Thus, it possible to verify that the MTL property ϕ holds for all the traces of the dynamical system Σ by evaluating the robustness degree of only a finite number of simulated trajectories.

Remark 22. Similar to Theorem 21, we can prove the following statement: if for all $i = 1, \ldots, r$ it is $-[\![\phi]\!](\mathcal{T}_i) > \delta$, then for all $\mathcal{T} \in \mathcal{L}_\tau(\Sigma)$ it is $\langle\!\langle \phi \rangle\!\rangle(\mathcal{T}) = \bot$. Therefore in this case, we can conclude that all the trajectories of Σ starting in I do not satisfy the MTL specification.

4.3 Verification Algorithm

Algorithm 1 verifies that the property ϕ holds for all the traces in $\mathcal{L}_\tau(\Sigma)$. The main idea is the following. We start with a rough discretization (using a parameter $\delta > 0$) of the set of initial states – typically we pick just one point. Then, we try to verify the property using the traces associated with these initial states. When the result of the verification is inconclusive (for example when $[\![\phi]\!](\mathcal{T}_i) < \delta$), the discretization of the initial states is refined locally (using a refinement parameter $r \in (0, 1)$) around the initial states for which we were unable to conclude the property. This algorithm, therefore, allows the fast verification of robust properties, whereas more computational effort is required for non-robust properties. The refinement operation is repeated at most K times (a user defined parameter). The algorithm can terminate in one of three possible states: (i) the property has been verified for all the initial states of the system Σ, (ii) the property has been falsified (we have found a trace that does not satisfy the specification) or (iii) we have computed a subset I' of the initial states I such that all the traces initiating from I' satisfy the MTL property. In the last case, we also get a degree of coverage of the initial states that have been verified. The proof of the correctness of the algorithm is not stated here but is very similar to that of Theorem 21.

Remark 23. Let us define $\varepsilon^* := \inf_{\mathcal{T} \in \mathcal{L}_\tau(\Sigma)} |\mathbf{Dist}_\rho(\sigma, P_{\mathcal{T}}^{\phi})|$ to be the robustness degree of the system Σ with respect to the specification ϕ. Furthermore, consider replacing $[\![\phi]\!](\mathcal{T})$ in Algorithm 1 by the theoretical quantity $\varepsilon = \mathbf{Dist}_\rho(\sigma, P_{\mathcal{T}}^{\phi})$. In this case, it can be shown that whenever $\varepsilon^* > 0$, the algorithm is complete

Algorithm 1. Temporal Logic Verification Using Simulation

```
Input: A dynamical system $\Sigma = (N, P, f, g, I, AP, \mathcal{O})$, an MTL formula $\phi$, a sequence
    of time stamps $\tau$ and numbers $\delta > 0$, $r \in (0, 1)$ and $K \in \mathbb{N}$.
 1: procedure VERIFY($\Sigma, \phi, \tau, \delta, r, K$)
 2:     $P \leftarrow Disc(I, \delta)$, $C \leftarrow \emptyset$, $k \leftarrow 0$
 3:     while $k \le K$ and $P \ne \emptyset$ do
 4:         $P' \leftarrow \emptyset$
 5:         for $x \in P$ do
 6:             Pick $\mathcal{T} \in \mathcal{L}_\tau(\Sigma)$ with $\sigma_0 = x$        ▷ Simulate $\Sigma$ for initial state $x$
 7:             if $[\![\phi]\!](\mathcal{T}) < 0$ then return "$\mathcal{L}_\tau(\Sigma) \not\subseteq \mathcal{L}(\phi)$"    ▷ $\phi$ does not hold on $\Sigma$
 8:             else if $[\![\phi]\!](\mathcal{T}) > r^k\delta$ then $C \leftarrow C \cup N_V(x, r^k\delta)$
 9:             else $P' \leftarrow P' \cup Disc(I \cap N_V(x, r^k\delta), r^{k+1}\delta)$
10:             end if        ▷ In lines 8,9: $N_V(x, \delta) = \{x' \in \mathbb{R}^N | V(x, x') \le \delta\}$
11:         end for
12:         $k \leftarrow k + 1$, $P \leftarrow P'$
13:     end while
14:     if $P = \emptyset$ then return "$\mathcal{L}_\tau(\Sigma) \subseteq \mathcal{L}(\phi)$ "        ▷ $\phi$ holds on $\Sigma$
15:     else return "$\mathcal{L}_\tau(\Sigma') \subseteq \mathcal{L}(\phi)$"    ▷ $\phi$ holds on $\Sigma' = (N, P, f, g, I \cap C, AP, \mathcal{O})$
16:     end if
17: end procedure
```

and can verify the system using only a finite number of simulations. The current algorithm may fail to be complete since we are using an under-approximation of the robustness degree (note also that $[\![\phi]\!](\mathcal{T}) = 0 \not\Rightarrow \mathbf{Dist}_\rho(\sigma, P^\phi_\mathcal{T}) = 0$).

5 Experimental Results

In this section, we demonstrate the applicability of our framework through some experimental results. We have implemented Algorithm 1 in MATLAB for linear dynamical systems and applied it to the problem presented in Example 7. A bisimulation function of the form $V(x_1, x_2) = \sqrt{(x_1 - x_2)^T M (x_1 - x_2)}$, where M is a positive definite symmetric matrix, has been computed following the technique described in [25]. We run the verification algorithm for $T \in \{0.8, 1.2, 1.6\}$ and $\theta \in \{1.4, 1.5, 1.6\}$. The results are summarized in Table 1.

Table 1. Experimental results of the verification algorithm for the transmission line example. For each value of (T, θ) the table gives whether the property ϕ holds on Σ and how many simulations of the system were necessary to conclude.

	$T = 0.8$	$T = 1.2$	$T = 1.6$
$\theta = 1.4$	False / 1	False / 7	False / 7
$\theta = 1.5$	False / 1	True / 15	True / 9
$\theta = 1.6$	False / 1	True / 15	True / 7

We can see that even though our algorithm does not have a completeness result, we were able in all the cases we considered to verify or falsify the property. The number of simulations needed for the verification depends on the value of the parameters T and θ. For instance, for the value $T = 0.8$, we were able to falsify the property using only one simulation, independently of the value of θ. For $\theta = 1.4$, the property is also false independently of the value of T. However, for $T = 1.2$ and $T = 1.6$, we needed 7 simulations of the system to falsify the property. Essentially, this means that the properties $\Box\pi_1$ with $\mathcal{O}(\pi_1) = [-1.4, 1.4]$ and $\Diamond_{[0,0.8]}\Box\pi_2$ are both false on Σ but the former much less robustly than the latter. For the cases where the property ϕ holds on Σ, we can see that the number of simulations needed for the verification is also related to the robustness of the system with respect to the property. Indeed, the larger the T and θ are, the more robust the Σ is with respect to the property ϕ and, in turn, the less is the number of the simulations that are required for verification. This is one interesting feature of our approach which relates robustness to the computational complexity of the verification.

6 Conclusions and Future Work

We have presented a novel approach to the verification problem of temporal properties of continuous time dynamical systems. Our framework reinforces a very intuitive observation: robustly (safe or unsafe) systems are easier to verify. We believe that light weight verification methods, such as the one presented here, can offer valuable assistance to the practitioner. This line of work can be extended to multiple fronts. One important direction, as advocated in [18,19,20], is to relax the requirement that all the traces should have the same sequence of time stamps. Another direction is to move toward the simulation based verification of hybrid and stochastic systems.

Acknowledgments. The authors are grateful to Zhi Han for providing them with the transmission line example. This work has been partially supported by NSF ITR Grant 0121431, NSF EHS Grant 0311123, NSF PECASE Grant 0132716 and the European community project IST-2003-507219 PROSYD (Property-based System Design).

References

1. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press, Cambridge, Massachusetts (1999)
2. Alur, R.: Timed automata. In: Proceedings of the 11th Computer Aided Verification. Volume 1633 of LNCS., Springer (1999) 8–22
3. Asarin, E., Bournez, O., Dang, T., Maler, O.: Approximate reachability analysis of piecewise linear dynamical systems. In: Hybrid Systems: Computation and Control. Volume 1790 of LNCS., Springer (2000) 21–31

4. Asarin, E., Dang, T., Girard, A.: Reachability of non-linear systems using conservative approximations. In: Hybrid Systems: Computation and Control. Volume 2623 of LNCS., Springer (2003) 22–35
5. Alur, R., Courcoubetis, C., Halbwachs, N., Henzinger, T.A., Ho, P.H., Nicollin, X., Olivero, A., Sifakis, J., Yovine, S.: The algorithmic analysis of hybrid systems. Theoretical Computer Science **138** (1995) 3–34
6. Chutinan, A., Krogh, B.: Verification of polyhedral invariant hybrid automata using polygonal flow pipe approximations. In: Hybrid Systems: Computation and Control. Volume 1569 of LNCS., Springer (1999) 76–90
7. Henzinger, T.A., Ho, P.H., Wong-Toi, H.: Algorithmic analysis of nonlinear hybrid systems. IEEE Transactions on Automatic Control **43** (1998) 540–554
8. Frehse, G.: Phaver: Algorithmic verification of hybrid systems past hytech. In: Hybrid Systems: Computation and Control. Volume 3414 of LNCS., Springer (2005) 258–273
9. Girard, A.: Reachability of uncertain linear systems using zonotopes. In: Hybrid Systems: Computation and Control. Volume 3414 of LNCS. (2005) 291–305
10. Han, Z.: Formal Verification of Hybrid Systems using Model Order Reduction and Decomposition. PhD thesis, Dept. of ECE, Carnegie Mellon University (2005)
11. Maler, O., Nickovic, D.: Monitoring temporal properties of continuous signals. In: Proceedings of FORMATS-FTRTFT. Volume 3253 of LNCS. (2004) 152–166
12. Kapinski, J., Krogh, B.H., Maler, O., Stursberg, O.: On systematic simulation of open continuous systems. In: Hybrid Systems: Computation and Control. Volume 2623 of LNCS., Springer (2003) 283–297
13. Esposito, J.M., Kim, J., Kumar, V.: Adaptive RRTs for validating hybrid robotic control systems. In: International Workshop on the Algorithmic Foundations of Robotics. (2004)
14. Girard, A., Pappas, G.J.: Verification using simulation. In: Hybrid Systems: Computation and Control (HSCC). Volume 3927 of LNCS., Springer (2006) 272 – 286
15. Koymans, R.: Specifying real-time properties with metric temporal logic. Real-Time Systems **2** (1990) 255–299
16. Fainekos, G.E., Pappas, G.J.: Robustness of temporal logic specifications for finite state sequences in metric spaces. Technical Report MS-CIS-06-05, Dept. of CIS, Univ. of Pennsylvania (2006)
17. Girard, A., Pappas, G.J.: Approximation metrics for discrete and continuous systems. Technical Report MS-CIS-05-10, Dept. of CIS, Univ. of Pennsylvania (2005)
18. 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. (2003) 163–171
19. Henzinger, T.A., Majumdar, R., Prabhu, V.S.: Quantifying similarities between timed systems. In: FORMATS. Volume 3829 of LNCS., Springer (2005) 226–241
20. Alur, R., Torre, S.L., Madhusudan, P.: Perturbed timed automata. In: Hybrid Systems: Computation and Control. Volume 3414 of LNCS. (2005) 70–85
21. de Alfaro, L., Faella, M., Henzinger, T.A., Majumdar, R., Stoelinga, M.: Model checking discounted temporal properties. In: Tools and Algorithms for the Construction and Analysis of Systems. Volume 2988 of LNCS., Springer (2004) 77–92
22. Press, W.H., Flannery, B.P., Teukolsky, S.A., Vetterling, W.T.: Numerical Recipes: The Art of Scientific Computing. 2nd edn. Cambridge University Press, Cambridge (UK) and New York (1992)
23. Ouaknine, J., Worrell, J.: On the decidability of metric temporal logic. In: 20th IEEE Symposium on Logic in Computer Science (LICS). (2005) 188–197

24. Thati, P., Rosu, G.: Monitoring algorithms for metric temporal logic specifications. In: Runtime Verification. Volume 113 of ENTCS., Elsevier (2005) 145–162
25. Girard, A., Pappas, G.J.: Approximate bisimulations for constrained linear systems. In: Proceedings of 44th IEEE Conference on Decision and Control and European Control Conference. (2005) 4700–4705
26. Girard, A., Pappas, G.J.: Approximate bisimulations for nonlinear dynamical systems. In: Proceedings of 44th IEEE Conference on Decision and Control and European Control Conference. (2005) 684–689

Undecidable Problems About Timed Automata

Olivier Finkel

Equipe de Logique Mathématique
U.F.R. de Mathématiques, Université Paris 7
2 Place Jussieu 75251 Paris cedex 05, France
finkel@logique.jussieu.fr.

Abstract. We solve some decision problems for timed automata which were raised by S. Tripakis in [Tri04] and by E. Asarin in [Asa04]. In particular, we show that one cannot decide whether a given timed automaton is determinizable or whether the complement of a timed regular language is timed regular. We show that the problem of the minimization of the number of clocks of a timed automaton is undecidable. It is also undecidable whether the shuffle of two timed regular languages is timed regular. We show that in the case of timed Büchi automata accepting infinite timed words some of these problems are Π_1^1-hard, hence highly undecidable (located beyond the arithmetical hierarchy).[1]

Keywords: Timed automata; timed Büchi automata; timed regular (ω)-languages; decision problems; universality problem; determinizability; complementability; shuffle operation; minimization of the number of clocks.

1 Introduction

R. Alur and D. Dill introduced in [AD94] the notion of timed automata reading timed words. Since then the theory of timed automata has been much studied and used for specification and verification of timed systems.

In a recent paper, E. Asarin raised a series of questions about the theoretical foundations of timed automata and timed languages which were still open and wrote: "I believe that getting answers to them would substantially improve our understanding of the area" of timed systems, [Asa04].

Some of these questions concern decision problems "à la [Tri04]". For instance : "Is it possible, given a timed automaton $\mathcal{A}$, to decide whether it is equivalent to a deterministic one ?".

S. tripakis showed in [Tri04] that there is no algorithm which, given a timed automaton $\mathcal{A}$, decides whether it is equivalent to a deterministic one, **and** if this is the case gives an equivalent deterministic automaton $\mathcal{B}$. But the above

[1] Part of the results stated in this paper were presented very recently in the Bulletin of the EATCS [Fin05, Fin06].

question of the decidability of the determinizability alone (where we do not require the construction of the witness $\mathcal{B}$) was still open.

We give in this paper the answer to this question and to several other ones of [Tri04, Asa04]. In particular, we show that one cannot decide whether a given timed automaton is determinizable or whether the complement of a timed regular language is timed regular. We study also the corresponding problems but with "bounded resources" stated in [Tri04].

For that purpose we use a method which is very similar to that one used in [Fin03b] to prove undecidability results about infinitary rational relations, reducing the universality problem, which is undecidable, to some other decision problems.

We study also the problem of the minimization of the number of clocks of a timed automaton, showing that one cannot decide, for a given timed automaton $\mathcal{A}$ with n clocks, $n \geq 2$, whether there is an equivalent timed automaton $\mathcal{B}$ with at most $n-1$ clocks.

The question of the closure of the class of timed regular languages under shuffle was also raised by E. Asarin in [Asa04]. C. Dima proved in [Dim05] that timed regular expressions with shuffle characterize timed languages accepted by stopwatch automata. This implies that the class of timed regular languages is not closed under shuffle. We proved this result independently in [Fin06]. We recall the proof here, giving a simple example of two timed regular languages whose shuffle is not timed regular. Next we use this example to prove that one can not decide whether the shuffle of two given timed regular languages is timed regular.

We extend also the previous undecidability results to the case of timed Büchi automata accepting infinite timed words. In this case many problems are Π_1^1-hard, hence highly undecidable (located beyond the arithmetical hierarchy), because the universality problem for timed Büchi automata, which is itself Π_1^1-hard, [AD94], can be reduced to these other decision problems.

We mention that part of the results stated in this paper were presented very recently in the Bulletin of the EATCS [Fin05, Fin06].

The paper is organized as follows. We recall usual notations in Section 2. The undecidability of determinizability or regular complementability for timed regular languages is proved in Section 3. The problem of minimization of the number of clocks is studied in Section 4. Results about the shuffle operation are stated in Section 5. Finally we extend in Section 6 some undecidability results to the case of timed Büchi automata.

2 Notations

We assume the reader to be familiar with the basic theory of timed languages and timed automata (TA) [AD94].

The set of positive reals will be denoted $\mathcal{R}$. A (finite length) timed word over a finite alphabet Σ is of the form $t_1.a_1.t_2.a_2 \ldots t_n.a_n$, where, for all integers $i \in [1, n]$, $t_i \in \mathcal{R}$ and $a_i \in \Sigma$. It may be seen as a *time-event sequence*, where the $t_i \in \mathcal{R}$ represent time lapses between events and the letters $a_i \in \Sigma$ represent events. The set of all (finite length) timed words over a finite alphabet Σ is the set $(\mathcal{R} \times \Sigma)^\star$. A timed language is a subset of $(\mathcal{R} \times \Sigma)^\star$. The complement (in $(\mathcal{R} \times \Sigma)^\star$) of a timed language $L \subseteq (\mathcal{R} \times \Sigma)^\star$ is $(\mathcal{R} \times \Sigma)^\star - L$ denoted L^c.

We consider a basic model of timed automaton, as introduced in [AD94]. A timed automaton $\mathcal{A}$ has a finite set of states and a finite set of transitions. Each transition is labelled with a letter of a finite input alphabet Σ. We assume that each transition of $\mathcal{A}$ has a set of clocks to reset to zero and only *diagonal-free* clock guard [AD94].

A timed automaton $\mathcal{A}$ is said to be deterministic iff it satisfies the two following requirements:

(a) $\mathcal{A}$ has only one start state, and
(b) if there are multiple transitions starting at the same state with the same label, then their clock constraints are mutually exclusive.
Then a deterministic timed automaton $\mathcal{A}$ has at most one run on a given timed word [AD94].

As usual, we denote by $L(\mathcal{A})$ the timed language accepted (by final states) by the timed automaton $\mathcal{A}$. A timed language $L \subseteq (\mathcal{R} \times \Sigma)^\star$ is said to be timed regular iff there is a timed automaton $\mathcal{A}$ such that $L = L(\mathcal{A})$.

An infinite timed word over a finite alphabet Σ is of the form $t_1.a_1.t_2.a_2.t_3.a_3 \ldots$, where, for all integers $i \geq 1$, $t_i \in \mathcal{R}$ and $a_i \in \Sigma$. It may be seen as an infinite *time-event sequence*. The set of all infinite timed words over Σ is the set $(\mathcal{R} \times \Sigma)^\omega$. A timed ω-language is a subset of $(\mathcal{R} \times \Sigma)^\omega$. The complement (in $(\mathcal{R} \times \Sigma)^\omega$) of a timed ω-language $L \subseteq (\mathcal{R} \times \Sigma)^\omega$ is $(\mathcal{R} \times \Sigma)^\omega - L$ denoted L^c.

We consider a basic model of timed Büchi automaton, (TBA), as introduced in [AD94]. We assume, as in the case of TA accepting finite timed words, that each transition of $\mathcal{A}$ has a set of clocks to reset to zero and only *diagonal-free* clock guard [AD94]. The timed ω-language accepted by the timed Büchi automaton $\mathcal{A}$ is denoted $L_\omega(\mathcal{A})$. A timed language $L \subseteq (\mathcal{R} \times \Sigma)^\omega$ is said to be timed ω-regular iff there is a timed Büchi automaton $\mathcal{A}$ such that $L = L_\omega(\mathcal{A})$.

3 Complementability and Determinizability

We first state the undecidability of determinizability or regular complementability for timed regular languages.

Theorem 1. *It is undecidable to determine, for a given TA $\mathcal{A}$, whether*

1. *$L(\mathcal{A})$ is accepted by a deterministic TA.*
2. *$L(\mathcal{A})^c$ is accepted by a TA.*

Proof. It is well known that the class of timed regular languages is not closed under complementation. Let Σ be a finite alphabet and let $a \in \Sigma$. Let A be the set of timed words of the form $t_1.a.t_2.a \ldots t_n.a$, where, for all integers $i \in [1, n]$,

$t_i \in \mathcal{R}$ and there is a pair of integers (i, j) such that $i, j \in [1, n]$, $i < j$, and $t_{i+1} + t_{i+2} + \ldots + t_j = 1$. The timed language A is formed by timed words containing only letters a and such that there is a pair of a's which are separated by a time distance 1. The timed language A is regular but its complement can not be accepted by any timed automaton because such an automaton should have an unbounded number of clocks to check that no pair of a's is separated by a time distance 1, [AD94].

We shall use the undecidability of the universality problem for timed regular languages: one cannot decide, for a given timed automaton $\mathcal{A}$ with input alphabet Σ, whether $L(\mathcal{A}) = (\mathcal{R} \times \Sigma)^\star$, [AD94].

Let c be an additional letter not in Σ. For a given timed regular language $L \subseteq (\mathcal{R} \times \Sigma)^\star$, we are going to construct another timed language $\mathcal{L}$ over the alphabet $\Gamma = \Sigma \cup \{c\}$ defined as the union of the following three languages.

– $\mathcal{L}_1 = L.(\mathcal{R} \times \{c\}).(\mathcal{R} \times \Sigma)^\star$
– $\mathcal{L}_2$ is the set of timed words over Γ having no c's or having at least two c's.
– $\mathcal{L}_3 = (\mathcal{R} \times \Sigma)^\star.(\mathcal{R} \times \{c\}).A$, where A is the above defined timed regular language over the alphabet Σ.

The timed language $\mathcal{L}$ is regular because L and A are regular timed languages. There are now two cases.

(1) **First case.** $L = (\mathcal{R} \times \Sigma)^\star$. Then $\mathcal{L} = (\mathcal{R} \times (\Sigma \cup \{c\}))^\star$. Therefore $\mathcal{L}$ has the minimum possible complexity. $\mathcal{L}$ is of course accepted by a deterministic timed automaton (without any clock). Moreover its complement $\mathcal{L}^c$ is empty thus it is also accepted by a deterministic timed automaton (without any clock).
(2) **Second case.** L is strictly included into $(\mathcal{R} \times \Sigma)^\star$. Then there is a timed word $u = t_1.a_1.t_2.a_2 \ldots t_n.a_n \in (\mathcal{R} \times \Sigma)^\star$ which does not belong to L. Consider now a timed word $x \in (\mathcal{R} \times \Sigma)^\star$. It holds that $u.1.c.x \in \mathcal{L}$ iff $x \in A$. Then we have also : $u.1.c.x \in \mathcal{L}^c$ iff $x \in A^c$.

    We are going to show that $\mathcal{L}^c$ is not timed regular. Assume on the contrary that there is a timed automaton $\mathcal{A}$ such that $\mathcal{L}^c = L(\mathcal{A})$. There are only finitely many possible global states (including the clock values) of $\mathcal{A}$ after the reading of the initial segment $u.1.c$. It is clearly not possible that the timed automaton $\mathcal{A}$, from these global states, accept all timed words in A^c and only these ones, for the same reasons which imply that A^c is not timed regular. Thus $\mathcal{L}^c$ is not timed regular. This implies that $\mathcal{L}$ is not accepted by any deterministic timed automaton because the class of deterministic regular timed languages is closed under complement.

In the first case $\mathcal{L}$ is accepted by a deterministic timed automaton and $\mathcal{L}^c$ is timed regular. In the second case $\mathcal{L}$ is not accepted by any deterministic timed automaton and $\mathcal{L}^c$ is not timed regular. But one cannot decide which case holds because of the undecidability of the universality problem for timed regular languages. □

Below $TA(n, K)$ denotes the class of timed automata having at most n clocks and where constants are at most K. In [Tri04], Tripakis stated the following problems which are similar to the above ones but with "bounded resources".

Problem 10 of [Tri04]. Given a TA $\mathcal{A}$ and non-negative integers n, K, does there exist a TA $\mathcal{B} \in TA(n, K)$ such that $L(\mathcal{B})^c = L(\mathcal{A})$? If so, construct such a $\mathcal{B}$.

Problem 11 of [Tri04]. Given a TA $\mathcal{A}$ and non-negative integers n, K, does there exist a deterministic TA $\mathcal{B} \in TA(n, K)$ such that $L(\mathcal{B}) = L(\mathcal{A})$? If so, construct such a $\mathcal{B}$.

Tripakis showed that these problems are not algorithmically solvable. He asked also whether these bounded-resource versions of previous problems remain undecidable if we do not require the construction of the witness $\mathcal{B}$, i.e. if we omit the sentence "If so construct such a $\mathcal{B}$" in the statement of Problems 10 and 11. It is easy to see, from the proof of preceding Theorem, that this is actually the case because we have seen that, in the first case, $\mathcal{L}$ and $\mathcal{L}^c$ are accepted by deterministic timed automata *without any clock*.

4 Minimization of the Number of Clocks

The following problem was shown to be undecidable by S. Tripakis in [Tri04].

Problem 5 of [Tri04]. Given a TA $\mathcal{A}$ with n clocks, does there exists a TA $\mathcal{B}$ with $n-1$ clocks, such that $L(\mathcal{B}) = L(\mathcal{A})$? If so, construct such a $\mathcal{B}$.

The corresponding decision problem, where we require only a Yes / No answer but no witness in the case of a positive answer, was left open in [Tri04].
Using a very similar reasoning as in the preceding section, we can prove that this problem is also undecidable.

Theorem 2. *Let $n \geq 2$ be a positive integer. It is undecidable to determine, for a given TA $\mathcal{A}$ with n clocks, whether there exists a TA $\mathcal{B}$ with $n-1$ clocks, such that $L(\mathcal{B}) = L(\mathcal{A})$.*

Proof. Let Σ be a finite alphabet and let $a \in \Sigma$. Let $n \geq 2$ be a positive integer, and A_n be the set of timed words of the form $t_1.a.t_2.a \ldots t_k.a$, where, for all integers $i \in [1, k]$, $t_i \in \mathcal{R}$ and there are n pairs of integers (i, j) such that $i, j \in [1, k]$, $i < j$, and $t_{i+1} + t_{i+2} + \ldots + t_j = 1$. The timed language A_n is formed by timed words containing only letters a and such that there are n pairs of a's which are separated by a time distance 1. A_n is a timed regular language but it can not be accepted by any timed automaton with less than n clocks, see [HKW95].

Let c be an additional letter not in Σ. For a given timed regular language $L \subseteq (\mathcal{R} \times \Sigma)^\star$ accepted by a TA with at most n clocks, we construct another timed language $\mathcal{V}_n$ over the alphabet $\Gamma = \Sigma \cup \{c\}$ defined as the union of the following three languages.

- $\mathcal{V}_{n,1} = L.(\mathcal{R} \times \{c\}).(\mathcal{R} \times \Sigma)^\star$
- $\mathcal{V}_{n,2}$ is the set of timed words over Γ having no c's or having at least two c's.
- $\mathcal{V}_{n,3} = (\mathcal{R} \times \Sigma)^\star.(\mathcal{R} \times \{c\}).A_n$.

The timed language $\mathcal{V}_n$ is regular because L and A_n are regular timed languages. Moreover it is easy to see that $\mathcal{V}_n$ is accepted by a TA with at most n clocks, because L and A_n are accepted by timed automata with at most n clocks. There are now two cases.

(1) **First case.** $L = (\mathcal{R} \times \Sigma)^\star$. Then $\mathcal{V}_n = (\mathcal{R} \times (\Sigma \cup \{c\}))^\star$, thus $\mathcal{V}_n$ is accepted by a (deterministic) timed automaton *without any clock.*

(2) **Second case.** L is strictly included into $(\mathcal{R} \times \Sigma)^\star$. Then there is a timed word $u = t_1.a_1.t_2.a_2 \ldots t_k.a_k \in (\mathcal{R} \times \Sigma)^\star$ which does not belong to L. Consider now a timed word $x \in (\mathcal{R} \times \Sigma)^\star$. It holds that $u.1.c.x \in \mathcal{V}_n$ iff $x \in A_n$.

Towards a contradiction, assume that $\mathcal{V}_n$ is accepted by a timed automaton $\mathcal{B}$ with at most $n-1$ clocks. There are only finitely many possible global states (including the clock values) of $\mathcal{B}$ after the reading of the initial segment $u.1.c$. It is clearly not possible that the timed automaton $\mathcal{B}$, from these global states, accept all timed words in A_n and only these ones, because it has less than n clocks.

But one cannot decide which case holds because of the undecidability of the universality problem for timed regular languages accepted by timed automata with n clocks, where $n \geq 2$. □

Remark 3. *For timed automata with only one clock, the inclusion problem, hence also the universality problem, have recently been shown to be decidable by J. Ouaknine and J. Worrell [OW04]. Then the above method can not be applied. It is easy to see that it is decidable whether a timed regular language accepted by a timed automaton with only one clock is also accepted by a timed automaton without any clock.*

5 Shuffle Operation

It is well known that the class of timed regular languages is closed under union, intersection, but not under complementation. Another usual operation is the shuffle operation. Recall that the shuffle $x \bowtie y$ of two elements x and y of a monoid M is the set of all products of the form $x_1 \cdot y_1 \cdot x_2 \cdot y_2 \cdots x_n \cdot y_n$ where $x = x_1 \cdot x_2 \cdots x_n$ and $y = y_1 \cdot y_2 \cdots y_n$.

This operation can naturally be extended to subsets of M by setting, for $R_1, R_2 \subseteq M$, $R_1 \bowtie R_2 = \{x \bowtie y \mid x \in R_1 \text{ and } y \in R_2\}$.

We know that the class of regular (untimed) languages is closed under shuffle. The question of the closure of the class of timed regular languages under shuffle was raised by E. Asarin in [Asa04]. C. Dima proved in [Dim05] that timed regular expressions with shuffle characterize timed languages accepted by stopwatch automata. This implies that the class of timed regular languages is not closed under shuffle. We proved this result independently in [Fin06].

We are going to reprove this here, giving a simple example of two timed regular languages whose shuffle is not timed regular. Next we shall use this example to prove that one cannot decide whether the shuffle of two given timed regular languages is timed regular.

Theorem 4. *The shuffle of timed regular languages is not always timed regular.*

Proof. Let a, b be two different letters and $\Sigma = \{a, b\}$.
Let R_1 be the language of timed words over Σ of the form

$$t_1 \cdot a \cdot 1 \cdot a \cdot t_2 \cdot a$$

for some positive reals t_1 and t_2 such that $t_1 + 1 + t_2 = 2$, i.e. $t_1 + t_2 = 1$.
It is clear that R_1 is a timed regular language of finite timed words.

Remark. As remarked in [AD94, page 217], a timed automaton can compare delays with constants, but it cannot remember delays. If we would like a timed automaton to be able to compare delays, we should add clock constraints of the form $x + y \leq x' + y'$ for some clock values x, y, x', y'. But this would greatly increase the expressive power of automata: the languages accepted by such automata are not always timed regular, and if we allow the addition primitive in the syntax of clock constraints, then the emptiness problem for timed automata would be undecidable [AD94, page 217].

Notice that the above language R_1 is timed regular because a timed automaton $\mathcal{B}$ reading a word of the form $t_1 \cdot a \cdot 1 \cdot a \cdot t_2 \cdot a$, for some positive reals t_1 and t_2, can compare the delays t_1 and t_2 in order to check that $t_1 + t_2 = 1$. This is due to the fact that the delay between the two first occurrences of the event a is *constant* equal to 1.

Using the shuffle operation we shall construct a language $R_1 \bowtie R_2$, for a regular timed language R_2. Informally speaking, this will "insert a variable delay" between the two first occurrences of the event a and the resulting language $R_1 \bowtie R_2$ will not be timed regular.

We now give the details of this construction.

Let R_2 be the language of timed words over Σ of the form

$$1 \cdot b \cdot s \cdot b$$

for some positive real s.

The language R_2 is of course also a timed regular language.

We are going to prove that $R_1 \bowtie R_2$ is not timed regular.

Towards a contradiction, assume that $R_1 \bowtie R_2$ is timed regular. Let R_3 be the set of timed words over Σ of the form

$$t_1 \cdot a \cdot 1 \cdot b \cdot s \cdot b \cdot 1 \cdot a \cdot t_2 \cdot a$$

for some positive reals t_1, s, t_2. It is clear that R_3 is timed regular. On the other hand the class of timed regular languages is closed under intersection thus the timed language $(R_1 \bowtie R_2) \cap R_3$ would be also timed regular. But this language is simply the set of timed words of the form $t_1 \cdot a \cdot 1 \cdot b \cdot s \cdot b \cdot 1 \cdot a \cdot t_2 \cdot a$, for some positive reals t_1, s, t_2 such that $t_1 + t_2 = 1$.

Assume that this timed language is accepted by a timed automaton $\mathcal{A}$.

Consider now the reading by $\mathcal{A}$ of a word of the form $t_1 \cdot a \cdot 1 \cdot b \cdot s \cdot b \cdot 1 \cdot a \cdot t_2 \cdot a$, for some positive reals t_1, s, t_2.

After reading the initial segment $t_1 \cdot a \cdot 1 \cdot b \cdot s \cdot b \cdot 1 \cdot a$ the value of any clock of $\mathcal{A}$ can only be $t_1 + s + 2$, $2 + s$, $1 + s$, or 1.

If the clock value of a clock $\mathcal{C}$ has been at some time reset to zero, its value may be $2 + s$, $1 + s$, or 1. So the value t_1 is not stored in the clock value and this clock can not be used to compare t_1 and t_2 in order to check that $t_1 + t_2 = 1$.

On the other hand if the clock value of a clock $\mathcal{C}$ has not been at some time reset to zero, then, after reading $t_1 \cdot a \cdot 1 \cdot b \cdot s \cdot b \cdot 1 \cdot a$, its value will be $t_1 + s + 2$. This must hold for uncountably many values of the real s, and again the value $t_1 + s + 2$ can not be used to accept, from the global state of $\mathcal{A}$ after reading the initial segment $t_1 \cdot a \cdot 1 \cdot b \cdot s \cdot b \cdot 1 \cdot a$, only the word $t_2 \cdot a$ for $t_2 = 1 - t_1$.

This implies that $(R_1 \bowtie R_2) \cap R_3$ hence also $(R_1 \bowtie R_2)$ are not timed regular. □

We can now state the following result:

Theorem 5. *It is undecidable to determine whether the shuffle of two given timed regular languages is timed regular.*

Proof. We shall use again the undecidability of the universality problem for timed regular languages: one cannot decide, for a given timed automaton $\mathcal{A}$ with input alphabet Σ, whether $L(\mathcal{A}) = (\mathcal{R} \times \Sigma)^\star$.

Let $\Sigma = \{a, b\}$, and c be an additional letter not in Σ. For a given timed regular language $L \subseteq (\mathcal{R} \times \Sigma)^\star$, we are going firstly to construct another timed language $\mathcal{L}$ over the alphabet $\Gamma = \Sigma \cup \{c\}$.

The language $\mathcal{L}$ is defined as the union of the following three languages.

- $\mathcal{L}_1 = L.(\mathcal{R} \times \{c\}).(\mathcal{R} \times \Sigma)^\star$
- $\mathcal{L}_2$ is the set of timed words over Γ having no c's or having at least two c's.
- $\mathcal{L}_3 = (\mathcal{R} \times \Sigma)^\star.1.c.R_1$, where R_1 is the above defined timed regular language over the alphabet Σ.

The timed language $\mathcal{L}$ is regular because L and R_1 are regular timed languages.

Consider now the language $\mathcal{L} \bowtie R_2$, where R_2 is the above defined regular timed language.

There are now two cases.

(1) **First case.** $L = (\mathcal{R} \times \Sigma)^\star$. Then $\mathcal{L} = (\mathcal{R} \times (\Sigma \cup \{c\}))^\star$ and $\mathcal{L} \bowtie R_2 = (\mathcal{R} \times (\Sigma \cup \{c\}))^\star$. Thus $\mathcal{L} \bowtie R_2$ is timed regular.

(2) **Second case.** L is strictly included into $(\mathcal{R} \times \Sigma)^\star$.
Towards a contradiction, assume that $\mathcal{L} \bowtie R_2$ is timed regular. Then the timed language $\mathcal{L}_4 = (\mathcal{L} \bowtie R_2) \cap [(\mathcal{R} \times \Sigma)^\star.1.c.R_3]$, where R_3 is the above defined timed regular language, would be also timed regular because it would be the intersection of two timed regular languages.

On the other hand L is strictly included into $(\mathcal{R} \times \Sigma)^\star$ thus there is a timed word $u = t_1.a_1.t_2.a_2 \ldots t_n.a_n \in (\mathcal{R} \times \Sigma)^\star$ which does not belong to L.

Consider now a timed word $x \in (\mathcal{R} \times \Sigma)^\star$. It holds that $u.1.c.x \in \mathcal{L}_4$ iff $x \in (R_1 \bowtie R_2) \cap R_3$.

We are going to show now that $\mathcal{L}_4$ is not timed regular. Assume on the contrary that there is a timed automaton $\mathcal{A}$ such that $\mathcal{L}_4 = L(\mathcal{A})$. There are only finitely many possible global states (including the clock values) of $\mathcal{A}$ after the reading of the initial segment $u.1.c$. It is clearly not possible that the timed automaton $\mathcal{A}$, from these global states, accept all timed words in $(R_1 \bowtie R_2) \cap R_3$ and only these ones, for the same reasons which imply that $(R_1 \bowtie R_2) \cap R_3$ is not timed regular. Thus $\mathcal{L}_4$ is not timed regular and this implies that $\mathcal{L} \bowtie R_2$ is not timed regular.

In the first case $\mathcal{L} \bowtie R_2$ is timed regular. In the second case $\mathcal{L} \bowtie R_2$ is not timed regular. But one cannot decide which case holds because of the undecidability of the universality problem for timed regular languages. □

We can also study the corresponding problems with "bounded resources":

Problem 1. Given two timed automata $\mathcal{A}$ and $\mathcal{B}$ and non-negative integers n, K, does there exist a TA $\mathcal{C} \in TA(n, K)$ such that $L(\mathcal{C}) = L(\mathcal{A}) \bowtie L(\mathcal{B})$?

Problem 2. Given two timed automata $\mathcal{A}$ and $\mathcal{B}$ and an integer $n \geq 1$, does there exist a TA $\mathcal{C}$ with less than n clocks such that $L(\mathcal{C}) = L(\mathcal{A}) \bowtie L(\mathcal{B})$?

Problem 3. Given two timed automata $\mathcal{A}$ and $\mathcal{B}$, does there exist a deterministic TA $\mathcal{C}$ such that $L(\mathcal{C}) = L(\mathcal{A}) \bowtie L(\mathcal{B})$?

From the proof of above Theorem 5, it is easy to see that these problems are also undecidable. Indeed in the first case $\mathcal{L} \bowtie R_2$ was accepted by a deterministic timed automaton without any clocks. And in the second case $\mathcal{L} \bowtie R_2$ was not accepted by any timed automaton.

E. Asarin, P. Carpi, and O. Maler have proved in [ACM02] that the formalism of timed regular expressions (with intersection and renaming) has the same expressive power than timed automata. C. Dima proved in [Dim05] that timed regular expressions with shuffle characterize timed languages accepted by stopwatch automata. We refer the reader to [Dim05] for the definition of stopwatch automata.

Dima showed that, from two timed automata $\mathcal{A}$ and $\mathcal{B}$, one can construct a stopwatch automaton $\mathcal{C}$ such that $L(\mathcal{C}) = L(\mathcal{A}) \bowtie L(\mathcal{B})$. Thus we can infer the following corollaries from the above results.

Notice that in [ACM02, Dim05] the authors consider automata with epsilon-transitions while in this paper we have only considered timed automata without epsilon-transitions, although we think that many results could be extended to the case of automata with epsilon-transitions. So in the statement of the following corollaries we consider stopwatch automata with epsilon-transitions but only timed automata without epsilon-transitions.

Corollary 6. *One cannot decide, for a given stopwatch automaton $\mathcal{A}$, whether there exists a timed automaton $\mathcal{B}$ (respectively, a deterministic timed automaton $\mathcal{B}$) such that $L(\mathcal{A}) = L(\mathcal{B})$.*

Corollary 7. *One cannot decide, for a given stopwatch automaton $\mathcal{A}$ and non-negative integers n, K, whether there exists a timed automaton $\mathcal{B} \in TA(n, K)$ such that $L(\mathcal{A}) = L(\mathcal{B})$.*

Corollary 8. *One cannot decide, for a given stopwatch automaton $\mathcal{A}$ and an integer $n \geq 1$, whether there exists a timed automaton $\mathcal{B}$ with less than n clocks such that $L(\mathcal{A}) = L(\mathcal{B})$.*

6 Timed Büchi Automata

The previous undecidability results can be extended to the case of timed Büchi automata accepting infinite timed words. Moreover in this case many problems are highly undecidable (Π_1^1-hard) because the universality problem for timed Büchi automata, which is itself Π_1^1-hard, [AD94], can be reduced to these problems.

For more information about the analytical hierarchy (containing in particular the class Π_1^1) see the textbook [Rog67].

We now consider first the problem of determinizability or regular complementability for timed regular ω-languages.

Theorem 9. *The following problems are Π_1^1-hard.*
For a given TBA $\mathcal{A}$, determine whether :

1. *$L_\omega(\mathcal{A})$ is accepted by a deterministic TBA.*
2. *$L_\omega(\mathcal{A})^c$ is accepted by a TBA.*

Proof. Let Σ be a finite alphabet and let $a \in \Sigma$. Let, as in Section 3, A be the set of timed words containing only letters a and such that there is a pair of a's which are separated by a time distance 1. The timed language A is regular but its complement is not timed regular [AD94].

We shall use the Π_1^1-hardness of the universality problem for timed regular ω-languages:

Let c be an additional letter not in Σ. For a given timed regular ω-language $L \subseteq (\mathcal{R} \times \Sigma)^\omega$, we can construct another timed language $\mathcal{L}$ over the alphabet $\Gamma = \Sigma \cup \{c\}$ defined as the union of the following three languages.

– $\mathcal{L}_1 = A.(\mathcal{R} \times \{c\}).(\mathcal{R} \times \Sigma)^\omega$, where A is the above defined timed regular language over the alphabet Σ.
– $\mathcal{L}_2$ is the set of infinite timed words over Γ having no c's or having at least two c's.
– $\mathcal{L}_3 = (\mathcal{R} \times \Sigma)^\star.(\mathcal{R} \times \{c\}).L$.

The timed ω-language $\mathcal{L}$ is regular because L is a regular timed ω-language and A is a regular timed language. There are now two cases.

(1) **First case.** $L = (\mathcal{R} \times \Sigma)^\omega$. Then $\mathcal{L} = (\mathcal{R} \times (\Sigma \cup \{c\}))^\omega$. Therefore $\mathcal{L}$ has the minimum possible complexity and it is accepted by a deterministic TBA (without any clock). Moreover its complement $\mathcal{L}^c$ is empty thus it is also accepted by a deterministic TBA (without any clock).

(2) **Second case.** L is strictly included into $(\mathcal{R} \times \Sigma)^\omega$, i.e. L^c is non-empty. It is then easy to see that :

$$\mathcal{L}^c = A^c.(\mathcal{R} \times \{c\}).L^c$$

where $\mathcal{L}^c = (\mathcal{R} \times \Gamma)^\omega - \mathcal{L}$, $A^c = (\mathcal{R} \times \Sigma)^\star - A$, and $L^c = (\mathcal{R} \times \Sigma)^\omega - L$.

We are going to show that $\mathcal{L}^c$ is not timed ω-regular. Assume on the contrary that there is a TBA $\mathcal{A}$ such that $\mathcal{L}^c = L_\omega(\mathcal{A})$. Consider the reading of a timed ω-word of the form $x.1.c.u$, where $x \in (\mathcal{R}\times\Sigma)^\star$ and $u \in (\mathcal{R}\times\Sigma)^\omega$, by the TBA $\mathcal{A}$. When reading the initial segment $x.1.c$, the TBA $\mathcal{A}$ has to check that $x \in A^c$, i.e. that no pair of a's in x is separated by a time distance 1; this is clearly not possible for the same reasons which imply that A^c is not timed regular (see above Section 3). Thus $\mathcal{L}^c$ is not timed ω-regular. This implies that $\mathcal{L}$ is not accepted by any deterministic TBA because the class of deterministic regular timed ω-languages is closed under complement, [AD94].

In the first case $\mathcal{L}$ is accepted by a deterministic TBA and $\mathcal{L}^c$ is timed ω-regular. In the second case $\mathcal{L}$ is not accepted by any deterministic TBA and $\mathcal{L}^c$ is not timed ω-regular.

This ends the proof because the universality problem for timed Büchi automata is Π^1_1-hard, [AD94]. □

As in the case of TA reading finite length timed words, we can consider the corresponding problems with "bounded resources".

Below $TBA(n, K)$ denotes the class of timed Büchi automata having at most n clocks, where constants are at most K.

Problem A. Given a TBA $\mathcal{A}$ and non-negative integers n, K, does there exist a TBA $\mathcal{B} \in TBA(n, K)$ such that $L_\omega(\mathcal{B})^c = L_\omega(\mathcal{A})$?

Problem B. Given a TBA $\mathcal{A}$ and non-negative integers n, K, does there exist a deterministic TBA $\mathcal{B} \in TBA(n, K)$ such that $L_\omega(\mathcal{B}) = L_\omega(\mathcal{A})$?

We can infer from the proof of preceding Theorem, that these problems are also Π^1_1-hard, because we have seen that, in the first case, $\mathcal{L}$ and $\mathcal{L}^c$ are accepted by deterministic timed Büchi automata *without any clock*.

In a very similar manner, using the same ideas as in the proof of Theorems 2 and 9, we can study the problem of minimization of the number of clocks for timed Büchi automata. We can then show that it is Π^1_1-hard, by reducing to it the universality problem for timed Büchi automata with n clocks, where $n \geq 2$, which is Π^1_1-hard. So we get the following result.

Theorem 10. *Let $n \geq 2$ be a positive integer. It is Π^1_1-hard to determine, for a given TBA $\mathcal{A}$ with n clocks, whether there exists a TBA $\mathcal{B}$ with $n-1$ clocks, such that $L_\omega(\mathcal{B}) = L_\omega(\mathcal{A})$.*

Remark 11. *We have already mentioned that, for timed automata with only one clock, the universality problem is decidable [OW04]. On the other hand, for timed Büchi automata with only one clock, the universality problem has been recently shown to be undecidable by P. A. Abdulla, J. Deneux, J. Ouaknine, and J. Worrell in [ADOW05]. However it seems to us that, in the paper [ADOW05], this problem is just proved to be undecidable and not Π_1^1-hard. Then we can just infer that the above theorem is still true for $n = 1$ if we replace "Π_1^1-hard" by "undecidable".*

Acknowledgements. Thanks to the anonymous referees for useful comments on a preliminary version of this paper.

References

[AD94] R. Alur and D. Dill, A Theory of Timed Automata, Theoretical Computer Science, Volume 126, p. 183-235, 1994.

[AM04] R. Alur and P. Madhusudan, Decision Problems for Timed Automata: A Survey, in Formal Methods for the Design of Real-Time Systems, International School on Formal Methods for the Design of Computer, Communication and Software Systems, SFM-RT 2004, Revised Lectures. Lecture Notes in Computer Science, Volume 3185, Springer, 2004, p. 1-24.

[ADOW05] P. A. Abdulla, J. Deneux, J. Ouaknine, and J. Worrell, Decidability and Complexity Results for Timed Automata via Channel Machines, in the Proceedings of the International Conference ICALP 2005, Lecture Notes in Computer Science, Volume 3580, Springer, 2005, p. 1089-1101.

[Asa04] E. Asarin, Challenges in Timed Languages, From Applied Theory to Basic Theory, Bulletin of the European Association for Theoretical Computer Science, Volume 83, p. 106-120, June 2004.

[ACM02] E. Asarin, P. Caspi, and O. Maler, Timed Regular Expressions, Journal of the ACM, Volume 49 (2), 2002, p. 172-206.

[Dim05] C. Dima, Timed Shuffle Expressions, in the Proceedings of the 16th International Conference on Concurrency Theory, CONCUR 2005, Lecture Notes in Computer Science, Volume 3653, Springer, 2005, p. 95-109.

[Fin03b] O. Finkel, Undecidability of Topological and Arithmetical Properties of Infinitary Rational Relations, RAIRO-Theoretical Informatics and Applications, Volume 37 (2), 2003, p. 115-126.

[Fin05] O. Finkel, On Decision Problems for Timed Automata, Bulletin of the European Association for Theoretical Computer Science, Volume 87, 2005, p. 185-190.

[Fin06] O. Finkel, On the Shuffle of Timed Regular languages, Bulletin of the European Association for Theoretical Computer Science, Volume 88, 2006, p. 182-184.

[HKW95] T. A. Henzinger, P. W. Kopke, and H. Wong-Toi, The Expressive Power of Clocks, in the Proceedings of the 22nd International Colloquium, ICALP95, Lecture Notes in Computer Science, Volume 944, Springer, 1995, p. 417-428.

[OW04] J. Ouaknine and J. Worrell, On the Language Inclusion Problem for Timed Automata: Closing a Decidability Gap, in the Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science, LICS 2004, Turku, Finland, IEEE Computer Society, 2004, p. 54-63.

[Rog67] H. Rogers, Theory of Recursive Functions and Effective Computability, McGraw-Hill, New York, 1967.

[Tri04] S. Tripakis, Folk Theorems on the Determinization and Minimization of Timed Automata, in the Proceedings of FORMATS'2003, Lecture Notes in Computer Science, Volume 2791, p. 182-188, 2004.

On Timed Simulation Relations for Hybrid Systems and Compositionality

Goran Frehse

VERIMAG
goran.frehse@verimag.fr
http://www-verimag.imag.fr/~frehse

Abstract. Timed and weak timed simulation relations are often used to show that operations on hybrid systems result in equivalent behavior or in conservative overapproximations. Given that systems are frequently designed and verified in a modular approach, it is desirable that this relationship is compositional, which is not the case for hybrid systems in general. We identify subclasses of linear hybrid automata that are compositional with respect to timed, respectively weak timed simulation.

1 Introduction

Hybrid automata are notoriously hard to analyze, so they are often overapproximated with hybrid automata of simpler dynamics, see [1,2,3] and references therein. The proofs used to show that the constructed automata are indeed conservative frequently involve timed simulation, or a weak variant that allows unobservable transitions as long as they don't change the variables. The analysis is usually challenging even for the abstracted system, and increases exponentially with the number of components and variables. Compositional reasoning is known as a valuable tool to counter this problem. However, timed and weak timed simulation are not compositional for hybrid automata with arbitrary dynamics. Consequently, a successful compositional analysis of the abstracted system does not imply safety of the original system when timed simulation was used in proving conservativeness.

In this paper we identify classes of hybrid automata that are compositional with respect to timed, respectively weak timed simulation. If such a class is used to overapproximate a system, conservativeness is consequently guaranteed and compositional reasoning valid. These results are directly applicable to strengthen the overapproximation operators in [1,2,3] with respect to compositionality.

Related Work. We use the hybrid automata in [4] with minor modifications. We define a subset of the controlled variables as output variables, specify the activities via their derivatives, include a set of initial states, and consider the same controlled variables in all locations. The hybrid automata in [4] are known to be compositional for trace inclusion [4], see [5] for applications. The controlled variables are needed to prove compositionality. We add output variables to hide internal (non-output) behavior, i.e., so we can compare automata whose output variables behave identically while the internal workings may be different. More sophisticated hybrid input/output-automata (HIOA) are proposed and studied in detail in [6]. HIOA impose input-enabledness that we do not require, so the hybrid automata in this paper are equivalent to the pre-HIOA of [6]. The stricter I/O-distinction in [6] may be used to ensure some liveness properties; we only consider safety. We use a compositional type of simulation from [6], which we call *trace simulation* to set it apart from timed simulation.

Timed simulation is usually defined using labeled transition system (LTS) semantics [7]. Our definition is directly based on runs of hybrid automata, but is otherwise equivalent. In earlier work, we proposed semantic criteria for compositionality of timed simulation without giving an interpretation on the hybrid automaton level, and not for weak timed simulation [8]. The framework used in this paper presents a substantial improvement and simplification, and our previous results on compositionality and assume/guarantee-reasoning from [9,10] can be transferred to it. For the sake of brevity, we provide mostly proof sketches. Detailed proofs for most of the results (except those involving overlap-closure) can be found in [10].

In the following section, we present our hybrid automata and their semantics. In Sect. 3 we define trace and timed simulation, as well as their weak counterparts. In Sect. 4 we identify compositional subclasses for these types of simulation. Finally, we draw some conclusions in Sect. 5.

2 Hybrid Automata

We use a standard hybrid automaton model and parallel composition operator from [4], to which we add a subset of output variables. A variable is either an *uncontrolled variable* (also called input), and can therefore change arbitrarily at any time, or *controlled*. In parallel composition, controlled variables can not be changed independently by other automata in the composition. These elements are essential to compositionality [11]. A subset of the controlled variables are *output* variables, which, together with the uncontrolled variables, define the externally visible behavior of the automaton. Note that the uncontrolled variables may be restricted in their derivatives, and can only change arbitrarily inside the invariant. This allows us to model causal and noncausal coupling between variables, which is useful, e.g., to model conservation laws.

Preliminaries. Given a set $X = \{x_1, \ldots, x_n\}$ of variables, a *valuation* is a function $v : X \to \mathbb{R}$. We use $\dot{X}$ to denote the set $\{\dot{x}_1, \ldots, \dot{x}_n\}$ of dotted variables, and X' to denote the set $\{x'_1, \ldots, x'_n\}$ of primed variables. Let $V(X)$ denote the set of valuations over X. The *projection* of v to variables $\bar{X} \subseteq X$ is $v{\downarrow_{\bar{X}}} = \{x \to v(x) | x \in \bar{X}\}$. The *embedding* of a set $U \subseteq V(X)$ into variables $\bar{X} \supseteq X$ is the largest subset of $V(\bar{X})$ whose projection is in U, written as $U|^{\bar{X}}$. When a valuation u over X and a valuation v over $\bar{X}$ *agree*, i.e., $u{\downarrow_{X \cap \bar{X}}} = v{\downarrow_{X \cap \bar{X}}}$, we use $u \sqcup v$ to denote the valuation w defined by $w{\downarrow_X} = u$ and $w{\downarrow_{\bar{X}}} = v$. Arithmetic operations on valuations are defined in the straightforward way. An *activity* over X is a function $f : \mathbb{R}^{\geq 0} \to V(X)$. Let *Acts*$(X)$ denote the set of activities over X. The *derivative* $\dot{f}$ of an activity f is an activity over $\dot{X}$, defined analogously to the derivative in $\mathbb{R}^n$. The extension of operators from valuations to activities is done pointwise. Let $const_X(Y) = \{(v, v')|v, v' \in V(X), v{\downarrow_Y} = v'{\downarrow_Y}\}$. The convex hull of a set of valuations S written as $chull(S)$.

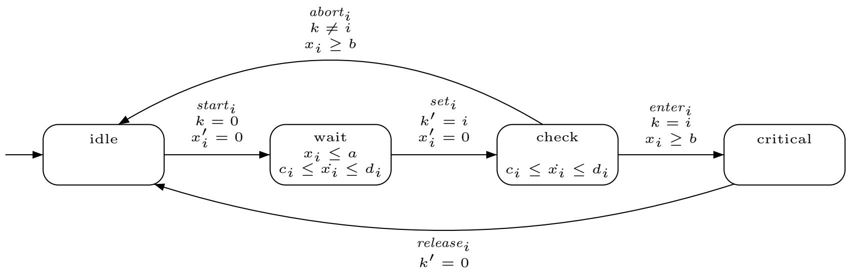
Definition 1 (Hybrid Automaton). (modified from [4]) *A* hybrid automaton *(HA)* $A = (Loc, (X, O, C), Lab, Edg, Flow, Inv, Init)$ *consists of:*

– *A finite set Loc called* locations.
– *A finite set called* variables X, *a subset* C *of* X *called* controlled *variables and a subset* $O \subseteq C$ *called* output *variables. Let* $I = X \setminus C$ *be the* input *variables and* $E = I \cup O$ *the* external *variables. A pair* $p = (l, v)$ *of a location and a valuation over* X *is a* state *of the automaton and the* state space *is* $S_H = Loc \times V(X)$. *For a state* $p = (l, v)$ *we define* $loc(p) := l$ *and* $val(p) := v$. *For a set of variables* Y, *let* $val_Y(p) := v{\downarrow_Y}$.
– *A finite set Lab of synchronization labels including the* stutter label τ.
– *A finite set Edg of edges called* transitions. *Each transition* $e = (l, a, \mu, l')$ *consists of a* source, *respectively* target *locations* $l, l' \in Loc$, *a synchronization label* $a \in Lab$, *and a* jump relation $\mu \subseteq V(X)^2$. *We require that for every location* $l \in Loc$ *there is a* stutter transition $(l, \tau, const_X(C), l) \in Edg$.
– *A set* $Flow \subseteq Loc \times V(X \cup \dot{X})$ *called* flows.
– *A set* $Inv \subseteq Loc \times V(X)$ *called* invariant.
– *A set* $Init \subseteq Inv$ *called* initial states.

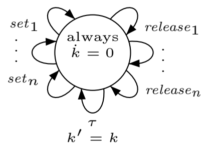
A class of hybrid automata of particular interest are *linear hybrid automata* (LHA), since they can be analyzed using simple polyhedral computations [7]. LHA are defined as follows. A *linear constraint* over a set of variables $X = \{x_1, \ldots, x_n\}$ is of the form $\sum_i a_i x_i \bowtie b$, where $\bowtie \in \{<, \leq\}$. A (convex) *linear predicate* is a (conjunctive) boolean combination of linear constraints. A linear hybrid automaton has invariants and initial states defined in each location by a linear predicate over the variables, jump relations defined by a linear predicate over $X \cup X'$, and flow valuations defined by convex linear predicates over $\dot{X}$.

Definition 2 (Parallel Composition). [4] *Hybrid automata* H_1, H_2 *are* compatible *if* $C_1 \cap C_2 = \emptyset$, $X_1 \cap C_2 \subseteq O_2$ *and* $X_2 \cap C_1 \subseteq O_1$. *The* parallel composition *of compatible hybrid automata* H_1, H_2 *is the hybrid automaton* H *with*

– $Loc = Loc_1 \times Loc_2$,
– $X = X_1 \cup X_2$, $C = C_1 \cup C_2$, $O = O_1 \cup O_2$, $Lab = Lab_1 \cup Lab_2$
– $((l_1, l_2), a, \mu, (l_2', l_2')) \in Edg$ *iff*
  • $(l_1, a_1, \mu_1, l_1') \in Edg_1$ *and* $(l_2, a_2, \mu_2, l_2') \in Edg_2$
  • *either* $a = a_1 = a_2$, *or* $a = a_1 \notin Lab_2$ *and* $a_2 = \tau$, *or* $a_1 = \tau$ *and* $a = a_2 \notin Lab_1$,
  • $\mu = \{(v, v')|(v{\downarrow_{X_i}}, v'{\downarrow_{X_i}}) \in \mu_i\}$;
– $Flow(l_1, l_2) = Flow_1(l_1)|^{X \cup \dot{X}} \cap Flow_2(l_2)|^{X \cup \dot{X}}$;
– $Inv(l_1, l_2) = Inv_1(l_1)|^X \cap Inv_2(l_2)|^X$;
– $Init(l_1, l_2) = Init_1(l_1)|^X \cap Init_2(l_2)|^X$.



(a) Process P_i



(b) Shared variable S

Fig. 1. Compositional model of timing based mutual-exclusion protocol in [12]

Example 1. Consider the model of a timing based mutual-exclusion protocol shown in Fig. 1. In every location l of P_i, there is a transition (l, τ, μ, l) with $\mu = \{(v, v')|v(x_i) = v'(x_i), v(k), v'(k) \in \mathbb{R}\}$ (omitted from the figure). The system is considered *safe* if there are never two or more processes in the critical section at the same time. It is a compositional adaptation of the model given in [12], and parameterized to n processes with time constants c_i and d_i that represent the minimal, respectively maximal, skew of their clocks. The processes P_i have a controlled variable x_i to model their local clock and an input variable k that models a semaphore. Because none of the processes controls k, it is modeled separately in an automaton S we call a *shared variable model.* S has k as a controlled variable and fixes its derivative to zero. It gives the processes access to k by synchronizing on transitions that wish to change the value of k. Note that it does not restrict the change of k in these transitions.

Semantics. We define the semantics of hybrid automata with *runs*, which we construct from *atomic runs* that represent a period of elapsing time followed by a (discrete) transition. The change of variables over time is described by an admissible activity. An activity $f(t) \in Acts(X)$ is called *admissible* over an interval $[0, \delta]$ in a location l if $\delta = 0$, or $\forall t, 0 \leq t \leq \delta : f(t) \in Inv(l), f(t) \sqcup \dot{f}(t) \in Flow(l)$. In weak runs, we consider τ-transitions that do not change the variables as unobservable.

Definition 3 (Run). *An* atomic run $\sigma = p \xrightarrow{\delta, f, a} p'$ *consists of* source and target states p, p'*, a* duration $\delta \in \mathbb{R}^{\geq 0}$*, an activity f over X called* witness *and a label $a \in Lab$ such that*

- $p, p' \in Inv$,
- *f is differentiable and admissible over $[0, \delta]$ in $loc(p)$ and $f(0) = val(p)$,*
- *there is a transition $(loc(p), a, \mu, loc(p')) \in Edg$ with $(f(\delta), val(p')) \in \mu$.*

A run *of a hybrid automaton H is a finite or infinite sequence*

$$\sigma = p_0 \xrightarrow{\delta_0, f_0, a_0} p_1 \xrightarrow{\delta_1, f_1, a_1} p_2 \ldots$$

such that $\sigma_i = p_i \xrightarrow{\delta_i, f_i, a_i} p_{i+1}$ is an atomic run for all $i \geq 0$. For a finite run, its length *is the number of atomic runs in the sequence. A* weak atomic run *$\sigma^w = p \overset{\delta,f,a}{\twoheadrightarrow} p'$ exists iff there is a finite run σ*

$$\sigma = p_0 \xrightarrow{\delta_0, f_0, \tau} p_1 \xrightarrow{\delta_1, f_1, \tau} \ldots \xrightarrow{\delta_{n-2}, f_{n-2}, \tau} p_{n-1} \xrightarrow{\delta_{n-1}, f_{n-1}, a} p_n$$

such that $\sum_{k=0}^{n-1} \delta_k = \delta$ and for all i,t, $0 \leq i < n-1$, $t_{i-1} \leq t \leq t_i$, holds $f(t) = f_i(t - t_{i-1})$, with $t_{-1} = 0$ and $t_i = \sum_{k=0}^{i} \delta_k$ for $0 \leq i$. A weak run *is defined analogously to a run as a sequence of weak atomic runs. A weak atomic run with all states in the same location l is called* unilocational*, and denoted by $p \overset{\delta,f,a}{\twoheadrightarrow}_l p'$.*

Remark 1. Due to the stutter transitions, there exists a run $p \xrightarrow{0,f,\tau} p$ in every state $p \in Inv$ and for every activity f with $f(0) = val(p)$. To underline that the activity is of no relevance, we may write $p \xrightarrow{0,\cdot,\tau} p$ instead.

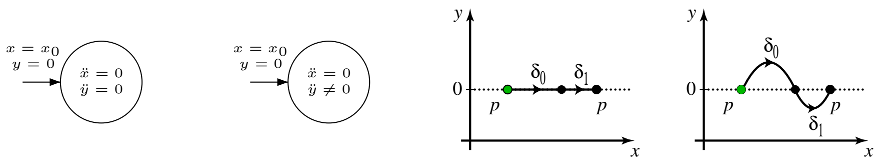
Remark 2. All except the last transition of a weak atomic run leave the variables unchanged, since $f_i(t_i - t_{i-1}) = f(t_i) = f_{i+1}(t_i - t_i)$ for $0 \leq i < n-2$.

3 Simulation Relations

To express that a hybrid automaton G is a valid abstraction of a hybrid automaton H (or equivalently that H refines G) one can establish a *simulation relation* over the product of their states. It relates a state in H to those in G that have the same, or more, behavior. Two types of simulation are predominant in literature: *trace simulation* compares the exact trace between source and target states, while *timed simulation* only considers how much time passed to get from one to the other. *Weak* versions of simulation are defined over weak traces. They are often used to show that a location with complex dynamics can be overapproximated by several locations with simpler dynamics that are connected with τ-transitions, e.g., in [13]. To be consistent with compositionality, two hybrid automata can only be compared if they have comparable inputs and outputs.

Definition 4. *H is* comparable *with G if $X_H = X_G$, $Lab_H = Lab_G$, $C_H \subseteq C_G$ and $O_H = O_G$.*

Note that according to this definition G may use less inputs than H, but not more.



(a) H (b) G (c) Run in H (d) Run in G

Fig. 2. H is not trace simulated by G, but timed simulated

Definition 5 (Trace Simulation). *A relation $R \subseteq S_H \times S_G$ is a* trace simulation relation *between comparable H and G iff for all $(p, q) \in R$, δ, f, a, p',*

$$p \xrightarrow{\delta,f,a}_H p' \quad \Rightarrow \quad \exists g, q' : q \xrightarrow{\delta,g,a}_G q' \wedge (p', q') \in R \wedge \forall t : f(t)\downarrow_{E_G} = g(t)\downarrow_{E_G} .$$

We write $H \preceq_t G$ iff there exists a trace simulation relation R such that $Init_H \subseteq R^{-1}(Init_G)$. R is called the witness *to the simulation.*

Definition 6 (Timed Simulation). *A relation $R \subseteq S_H \times S_G$ is a* timed simulation relation *between comparable H and G iff for all $(p, q) \in R$, δ, f, a, p',*

$$p \xrightarrow{\delta,f,a}_H p' \quad \Rightarrow \quad \exists g, q' : q \xrightarrow{\delta,g,a}_G q' \wedge (p', q') \in R \wedge f(0)\downarrow_{E_G} = g(0)\downarrow_{E_G} .$$

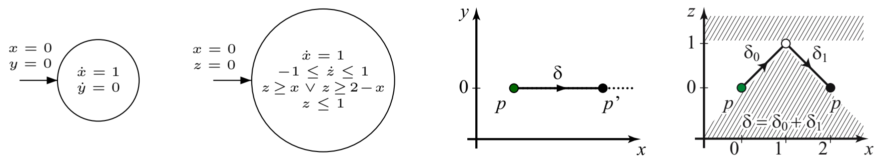
We write $H \preceq_0 G$ iff there exists a timed simulation relation R such that $Init_H \subseteq R^{-1}(Init_G)$. R is called the witness *to the simulation.*

Timed simulation forces G to have an activity that matches in the source and target states of an atomic run. It is, however, not guaranteed that H and G take the same path in between, as the following example demonstrates.

Example 2 (Trace vs. timed simulation). Consider H and G shown in Fig. 2 with $X_H = X_G = \{x, y\}$, $O_H = O_G = \{x\}$ (τ-transitions not shown). Recall that restrictions on the activities of input y are allowed. H has only trajectories in the form of straight lines, while G can nondeterministically chose any parabola with nonzero curvature. Consequently, G can not exactly match the atomic runs of H and the conditions for trace simulation are violated, i.e., $H \npreceq_t G$. However, for any given atomic run in H, G has an atomic run that, while not being identical over all points in time, matches in the timed sense, i.e., source and target states are equal and takes the same time to get from source to target. In any atomic run that might follow, G can chose a new parabola with a new curvature that matches in the timed sense. As result, $H \preceq_0 G$.

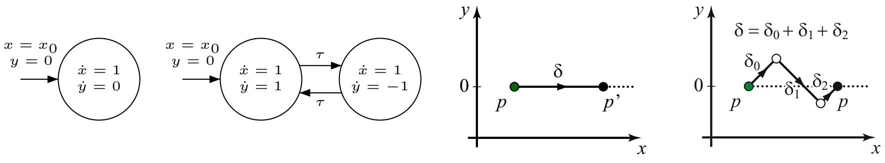
Often it is useful to consider τ-transitions unobservable in the comparison. This is achieved by looking at weak atomic runs instead of atomic runs:

Definition 7 (Weak Simulation). Weak *trace (timed) simulation is defined analogously to trace (timed) simulation over weak atomic runs, and denoted by $H \preceq_t^w G$ ($H \preceq_0^w G$).*



(a) H (b) G (c) Run in H (d) Run in G

Fig. 3. H is not trace simulated by G, but weakly trace simulated



(a) H (b) G (c) Run in H (d) Run in G

Fig. 4. H is not timed simulated by G, but weakly timed simulated

Weak atomic runs differ from atomic runs in two ways: The witnessing activity only has to be piecewise differentiable instead of differentiable, and the location can change during the period of time elapse. The following examples illustrate how this reflects in the automata that match in weak simulation, but not in simulation.

Example 3 (Weak trace simulation). Consider the LHA H and G shown in Fig. 3 with $X_H = C_H = \{x, y\}$, $X_G = C_G = \{x, z\}$, $O_H = O_G = \{x\}$. Consider the run of H shown in Fig. 3(c). Since y is not an external variable, both can take different activities with respect to y. G does not have any differentiable activity that matches because the only ones that do, e.g., a parabola from p to p', violate the flow constraint $|\dot{z}| \leq 1$. Consequently, $H \not\preceq_t G$. However, G does have a two-piece activity that can be represented by a weak atomic run, see Fig. 4(d), and $H \preceq_t^w G$.

Example 4 (Weak timed simulation). Consider the LHA H and G shown in Fig. 4 with $X_H = X_G = \{x, y\}$, $C_H = C_G = O_H = O_G = \{x\}$. Without taking τ-transitions, G can not match the activities in H, so $H \not\preceq_0 G$. However, $H \preceq_0^w G$ because every transition in H can be matched by a concatenation of transitions in G in which positive and negative change of y cancel each other out, as shown in Fig. 4(d).

We define the following equivalence relation based on simulation:

Definition 8 (Bisimulation). *A simulation relation R is a* bisimulation relation *between H and G iff R is simulation relation for $H \sim G$ and R^{-1} is a simulation relation for $G \sim H$, where $\sim \in \{\preceq_t, \preceq_0, \preceq_t^w, \preceq_0^w\}$. Bisimulation is denoted with $\cong_t, \cong_0, \cong_t^w, \cong_0^w$ depending on what relation was chosen for $\sim$.*

The different types of simulation introduced in this section are ordered with respect to how closely they distinguish behaviors of hybrid automata.

Proposition 1. *Simulation relations satisfy the following partial order:*

$$\begin{tikzcd} H \preceq_t G \arrow[r, Rightarrow] \arrow[d, Rightarrow] & H \preceq_0 G \arrow[d, Rightarrow] \\ H \preceq_t^w G \arrow[r, Rightarrow] & H \preceq_0^w G \end{tikzcd}$$

It will become apparent in the next section that the closer a simulation relation distinguishes behaviors, the larger is the class of hybrid automata for which it is compositional.

4 Compositionality

We identify subclasses of hybrid automata for which simulation is compositional. To do so we must show that the behavior of composed automata implies matching behavior of their composed specifications. Zero-duration atomic runs match for all the above types of simulation [8], so we can focus on continuous activities.

Definition 9. *A relation $\sim$ over hybrid automata is* compositional *iff*

$$H_1 \sim G_1 \wedge H_2 \sim G_2 \quad \Rightarrow \quad H_1||H_2 \sim G_1||G_2.$$

We will also use the following equivalent formulation of compositionality:

Lemma 1. *A preorder $\sim$ is* compositional *iff $H \sim G \;\Rightarrow\; H||M \sim G||M$.*

Compositionality is enforced by the fact that variables are controlled by at most one automaton.

Example 5. Consider the mutual-exclusion protocol of Ex. 1. In a noncompositional model, such as the one in [12], the analysis of n processes yields that $P_1|| \dots ||P_n$ is safe. However, this does not imply that $P_1|| \dots ||P_n||M$ is safe. M could reset k at the wrong time and cause more than one process to enter the critical section. In contrast, the compositional model does not allow M to change k in any way that is not already contained in S. Any transitions that attempt this will be blocked by the composition operator since it imposes that transitions of M either synchronize with existing transitions or with τ-transitions, which have the jump relation $const_X(C)$ and therefore leave k constant.

The simulation relations in this paper are preorders, which is easy to show using proofs similar to those in [10]. Consequently, we can use Lemma 1 to show compositionality.

Proposition 2. *Trace and timed simulation, as well as their weak variants, are preorders for comparable hybrid automata.*

If H is trace simulated by G, the external part of any activity in H must be matched exactly by an activity in G. Because M can inhibit only those same external variables, any activity in $H||M$ entails a matching activity in G. Compositionality is a direct consequence.

Proposition 3. *Trace and weak trace simulation are compositional.*

Proof. (Sketch) The compositionality of trace simulation was already shown in [6], but not that of weak trace simulation. We extend this result to weak trace simulation by showing that a weak atomic run in $H||M$ implies a weak atomic run in $G||M$ such that its target state is in the simulation relation. Our proof follows the structure of the one in [6] and relies strongly on the presence of stuttering steps. Let R_0 be the witnessing simulation relation for $H \preceq_t^w G$. We show that

$$R = \{(((l,m),x),((k,m),y)) \mid ((l,x{\downarrow_{X_H}}),(k,y{\downarrow_{X_G}})) \in R_0, x{\downarrow_{X_M}}= y{\downarrow_{X_M}}\}$$

is a witness to $H||M \preceq_t^w G||M$. A weak atomic run $\sigma_{H||M}$ in $H||M$ can be projected to weak atomic runs σ_H and σ_M in H, respectively M. Because G weakly trace simulates H, σ_H implies that there exists a weak atomic run σ_G in G with a matching activity and a matching jump at the end. Now σ_G and σ_M can be padded to have τ-transitions at identical intervals. Since G and H show the same external behavior in σ_G and σ_H, σ_G can be composed with the run σ_M to yield a weak atomic run in $G||M$. Since the external variables in the target states of $\sigma_{H||M}$ and $\sigma_{G||M}$ have the same values, the target states are in R. This shows that R is a simulation relation. It is straightfoward to show $Init_{H||M} \subseteq R^{-1}(Init_{G||M})$, which concludes the proof. □

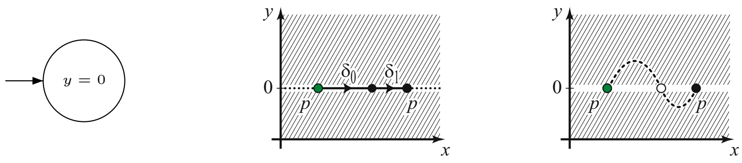
Timed simulation only forces G to have an activity that matches in the source and target states of an atomic run. It is not guaranteed that H and G take the same path in between, as the following example demonstrates.

Example 6 (Timed simulation and compositionality). Consider H and G shown in Fig. 2. In Ex. 2 we showed that $H \preceq_0 G$. If timed simulation were compositional, Lemma 1 says that simulation should still hold if we compose both sides with any K. Consider K from Fig. 5(a), with $X_K = C_K = \{y\}$. In $G||K$ the invariant $y = 0$ does not allow any timed transitions of nonzero duration, while in $H||K$ time can elapse forever, as illustrated in Fig. 5. Consequently, $H||K \not\preceq_0 G||K$.

Since timed simulation abstracts the exact activites away it is, in general, not compositional. We now show that it is, however, compositional for LHA with convex invariants. In the proof we use a lemma from [12], which states that if there is any admissible activity, there is also a linear one:

Lemma 2. (adapted from [12]) *Let l be a location of any linear hybrid automaton with a convex invariant $Inv(l)$, and $v, v' \in Inv(l)$ be any valuations inside it. If there exists an activity f that is admissible in l over some interval $[0,\delta]$ and $f(0) = v$, $f(\delta) = v'$ then $f'(t) = v + t/\delta(v' - v)$ is an equally admissible activity.*

As a consequence of this lemma, whenever there are activities with identical source and target states in LHA with convex invariants H and G, there is also



(a) K (b) Run in $H||K$ (c) No run in $G||K$

Fig. 5. Timed simulation for H, G from Fig. 2 is not compositional with K

a linear activity that is admissible in both automata. From the existence of two different activities we can thus infer the existence of a common activity, which immediately leads to compositionality:

Proposition 4. *Timed simulation is compositional for LHA with convex invariants.*

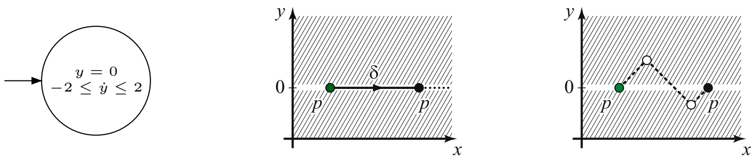
Proof. Timed simulation is compositional for compatible automata H, M if for any atomic runs $(k, u) \xrightarrow{\delta,f,\tau}_H (k', u')$ and $(l, v) \xrightarrow{\delta,g,\tau}_M (l', v')$ with $u{\downarrow_{X_H \cap X_M}} = v{\downarrow_{X_H \cap X_M}}$ and $u'{\downarrow_{X_H \cap X_M}} = v'{\downarrow_{X_H \cap X_M}}$ there is an admissible differentiable activity h in location (k, l) of $H||M$ with $h(0){\downarrow_{X_H}} = u$, $h(0){\downarrow_{X_M}} = v$ and $h(\delta){\downarrow_{X_H}} = u'$, $h(\delta){\downarrow_{X_M}} = v'$ [8]. If H, M are LHA with convex invariants, there exist, according to Lemma 2, linear activities f' and g' that witness $(k, u) \xrightarrow{\delta,f',\tau}_H (k', u')$ and $(l, v) \xrightarrow{\delta,g',\tau}_M (l', v')$. Since $f'{\downarrow_{X_H \cap X_M}} = g'{\downarrow_{X_H \cap X_M}}$, the activity h defined by $h{\downarrow_{X_H}} = f'$, $h{\downarrow_{X_M}} = g'$ is differentiable and admissible in $H||M$. □

We will later discuss compositionality of LHA with nonconvex invariants using weak simulation.

If one admits weak atomic runs in timed simulation, i.e., regards τ-transitions as unobservable, compositionality is lost even for LHA. We show this with the following counterexample.

Example 7 (Non-compositional LHA for weak timed simulation). Consider the LHA H and G from Ex. 4, shown in Fig. 4. $H \preceq_0^w G$ because every atomic run in H can be matched by a concatenation of atomic runs in G in which positive and negative change of y cancel each other out. Now consider the composition of H and G with K shown in Fig. 6(a), with $X_K = C_K = \{y\}$. For $H||K$ time can elapse forever, while for $G||K$ the invariant $y = 0$ does not allow any atomic runs of nonzero duration, as illustrated in Fig. 6. Consequently, $H||K \not\preceq_0^w G||K$.

We now identify a class of hybrid automata for which weak timed simulation is compositional. The alternation of τ-transitions with passing time allows an automaton to asymptotically mimic any activity that is a convex piecewise combination of admissible activities. Our compositional class is simply one for which we know that all the activities that the automaton can mimic are actually admissible, possibly in another location. The relevant τ-transitions in a weak atomic



(a) K (b) Run in $H||K$ (c) No run in $G||K$

Fig. 6. Weak timed simulation for H, G from Fig. 4 is not compositional with K

run do not change the variables, see Remark 2. The mimicking must therefore take place in the vicinity of the intersection of two invariants that are connected with τ-transitions. We demand that any such mimicking can take place entirely within one location, formally as follows:

Definition 10. *A hybrid automaton H is* overlap-closed *if for any (l, u) there is a $\delta_{max}(l, u)$ such that $(l, u) \xrightarrow{\delta,f,\tau} (l', u')$ with $\delta \leq \delta_{max}(l, u)$ implies*

(i) a run $(l, u) \xrightarrow{\delta,f,\tau}_l (l, u') \xrightarrow{0,\cdot,\tau} (l', u')$, or
(ii) a location k such that $(l, u) \xrightarrow{0,\cdot,\tau} (k, u) \xrightarrow{\delta,f,\tau}_k (k, u') \xrightarrow{0,\cdot,\tau} (l', u')$.

H is strongly overlap-closed *if $\inf_{l,u} \delta_{max}(l, u) > 0$.*

Remark 3. Note that any hybrid automaton is overlap-closed if it does not have different locations connected by τ-transitions.

According to Lemma 2, LHA with convex invariants always have a linear activity between two points of the same location. Combining this fact with the assumption of overlap-closedness, we can conclude that if $H \preceq_0^w G$, a weak run in H is matched in G with a weak run witnessed by the same external activity. From there it is straightforward to show compositionality as follows:

Proposition 5. *Let H_1, H_2 be LHA and G_1, G_2 be strongly overlap-closed LHA with convex invariants and bounded derivatives. If $H_1 \preceq_0^w G_1$ and $H_2 \preceq_0^w G_2$, then $H_1||H_2 \preceq_0^w G_1||G_2$.*

Proof. (Sketch) Let R_1, R_2 be witnessing weak simulation relations for $H_1 \preceq_0^w G_1$ and $H_2 \preceq_0^w G_2$, respectively. We show that

$$R = \{(((k_1, k_2), x), ((l_1, l_2), y)) \mid ((k_i, x\downarrow_{H_i}), (l_i, y\downarrow_{G_i})) \in R_i \text{ for } i = 1, 2\}$$

is a witnessing simulation relation for $H_1||H_2 \preceq_0^w G_1||G_2$. The containment of initial states in R follows straightforwardly from the containment in R_1 and R_2. It remains to demonstrate that for any pair of states in R, a weak atomic run in $H_1||H_2$ implies a matching weak atomic run in $G_1||G_2$ such that the target states are again in R. A weak atomic run $p \xrightarrow{\delta,f,\alpha} p'$ can be split in two: a run $p \xrightarrow{\delta,f,\tau} p''$ containing only τ-transitions that leave the variables unchanged (see Remark 2) and a run $p'' \xrightarrow{0,f,\alpha}\!\!\!\!\!\!\twoheadrightarrow p'$ of zero duration. The definition of weak simulation for a run of zero duration is the same as that of timed simulation, so with Prop. 4 we can deduce that the latter part satisfies compositionality. The rest of the proof is therefore concerned with the former part of the run, which only includes τ-transitions that do not change the variables.

Because the H_i and H are LHA, a weak atomic run in $H_1||H_2$ has a witnessing run whose activity is piecewise linear [12]. We pad it with τ-transitions to obtain a run

$$\sigma_H = r_0 \xrightarrow{\delta_0,f_0,\tau} r_1 \xrightarrow{\delta_1,f_1,\tau} \dots \xrightarrow{\delta_{n-2},f_{n-2},\tau} r_{n-1} \xrightarrow{\delta_{n-1},f_{n-1},\tau} r_n$$

with durations $\delta_j \leq \delta_{min}$ for some arbitrarily small $\delta_{min} > 0$, and linear activities f_1. Every one of the atomic runs $\sigma_{j,H} = r_j \xrightarrow{\delta_j,f_j,\tau} r_{j+1}$ in σ_H projects in the H_i onto corresponding runs σ_{j,H_i}. Since H_i is weakly simulated by G_i, there must be matching weak runs σ^w_{j,G_i}, i.e., with the same valuations of the external variables in the source and target state. Let $\delta_{min} = \inf_{l,u} \delta_{max}(l,u)$ from Def. 10. According to Def. 10, this implies that there is also matching weak run $\bar{\sigma}^w_{j,G_i}$ with all time-elapse inside a single location. Because the G_i are LHA with convex invariants, it follows from Lemma 2 that the linear activity between source and target state is also admissible in that location, thus matching the one in H_i.

It remains to show that the runs in G_1 and G_2 compose to a run in $G_1||G_2$, and that the target state of this run lies in R. Recall that the source and target states of σ_{j,H_i} and $\bar{\sigma}^w_{j,G_i}$ lie in R_i, which means they have the same values in the shared external variables. The shared variables of H_1 and H_2 also have the same values in the respective states, and due to comparability the same holds for the shared variables of G_1 and G_2. By padding with τ-transitions, we can obtain witnessing nonatomic runs for $\bar{\sigma}^w_{j,G_1}$ and $\bar{\sigma}^w_{j,G_2}$ that have the same length and whose atomic runs have the same duration. Because all the nonzero activities are linear, the source and target states still match. Consequently, the runs in G_1 and G_2 compose to a run in $G_1||G_2$. Because the target states of the runs in H_i and G_i lie in R_i, the target states of the runs in $H_1||H_2$ and $G_1||G_2$ lie in R. $\square$

Example 8 (Compositional LHA for weak timed simulation). [1] Consider the LHA H from Ex. 4, shown in Fig. 4, and G' from Fig. 7, with $X_{G'} = \{x, y\}$, $C_{G'} = O_{G'} = \{x\}$. G' is overlap-closed for the sign $\bowtie = \geq$, and not overlap-closed if $\bowtie = >$. In both cases $H \preceq^w_0 G'$ because every atomic run in H can be matched by a concatenation of atomic runs in G in which positive and negative change of y cancel each other out. Now consider the composition of H and G' with K shown in Fig. 6(a), with $X_K = C_K = \{y\}$. In $G'||K$ with $\bowtie = \geq$, there are silent transitions to location c, where time can elapse forever, and consequently $H||K \preceq^w_0 G'||K$. If $\bowtie = >$, there is no run from the initial location a to location c in $G'||K$, because the invariant of location b is empty. Consequently, $H||K \not\preceq^w_0 G'||K$.

[1] Thanks to the anynomous reviewer who inspired the example.

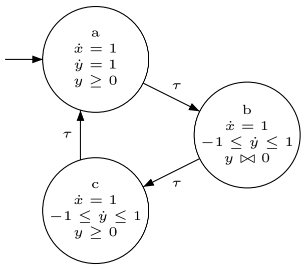


Fig. 7. LHA G', overlap-closed if $\bowtie = \geq$

For weak runs, trace simulation has the advantage over timed simulation because it is compositional for any hybrid automata. In [1], timed simulation was used to show that the invariant of a hybrid automaton can be partitioned into arbitrarily small parts with a splitting operation that does not modify the behavior. This is useful in many applications, e.g., to transform nonconvex into convex invariants, or to overapproximate the automaton with one of simpler dynamics [1]. The splitting operation is defined as follows:

Definition 11 (Invariant split). (modified from [1]) *An* (open) split $\mathcal{S}$ *for a hybrid automaton* H *maps each location* l *to a finite set* $\{S_1^l, \ldots, S_k^l\}$ *of sets of valuations over* X *such that there exists a finite (open) cover* $\mathcal{O}^l = \{O_1^l, \ldots, O_k^l\}$ *of* $Inv(l)$ *with* $S_i^l = Inv(l) \cap O_i^l$ *for* $i = 1, \ldots, k$*. The* split *of* H *along* $\mathcal{S}$ *is the hybrid automaton* $split(H, \mathcal{S}) = (Loc_{\mathcal{S}}, (X, C, O), Lab, \to_{\mathcal{S}}, Flow_{\mathcal{S}}, Inv_{\mathcal{S}}, Init_{\mathcal{S}})$ *with*

- $Loc_{\mathcal{S}} = \{(l, S) \mid l \in Loc, S \in \mathcal{S}(l)\}$,
- $\to_{\mathcal{S}} = \{((l, S), a, \mu, (l', S')) \mid (l, a, \mu, l') \in \to\}$,
- $Flow_{\mathcal{S}}((l, S)) = Flow(l)$, $Inv_{\mathcal{S}}((l, S)) = Inv(l) \cap S$, $Init_{\mathcal{S}}((l, S)) = Init(l) \cap S$.

We rephrase the following results of [1] and [3] using weak trace simulation, thus expanding their applicability to the context of compositional reasoning.

Proposition 6. *For any* H, $H \cong_t^w split(H, \mathcal{S})$ *if* $\mathcal{S}$ *is an open split or the admissible activities of* H *are analytic functions.*

Proof. In [1], it is shown that $H \cong_0^w split(H, \mathcal{S})$ if $\mathcal{S}$ is an open split. While timed simulation is used formally, the corresponding proof shows that the activities match identically over time. It is therefore straightforward to strengthen the result to weak trace bisimulation. In [3] it is shown, based on the results of [1], that the split does not have to be open if the admissible activities of H are analytic functions. □

The condition of analytic activities applies, e.g., to LHA, or hybrid automata with affine dynamics [3], whose flows are defined by conjunctions of linear constraints over $X \cup \dot{X}$.

5 Conclusions

Timed and weak timed simulation are often used to show equivalence and abstraction between hybrid automata. We identify the following subclasses of linear hybrid automata (LHA) for which these relations are compositional: LHA with convex invariants for timed simulation, and strongly overlap-closed LHA with convex invariants and bounded derivatives for weak timed simulation. An advantage of timed simulation relations is that for many LHA they can be computed, e.g., with PHAVer [9,8,2]. In addition, LHA can overapproximate any hybrid automata arbitrarily close [1]. Using the above results we can overapproximate with a compositional subclass of LHA, and thus apply compositional and assume/guarantee-reasoning to arbitrary hybrid automata.

On the downside, timed simulation is not compositional in general. Weak trace simulation, which is compositional for hybrid automata with arbitrary dynamics, can sometimes be used instead. E.g., one may substitute it for weak timed simulation in the proofs of [3,1] without having to change any essential parts of the proofs. The result is a notion of equivalence that is stronger per se and compositional. In future work we will identify subclasses for which timed simulation implies trace simulation.

Acknowledgements

The author wishes to thank Bruce Krogh and Oded Maler for their indispensable guidance and support, as well as Susanne Graf for her valuable comments. A special thanks goes to the anonymous reviewer who spotted a bug in Prop. 5.

References

1. Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. Algorithmic analysis of nonlinear hybrid systems. *IEEE Trans. Automatic Control*, 43(4):540–554, 1998.
2. Goran Frehse. PHAVer: Algorithmic verification of hybrid systems past HyTech. In Manfred Morari and Lothar Thiele, editors, *Hybrid Systems: Computation and Control (HSCC'05), Mar. 9–11, 2005, Zürich, CH*, volume 2289 of *LNCS*. Springer, 2005. PHAVer is available at `http://www.cs.ru.nl/~goranf/`.
3. Laurent Doyen, Thomas A. Henzinger, and Jean-François Raskin. Automatic rectangular refinement of affine hybrid systems. In *Proc. FORMATS'05*, volume 3829 of *LNCS*, pages 144–161. Springer, 2005.
4. Rajeev Alur, Costas Courcoubetis, Nicolas Halbwachs, Thomas A. Henzinger, Pei-Hsin Ho, Xavier Nicollin, Alfredo Olivero, Joseph Sifakis, and Sergio Yovine. The algorithmic analysis of hybrid systems. *Theor. Comp. Science*, 138(1):3–34, 1995.
5. Erika Ábrahám-Mumm, Ulrich Hannemann, and Martin Steffen. Verification of hybrid systems: Formalization and proof rules in PVS. In *Proc. IEEE Int. Conf. on Engineering of Complex Computer Systems (ICECCS 2001)*, June 2001.
6. Nancy A. Lynch, Roberto Segala, and Frits W. Vaandrager. Hybrid I/O automata. *Information and Computation*, 185(1):105–157, 2003.

7. Thomas A. Henzinger. The theory of hybrid automata. In *Proc. 11th Annual IEEE Symposium on Logic in Computer Science, LICS'96, New Brunswick, New Jersey, 27-30 July 1996*, pages 278–292. IEEE Computer Society Press, 1996.
8. Goran Frehse, Zhi Han, and Bruce H. Krogh. Assume-guarantee reasoning for hybrid i/o-automata by over-approximation of continuous interaction. In *Proc. IEEE Conf. Decision & Control (CDC'04), Dec. 14–17, 2004, Atl., Bahamas*, 2004.
9. Goran Frehse. Compositional verification of hybrid systems with discrete interaction using simulation relations. In *Proc. IEEE Conf. Computer-Aided Control System Design (CACSD'04), September 1–4, 2004, Taipei, Taiwan*, 2004.
10. Goran Frehse. *Compositional Verification of Hybrid Systems using Simulation Relations.* PhD thesis, Radboud Universiteit Nijmegen, October 2005.
11. Nancy A. Lynch and Michael J. Fischer. On describing the behavior and implementation of distributed systems. *Theoretical Computer Science*, 13(1):17–43, 1981. Special issue on Semantics of Concurrent Computation.
12. Rajeev Alur, Thomas A. Henzinger, and Pei-Hsin Ho. Automatic symbolic verification of embedded systems. *IEEE Trans. Soft. Engineering*, 22:181–201, 1996.
13. Thomas A. Henzinger and Howard Wong-Toi. Linear phase-portrait approximations for nonlinear hybrid systems. In Rajeev Alur, Thomas A. Henzinger, and Eduardo D. Sontag, editors, *Hybrid Systems III: Verification and Control*, volume 1066 of *LNCS*, pages 377–388. Springer, 1996.

Integrating Discrete- and Continuous-Time Metric Temporal Logics Through Sampling

Carlo A. Furia and Matteo Rossi

Dipartimento di Elettronica e Informazione, Politecnico di Milano
32, Piazza Leonardo da Vinci, 20133, Milano, Italy
{furia, rossi}@elet.polimi.it

Abstract. Real-time systems usually encompass parts that are best described by a continuous-time model, such as physical processes under control, together with other components that are more naturally formalized by a discrete-time model, such as digital computing modules. Describing such systems in a unified framework based on metric temporal logic requires to integrate formulas which are interpreted over discrete and continuous time.

In this paper, we tackle this problem with reference to the metric temporal logic TRIO, that admits both a discrete-time and a continuous-time semantics. We identify sufficient conditions under which TRIO formulas have a consistent truth value when moving from continuous-time to discrete-time interpretations, or vice versa. These conditions basically involve the restriction to a proper subset of the TRIO language and a requirement on the finite variability over time of the basic items in the specification formulas. We demonstrate the approach with an example of specification and verification.

Keywords: formal methods, real-time, integration, discretization, metric temporal logic, discrete time, continuous time, dense time.

1 Introduction and Motivation

The application of formal methods to the description and analysis of large and heterogenous systems inevitably requires being able to model different parts of the system using disparate notations and languages. In fact, it is usually the case that different modules are naturally described using diverse formal techniques, each one tailored to the specific nature of that component. Indeed, the past decades have seen the birth and proliferation of a plethora of different formal languages and techniques, each one usually focused on the description of a certain kind of systems and hinged on a specific approach. On the one hand, this proliferation is a good thing, as it allows the user to choose the notation and methodology that is best suited for her needs and that matches her intuition. However, this is also inevitably a hurdle to the true scalability in the application of formal techniques, since we end up having heterogeneous descriptions of large systems, where different modules, described using distinct notations, have no definite global semantics when put together. Therefore, we need to find ways to *integrate* dissimilar models into a global description which can then be analyzed, so that heterogeneity of notation.

A particularly relevant instance of the above general problem is encountered when describing real-time systems, which require a quantitative modeling of time. Commonly, such systems are composed of some parts representing physical environmental processes and some others being digital computing modules. The former ones have to model physical quantities that vary continuously over time, whereas the latter ones are digital components that are updated periodically at every (discrete) clock tick. Hence, a natural way to model the physical processes is by assuming a *continuous-time* model, and using a formalism with a compliant semantics, whereas digital components would be best described using a *discrete-time* model, and by adopting a formalism in accordance. Thus, the need to integrate continuous-time formalisms with discrete-time formalisms, which is the object of the present paper.

In particular, let us consider the framework of descriptive specifications based on (metric) temporal logic. Some temporal logic languages have semantics for both a continuous-time model and a discrete-time one: each formula of the language can be interpreted in one of the two classes of models. TRIO is an example of these logics [5], the one we are considering in this paper; MTL [14] is another well-known instance.[1] The discrete-time semantics and the continuous-time one, however, are unrelated in general, in that the same formula unpredictably changes its models when passing from one semantics to another. On the contrary, integration requires different formulas to describe parts of the *same* system, thus referring to unique underlying models.

To this end, we introduce the notion of *sampling invariance* of a specification formula. Informally, we say that a temporal logic formula is sampling invariant when its discrete-time models coincide with the samplings of all its continuous-time models (modulo some additional technical requirements). The *sampling* of a continuous-time model is a discrete-time model obtained by observing the continuous-time model at periodic instants of time. The justification for the notion of sampling invariance stems from how real systems are made. In fact, in a typical system the discrete-time part (e.g., a controller) is connected to the (probed) environment by a sampler, which communicates measurements of some physical quantities to the controller at some periodic time rate (see Figure 1). The discrete-time behaviors that the controller sees are samplings of the continuous-time behaviors that occur in the system under control. Our notion of sampling invariance captures abstractly this fact in relating a continuous-time formula to a discrete-time one, thus mirroring what happens in a real system.

Once we have a sampling invariant specification, we can integrate discrete-time and continuous-time parts, thus being able, among other things, to resort to verification in a discrete-time model, which often benefits from more automated approaches, while still being able to describe naturally physical processes in a continuous-time model.

[1] We note that all the results drawn in this paper about TRIO can be applied to MTL with little effort.

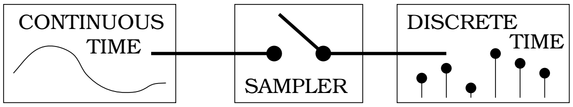


Fig. 1. A system with a sampler

Another interesting approach that could be achieved with a sampling-invariant language which is the subset of a more expressive one (TRIO, in our case) is one based on *refinement*. In the process of formal modeling of a digital component — and of a computer program in particular — one would start with a very high-level description that would refer actions and events to the "real, ideal, physical" time. The final implementation, however, will have to refer to a more concrete, *measured* view of time, such as the one achievable through periodic readings of an imperfect clock. The refinement of the specification from the ideal to the concrete could then move from full TRIO to the sampling-invariant subset of it; this latter description would be closer to the "implemented" view of time, and would therefore facilitate its realization.

Section 2 introduces $^{\mathbb{R}}_{\mathbb{Z}}$TRIO, a subset of TRIO for which sampling invariance can be achieved. Then, Section 3 defines the sampling invariance requirement and demonstrates that $^{\mathbb{R}}_{\mathbb{Z}}$TRIO formulas are invariant under sampling. Section 4 demonstrates the use of the notion of sampling invariance for integration by developing a simple example of specification and verification. Finally, Section 5 compares our approach with related works, and Section 6 draws conclusions and outlines future (and current) work. For the lack of space, we have omitted some technical details and proofs; they can be found in [9].

2 The $^{\mathbb{R}}_{\mathbb{Z}}$TRIO Metric Temporal Logic

Let us start by presenting our reference metric temporal logic, namely TRIO [10,15,5]. More precisely, this section introduces a fragment of full TRIO that we shall call $^{\mathbb{R}}_{\mathbb{Z}}$TRIO[2]; it is a syntactic and expressive subset of the former, robust with respect to our notion of sampling invariance (as it will be defined in the following Section 3). The presentation of $^{\mathbb{R}}_{\mathbb{Z}}$TRIO will be aided by an example consisting in the formal description of a simple controlled reservoir system, which will be further analyzed in Section 4.

2.1 Syntax

Whereas TRIO is based on a single modal operator named Dist [5], $^{\mathbb{R}}_{\mathbb{Z}}$TRIO adopts the bounded Until and Since as primitive operators, because this permits a simpler statement of the sampling invariance results. More precisely, let Ξ be a set of time-dependent *conditions*. These are basically Boolean expressions obtained by functional combination of basic time-dependent items with constants.

[2] You can read it as "ar-zee-TRIO".

We are going to define them precisely later on (in Section 3), for now let us just assume that they are time-dependent formulas whose truth value is defined at any given time. Let us consider a set $\mathbb{S}$ of symbols representing constants. We denote *intervals* by expressions of the form $\langle l, u\rangle$, with l, u constants from $\mathbb{S}$, $\langle$ a left parenthesis from $\{(, [\}$, and $\rangle$ a right parenthesis from $\{),]\}$; let $\mathcal{I}$ be the set of all such intervals. Then, if $\xi, \xi_1, \xi_2 \in \Xi$, $I \in \mathcal{I}$, $\langle \in \{(, [\}$, and $\rangle \in \{),]\}$, well-formed formulas ϕ are defined recursively as follows.

$$\phi ::= \xi \mid \text{Until}_{I\rangle}(\phi_1, \phi_2) \mid \text{Since}_{I\langle}(\phi_1, \phi_2) \mid \neg\phi \mid \phi_1 \wedge \phi_2$$

From these basic operators, it is customary to define a number of *derived* operators: Table 1 lists those used in this paper.[3]

Table 1. Some $^{\mathbb{R}}_{\mathbb{Z}}$TRIO derived temporal operators

Operator	$\equiv$	Definition
$\text{Releases}_{I\rangle}(\phi_1, \phi_2)$	$\equiv$	$\neg\text{Until}_{I\rangle}(\neg\phi_1, \neg\phi_2)$
$\text{Released}_{I\langle}(\phi_1, \phi_2)$	$\equiv$	$\neg\text{Since}_{I\langle}(\neg\phi_1, \neg\phi_2)$
$\exists t \in I = \langle l, u\rangle : \text{Dist}(\phi, t)$	$\equiv$	$\begin{cases} \text{Until}_{\langle l,u\rangle\rangle}(\text{true}, \phi) & \text{if } u \geq l \geq 0 \\ \text{Since}_{\langle -l,-u\rangle\rangle}(\text{true}, \phi) & \text{if } u \leq l \leq 0 \end{cases}$
$\forall t \in I = \langle l, u\rangle : \text{Dist}(\phi, t)$	$\equiv$	$\begin{cases} \text{Releases}_{\langle l,u\rangle\rangle}(\text{false}, \phi) & \text{if } u \geq l \geq 0 \\ \text{Released}_{\langle -l,-u\rangle\rangle}(\text{false}, \phi) & \text{if } u \leq l \leq 0 \end{cases}$
$\text{Dist}(\phi, d)$	$\equiv$	$\forall t \in [d, d] : \text{Dist}(\phi, t)$
$\text{Futr}(\phi, d)$	$\equiv$	$d \geq 0 \wedge \text{Dist}(\phi, d)$
$\text{Som}(\phi)$	$\equiv$	$\exists t \in (-\infty, 0] : \text{Dist}(\phi, t) \vee \exists t \in [0, +\infty) : \text{Dist}(\phi, t)$
$\text{Alw}(\phi)$	$\equiv$	$\forall t \in (-\infty, 0] : \text{Dist}(\phi, t) \wedge \forall t \in [0, +\infty) : \text{Dist}(\phi, t)$
$\text{AlwP}(\phi)$	$\equiv$	$\forall t \in (-\infty, 0) : \text{Dist}(\phi, t)$
$\text{AlwP}_{\text{i}}(\phi)$	$\equiv$	$\forall t \in (-\infty, 0] : \text{Dist}(\phi, t)$
$\text{WithinP}(\phi, \tau)$	$\equiv$	$\exists t \in (-\tau, 0) : \text{Dist}(\phi, t)$
$\text{WithinP}_{\text{ii}}(\phi, \tau)$	$\equiv$	$\exists t \in [-\tau, 0] : \text{Dist}(\phi, t)$
$\text{Lasts}(\phi, \tau)$	$\equiv$	$\forall t \in (0, \tau) : \text{Dist}(\phi, t)$
$\text{Lasts}_{\text{ii}}(\phi, \tau)$	$\equiv$	$\forall t \in [0, \tau] : \text{Dist}(\phi, t)$

2.2 Semantics

In defining $^{\mathbb{R}}_{\mathbb{Z}}$TRIO semantics we assume that constants in $\mathbb{S}$ are interpreted naturally as numbers from the time domain $\mathbb{T}$ plus the symbols $\pm\infty$, which are treated as usual. Correspondingly, intervals $\mathcal{I}$ are interpreted as intervals of $\mathbb{T}$ which are closed/open to the left/right (as usual square brackets denote an included endpoint, and round brackets denote an excluded one).

Then, we define the semantics of an $^{\mathbb{R}}_{\mathbb{Z}}$TRIO formula using as interpretations mappings from the time domain $\mathbb{T}$ to the domain D the basic items map their values to. Let $\mathcal{B}_{\mathbb{T}}$ be the set of all such mappings, which we call *behaviors*, and let $b \in \mathcal{B}_{\mathbb{T}}$ be any element from that set. If we denote by $\xi|_{b(t)}$ the truth value of the condition ξ at time $t \in \mathbb{T}$ according to behavior b, we can define the

[3] Disjunction $\vee$ is defined as usual. For simplicity we assume that $I = \langle l, u\rangle$ is such that $l, u \geq 0$ or $l, u \leq 0$ in the definition of the $\exists t \in I$ and $\forall t \in I$ operators.

semantics of $^{\mathbb{R}}_{\mathbb{Z}}$TRIO formulas as follows. We write $b \models_{\mathbb{T}} \phi$ to indicate that the behavior b is a model for formula ϕ under the time model $\mathbb{T}$. Thus, let us define the semantics for the generic time model $\mathbb{T}$; in practice this will be either the reals $\mathbb{R}$ or the integers $\mathbb{Z}$.

$$\begin{array}{lll}
b(t) \models_{\mathbb{T}} \xi & \text{iff} & \xi|_{b(t)} \\
b(t) \models_{\mathbb{T}} \mathrm{Until}_{I\rangle}(\phi_1, \phi_2) & \text{iff} & \text{there exists } d \in I \text{ such that } b(t+d) \models_{\mathbb{T}} \phi_2 \\
 & & \text{and, for all } u \in [0, d\rangle \text{ it is } b(t+u) \models_{\mathbb{T}} \phi_1 \\
b(t) \models_{\mathbb{T}} \mathrm{Since}_{I\langle}(\phi_1, \phi_2) & \text{iff} & \text{there exists } d \in I \text{ such that } b(t-d) \models_{\mathbb{T}} \phi_2 \\
 & & \text{and, for all } u \in \langle -d, 0] \text{ it is } b(t+u) \models_{\mathbb{T}} \phi_1 \\
b(t) \models_{\mathbb{T}} \neg\phi & \text{iff} & b(t) \not\models_{\mathbb{T}} \phi \\
b(t) \models_{\mathbb{T}} \phi_1 \wedge \phi_2 & \text{iff} & b(t) \models_{\mathbb{T}} \phi_1 \text{ and } b(t) \models_{\mathbb{T}} \phi_2 \\
b \models_{\mathbb{T}} \phi & \text{iff} & \text{for all } t \in \mathbb{T}\text{: } b(t) \models_{\mathbb{T}} \phi
\end{array}$$

Thus, an $^{\mathbb{R}}_{\mathbb{Z}}$TRIO formula ϕ constitutes the specification of a system, representing exactly all behaviors that are models of the formula. We denote by $[\![\phi]\!]_{\mathbb{T}}$ the set of all models of formula ϕ with time domain $\mathbb{T}$, i.e., $[\![\phi]\!]_{\mathbb{T}} \equiv \{b \in \mathcal{B}_{\mathbb{T}} \mid b \models_{\mathbb{T}} \phi\}$. In the remainder we will sometime use the expression "behaviors of a formula ϕ" to mean the set $[\![\phi]\!]_{\mathbb{T}}$.

2.3 The Controlled Reservoir

Let us briefly illustrate the practical use of the $^{\mathbb{R}}_{\mathbb{Z}}$TRIO language by building a descriptive specification (i.e., consisting of a set of logic axioms) for a controlled reservoir.

The reservoir is filled with some liquid; at any time, the (measured) level of the liquid is represented by a time-dependent item l that takes value in the set $\mathbb{R}_{\geq 0}$. Furthermore, the reservoir can leak and/or be filled with new liquid. This is modeled through two Boolean-valued time-dependent items L and F indicating the reservoir leaking and being filled, respectively. Therefore, the behaviors (i.e., the logical models) of our reservoir will be functions mapping the time domain $\mathbb{T}$ to the domain $D_{\mathrm{res}} = \mathbb{R}_{\geq 0} \times \{0,1\} \times \{0,1\}$.

Let us start the formal specification by writing some TRIO (not $^{\mathbb{R}}_{\mathbb{Z}}$TRIO) axioms that define how the level varies over time according to whether the reservoir is leaking and/or being filled. Thus, let us introduce two positive real constants $\mathrm{r_f}, \mathrm{r_l}$ that denote the rate at which the level increases, and at which it decreases, when the reservoir is filling and leaking, respectively. Then, we state that: whenever the reservoir is being filled (i.e., F) and is not leaking (i.e., $\neg\mathsf{L}$) for a (generic) time interval of length t (i.e., $\mathrm{Lasts}(\mathsf{F} \wedge \neg\mathsf{L}, t)$, and the level is some l at the beginning of the interval (i.e., $\mathsf{l} = l$), then at the end of the interval the level will have grown to $l + \mathrm{r_f} t$ (i.e., $\mathrm{Futr}(\mathsf{l} = l + \mathrm{r_f} t, t)$). Let us define similarly three other axioms to describe all the possible combinations of filling and leaking.

$$\mathrm{Lasts}(\mathsf{F} \wedge \mathsf{L}, t) \wedge \mathsf{l} = l \;\Rightarrow\; \mathrm{Futr}(\mathsf{l} = l + (\mathrm{r_f} - \mathrm{r_l})t, t) \tag{1}$$

$$\mathrm{Lasts}(\mathsf{F} \wedge \neg\mathsf{L}, t) \wedge \mathsf{l} = l \;\Rightarrow\; \mathrm{Futr}(\mathsf{l} = l + \mathrm{r_f} t, t) \tag{2}$$

$$\mathrm{Lasts}(\neg\mathsf{F} \wedge \mathsf{L}, t) \wedge \mathsf{l} = l \;\Rightarrow\; \mathrm{Futr}(\mathsf{l} = \max(l - \mathrm{r_l} t, 0), t) \tag{3}$$

$$\mathrm{Lasted}(\neg\mathsf{F} \wedge \neg\mathsf{L}, t) \wedge \mathsf{l} = l \;\Rightarrow\; \mathrm{Futr}(\mathsf{l} = l, t) \tag{4}$$

In order to define the control action, let us introduce two more constants. The overall goal of the control action is to keep the level above a minimum level l, whatever the leaking behavior is, by filling it whenever needed. To this end, we define a threshold level denoted by $\mathrm{t_l}$: whenever the level goes below $\mathrm{t_l}$, the controller activates the filling, as indicated by Axiom 5 below. Obviously, we assume $\mathrm{t_l} > \mathrm{l}$. Finally, let us also state that the level is initially above the threshold $\mathrm{t_l}$ (Formula 6).

$$\mathsf{l} < \mathrm{t_l} \;\Rightarrow\; \mathsf{F} \tag{5}$$

$$\mathrm{Som}(\mathrm{AlwP}(\mathsf{l} \geq \mathrm{t_l})) \tag{6}$$

Let us close this section with two remarks. First, we assume that formulas (1–4) and (6) are interpreted over $\mathbb{R}$ as time domain, since they model the behavior of a physical system; conversely, formula (5) is implicitly interpreted over $\mathbb{Z}$ as time domain, since it describes the behavior of a digital device that "watches" the physical system through a sampler. Second, while Formulas 5 and 6 qualify as $^{\mathbb{R}}_{\mathbb{Z}}$TRIO formulas, the other Formulas (1–4) involve free (time) variables (namely, t), so they are full-TRIO formulas, *not* $^{\mathbb{R}}_{\mathbb{Z}}$TRIO.

3 Sampling Invariance and Integration

The overall goal of integrating formulas that are interpreted over different time domains basically requires to define a suitable notion of *invariance*. Whenever a formula achieves this invariance it means that its "intended meaning" is preserved by changes of the temporal domain. Thus, the formula can be equivalently interpreted under any of the time domains, putting it on a common ground with the other formulas, effectively integrating it with them.

Our notion of invariance is named *sampling invariance*, and relates continuous- and discrete-time behaviors of a formula through what we call the *sampling* of a behavior. Precisely, given a continuous-time behavior $b \in \mathcal{B}_{\mathbb{R}}$, we define its *sampling* as the discrete-time behavior $\sigma_{\delta,z}[b] \in \mathcal{B}_{\mathbb{Z}}$ that agrees with b at all integer time instants corresponding to multiples of a constant $\delta \in \mathbb{R}_{>0}$ from a basic offset $z \in \mathbb{R}$. We call δ the *sampling period* and z the *origin* of the sampling. In formulas, we have the following definition:

$$\forall k \in \mathbb{Z} : \quad \sigma_{\delta,z}[b](k) \;\equiv\; b(z + k\delta)$$

3.1 Sampling Invariance

Let us now give a sensible definition of *sampling invariance*. Ideally, we would like to be able to say that, for a given formula ϕ, sampling period δ, and origin z: the sampling of any of its continuous-time behaviors is one of its discrete-time behaviors (i.e., $b \in [\![\phi]\!]_{\mathbb{R}} \Rightarrow \sigma_{\delta,z}[b] \in [\![\phi]\!]_{\mathbb{Z}}$); and, conversely, any continuous-time behavior whose sampling is a discrete-time behavior of ϕ is also a valid continuous-time behavior of ϕ (i.e., $b \in [\![\phi]\!]_{\mathbb{Z}} \Rightarrow \forall b' : (\sigma_{\delta,z}[b'] = b \Rightarrow b' \in [\![\phi]\!]_{\mathbb{R}})$).

This ideal notion of sampling invariance is not achievable, as it is, by any non-trivial temporal logic language (and $^{\mathbb{R}}_{\mathbb{Z}}$TRIO in particular), as the continuous-time and the discrete-time behaviors of a formula are in general only loosely related. In particular, let us point out two basic reasons why the above definitions must be relaxed to be pursuable.

Regularity of behaviors. The first and most relevant issue concerns the fact that the density of the time domain $\mathbb{R}$ allows for behaviors that may change an unbounded number of times between any two sampling instants; therefore, the information about these "intermediate changes" is completely lost by sampling the behavior. Consider, for instance, Figure 2(a). There, a Boolean-valued item

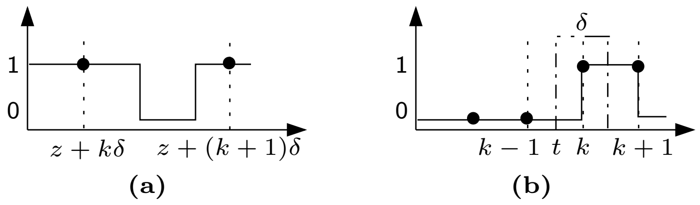


Fig. 2. (a) Change detection failure; (b) Moving interval

changes its values twice (from true to false and then back to true) within the interval, of length δ, between two adjacent sampling instants. Therefore, any continuous-time formula predicating about the value of the item within those instants may be true in continuous time and false for the sampling of the behavior, which only "detects" two consecutive true values. Instead, we would like that the "rate of change" of the items is slow-paced enough that every change is detected at some sampling instant, before it changes again.

To this end, let us introduce a *constraint* on the continuous-time behaviors of a formula: only behaviors conformant to the constraint can be invariant under sampling. The precise form of the constraint to be introduced depends on the kind of items we are dealing with in our specification (namely, whether they take values to discrete or continuous domains); we will define it precisely in the next Section 3.2. In practice, however, the constraint is expressed by an additional $^{\mathbb{R}}_{\mathbb{Z}}$TRIO formula χ, which depends, in general, on the particular specification we have written; we call χ the *behavior constraint.*

Discrete steps and continuous units. The second obstacle to achieving sampling invariance is instead a technical issue concerning measurement units. Let us illustrate it with the $^{\mathbb{R}}_{\mathbb{Z}}$TRIO formula $\text{Lasts}(\mathsf{F}, 1/2)$. In a discrete-time setting, we would like the formula to mean: for all (integer) time distances that fall in an interval between the two time instants corresponding to the sampling instants closest to 0 and 1/2, etc. Hence, we should actually adopt the formula $\text{Lasts}(\mathsf{F}, 1/2\delta)$ as discrete-time counterpart, dividing every *time constant* by the sampling period.

Conversely, there is another change in the time constants — probably less manifest — that must be introduced when interpreting a discrete-time formula in continuous time. To give the intuition, consider the formula $\text{Lasts}_{\text{ii}}(\mathsf{F}, 1)$ in discrete time: F holds for two consecutive time steps (including the current one). In continuous time, when evaluating the corresponding $\text{Lasts}_{\text{ii}}(\mathsf{F}, \delta)$ between two consecutive sampling instants (cf. instant t in Figure 2(b)), we can only state a *weaker* property about F holding, namely that F holds in a *subset* of the δ-wide interval in the future. On basic formulas, this requirement is rendered as a *shrinking* (or *stretching*, for existential formulas) of the intervals bounds by one discrete time unit.

In Section 3.2 we will define how to modify the time constants in any $^{\mathbb{R}}_{\mathbb{Z}}$TRIO formula when passing from continuous to discrete time and vice versa. Let us name *adaptation function* this simple translation rule, as it just adapts the time bounds in our formulas, without changing its structure. We denote the *adaptation* function from continuous to discrete time as $\eta^{\mathbb{R}}_{\delta}\{\cdot\}$, and the converse function from discrete to continuous time as $\eta^{\mathbb{Z}}_{\delta}\{\cdot\}$.

Definition of Sampling Invariance. We are finally able to give a formal definition of sampling invariance which, by employing the above outlined ingredients, is achievable for $^{\mathbb{R}}_{\mathbb{Z}}$TRIO formulas (as it will be shown in Section 3.3).

Definition 1 (Sampling Invariance). *Given a formula ϕ, a behavior constraint formula χ, two adaptation functions $\eta^{\mathbb{R}}_{\delta}\{\cdot\}$ and $\eta^{\mathbb{Z}}_{\delta}\{\cdot\}$, a sampling period δ, and an origin z, we say that:*

- *ϕ is* closed under sampling *iff for any continuous-time behavior $b \in \mathcal{B}_{\mathbb{R}}$:*

$$b \in [\![\phi \wedge \chi]\!]_{\mathbb{R}} \quad \Rightarrow \quad \sigma_{\delta,z}\left[b\right] \in [\![\eta^{\mathbb{R}}_{\delta}\{\phi\}]\!]_{\mathbb{Z}}$$

- *ϕ is* closed under inverse sampling *iff for any discrete-time behavior $b \in \mathcal{B}_{\mathbb{Z}}$:*

$$b \in [\![\phi]\!]_{\mathbb{Z}} \quad \Rightarrow \quad \forall b' \in [\![\chi]\!]_{\mathbb{R}} : \left(\sigma_{\delta,z}\left[b'\right] = b \;\Rightarrow\; b' \in [\![\eta^{\mathbb{Z}}_{\delta}\{\phi\} \wedge \chi]\!]_{\mathbb{R}}\right)$$

- *ϕ is* sampling invariant *iff it is closed under sampling (when interpreted in the continuous-time domain) and closed under inverse sampling (when interpreted in the discrete-time domain).*

3.2 Adaptation, Conditions, and Behavior Constraints

The proof that $^{\mathbb{R}}_{\mathbb{Z}}$TRIO is sampling invariant is given for formulas of the language that are in a *normal form*, where there is no nesting of temporal operators, and all negations are pushed inside. Consequently, negations on Since and Until are replaceable by the Releases and Released operators. As it is fully shown in [9], any $^{\mathbb{R}}_{\mathbb{Z}}$TRIO can be put in normal form, possibly introducing auxiliary time-dependent items. Therefore, in the remainder of this section we will assume to deal with formulas in normal form.

Adaptation Function. Let us define the adaptation function $\eta_\delta^{\mathbb{R}}\{\cdot\}$ from continuous to discrete time as follows, where l' is $\lfloor l/\delta \rfloor$ if $\langle$ is (, and $\lceil l/\delta \rceil$ if $\langle$ is [, and u' is $\lceil u/\delta \rceil$ if $\rangle$ is), and $\lfloor u/\delta \rfloor$ if $\rangle$ is].[4]

$$\begin{array}{ll}
\eta_\delta^{\mathbb{R}}\{\xi\} & \equiv\ \xi \\
\eta_\delta^{\mathbb{R}}\{\mathrm{Until}_{\langle l,u\rangle\rangle}(\xi_1,\xi_2)\} & \equiv\ \mathrm{Until}_{[\lfloor l/\delta \rfloor, \lceil u/\delta \rceil])}(\xi_1,\xi_2) \\
\eta_\delta^{\mathbb{R}}\{\mathrm{Since}_{\langle l,u\rangle\langle}(\xi_1,\xi_2)\} & \equiv\ \mathrm{Since}_{[\lfloor l/\delta \rfloor, \lceil u/\delta \rceil](}(\xi_1,\xi_2) \\
\eta_\delta^{\mathbb{R}}\{\mathrm{Releases}_{\langle l,u\rangle\rangle}(\xi_1,\xi_2)\} & \equiv\ \mathrm{Releases}_{\langle l',u'\rangle]}(\xi_1,\xi_2) \\
\eta_\delta^{\mathbb{R}}\{\mathrm{Released}_{\langle l,u\rangle\langle}(\xi_1,\xi_2)\} & \equiv\ \mathrm{Released}_{\langle l',u'\rangle[}(\xi_1,\xi_2) \\
\eta_\delta^{\mathbb{R}}\{\phi_1 \wedge \phi_2\} & \equiv\ \eta_\delta^{\mathbb{R}}\{\phi_1\} \wedge \eta_\delta^{\mathbb{R}}\{\phi_2\} \\
\eta_\delta^{\mathbb{R}}\{\phi_1 \vee \phi_2\} & \equiv\ \eta_\delta^{\mathbb{R}}\{\phi_1\} \vee \eta_\delta^{\mathbb{R}}\{\phi_2\}
\end{array}$$

The adaptation function $\eta_\delta^{\mathbb{Z}}\{\cdot\}$ from discrete to continuous time is instead defined as follows, where the adapted interval $\langle(l-1)\delta, (u-1)\delta\rangle$ for the Until and Since operators can indifferently be taken to include or exclude their endpoints.[5]

$$\begin{array}{ll}
\eta_\delta^{\mathbb{Z}}\{\xi\} & \equiv\ \xi \\
\eta_\delta^{\mathbb{Z}}\{\mathrm{Until}_{[l,u]]}(\xi_1,\xi_2)\} & \equiv\ \mathrm{Until}_{\langle(l-1)\delta,(u+1)\delta\rangle]}(\xi_1,\xi_2) \\
\eta_\delta^{\mathbb{Z}}\{\mathrm{Since}_{[l,u][}(\xi_1,\xi_2)\} & \equiv\ \mathrm{Since}_{\langle(l-1)\delta,(u+1)\delta\rangle[}(\xi_1,\xi_2) \\
\eta_\delta^{\mathbb{Z}}\{\mathrm{Releases}_{[l,u])}(\xi_1,\xi_2)\} & \equiv\ \mathrm{Releases}_{[(l+1)\delta,(u-1)\delta])}(\xi_1,\xi_2) \\
\eta_\delta^{\mathbb{Z}}\{\mathrm{Released}_{[l,u](}(\xi_1,\xi_2)\} & \equiv\ \mathrm{Released}_{[(l+1)\delta,(u-1)\delta](}(\xi_1,\xi_2) \\
\eta_\delta^{\mathbb{Z}}\{\phi_1 \wedge \phi_2\} & \equiv \eta_\delta^{\mathbb{Z}}\{\phi_1\} \wedge \eta_\delta^{\mathbb{Z}}\{\phi_2\} \\
\eta_\delta^{\mathbb{Z}}\{\phi_1 \vee \phi_2\} & \equiv \eta_\delta^{\mathbb{Z}}\{\phi_1\} \vee \eta_\delta^{\mathbb{Z}}\{\phi_2\}
\end{array}$$

Conditions. The definitions of the primitive conditions ξ appearing in $^{\mathbb{R}}_{\mathbb{Z}}$TRIO formulas depend on whether we deal with primitive time-dependent items take values to discrete or dense domains. This should not be confused with the density of the time domain: for behaviors mapping $\mathbb{T}$ to a generic domain D, we now distinguish whether D is a discrete or a dense set. Note that when discrete- and dense-valued items coexist in the same specification, we just have to consider them separately.

Discrete-valued items. If all primitive time-dependent items in $\Psi = \{\psi_1, \dots \psi_n\}$ have discrete codomains, then $D \equiv D_1 \times \cdots \times D_n$ where each D_i is a discrete set. Let Λ be a set of constants from the sets D_i's, and Γ be a set of functions having domains in subsets of D and codomains in some D_i's. Then, if $\lambda \in \Lambda$, $f, f_1, f_2 \in \Gamma$, and $\bowtie \in \{=, <, \leq, >, \geq\}$, conditions ξ can be defined recursively as follows, assuming type compatibility is respected:

$$\xi \ ::= \ f(\psi_1, \dots, \psi_n) \bowtie \lambda \mid f_1(\psi_1, \dots, \psi_n) \bowtie f_2(\psi_1, \dots, \psi_n) \mid \neg\xi \mid \xi_1 \wedge \xi_2$$

Dense-valued items. If all time-dependent items in Ψ have dense codomains, then $D \equiv D_1 \times \cdots \times D_n$ where each D_i is a dense set. Let Λ be a set of n-tuples of the form $\langle \langle^1 l_1, u_1 \rangle^1, \dots, \langle^n l_n, u_n \rangle^n \rangle$, with $l_i, u_i \in D_i$, $l_i \leq u_i$, $\langle^i \in \{(, [\}$, and

[4] As it is customary, the brackets $\lfloor \cdot \rfloor$ and $\lceil \cdot \rceil$ denote the floor and ceiling functions [11].

[5] Note that in discrete time we need only consider closed intervals.

$\rangle^i \in \{),]\}$, for all $i = 1, \ldots, n$. Then, if $\lambda \in \Lambda$, conditions ξ are defined simply as follows:

$$\xi \ ::= \ \langle \psi_1, \ldots, \psi_n \rangle \in \lambda \mid \neg\xi \mid \xi_1 \wedge \xi_2$$

Then, for a behavior $b : \mathbb{T} \to D$, the evaluation of conditions on b at $t \in \mathbb{T}$ is defined as obvious; in particular — for discrete-valued items — a function application $f(\psi_1, \ldots, \psi_n)$ is interpreted by taking $f(\psi_1|_{b(t)}, \ldots, \psi_n|_{b(t)})$, and — for dense-valued items — $\langle \psi_1, \ldots, \psi_n \rangle|_{b(t)} \in \lambda$ is interpreted as $\forall i = 1, \ldots, n : \psi_i(t) \in \langle^i l_i, u_i \rangle^i$ (i.e., all items are in their respective ranges at time t).

Behavior Constraints. According to whether we are considering discrete- or dense-valued items in our specification, we have two different definitions for the behavior constraint χ. Again, if discrete and dense items coexist in the same specification, we just consider both conditions for their respective items.

Discrete-valued items. The constraint on behaviors for discrete-valued items is given in Formula $\chi_\circ$ and basically requires that the value of each item in Ψ is held for δ time units, at least.[6]

$$\chi_\circ \triangleq \forall v \in D : \ (\langle \psi_1, \ldots, \psi_n \rangle = v \ \Rightarrow \ \mathrm{WithinP_{ii}}(\mathrm{Lasts_{ii}}(\langle \psi_1, \ldots, \psi_n \rangle = v, \delta), \delta))$$

Dense-valued items. The constraint on behaviors for dense-valued items is expressed by Formula $\chi_\bullet^\phi$ depends on the actual conditions ξ that we introduced in the formula ϕ. Let $\widetilde{\Xi}$ be the set of conditions that appear in ϕ; $\chi_\bullet^\phi$ requires that the value of every item in Ψ is such that its changes with respect to the conditions in $\widetilde{\Xi}$ occur at most once every δ time units. This is expressed by the following "pseudo"-$^{\mathbb{R}}_{\mathbb{Z}}$TRIO formula, where the higher-order quantification $\forall\widetilde{\xi} \in \widetilde{\Xi}$ is just a shorthand for the explicit enumeration of all the conditions in (the finite set) $\widetilde{\Xi}$.

$$\chi_\bullet^\phi \triangleq \forall\widetilde{\xi} \in \widetilde{\Xi} : \ \mathrm{WithinP_{ii}}\left(\mathrm{Lasts_{ii}}\left(\widetilde{\xi}, \delta\right), \delta\right) \vee \mathrm{WithinP_{ii}}\left(\mathrm{Lasts_{ii}}\left(\neg\widetilde{\xi}, \delta\right), \delta\right)$$

3.3 Sampling Invariance of $^{\mathbb{R}}_{\mathbb{Z}}$TRIO

We finally state the sampling invariance of $^{\mathbb{R}}_{\mathbb{Z}}$TRIO formulas.

Theorem 1 (Sampling Invariance). *$^{\mathbb{R}}_{\mathbb{Z}}$TRIO is sampling invariant with respect to the behavior constraints $\chi_\circ$ and $\chi_\bullet^\phi$, the adaptation functions $\eta_\delta^{\mathbb{R}}\{\cdot\}$ and $\eta_\delta^{\mathbb{Z}}\{\cdot\}$, for any sampling period δ and origin z.*

Proof (sketch). For the sake of space we just sketch the outline of the proof and refer to [9] for the details.

[6] Actually, while $^{\mathbb{R}}_{\mathbb{Z}}$TRIO, as we defined it above, is purely propositional, $\chi_\circ$ uses a universal quantification on non-temporal variables. However, it is not difficult to understand that this generalization does not impact on sampling-invariance.

First of all, let us notice that the proof for both dense- and discrete-valued items is the same, and it relies on the basic fact that the truth value of any condition ξ cannot change more than once between any two consecutive sampling instants, because of the behavior constraints $\chi_\circ$ and $\chi_\bullet^\phi$. Then, let ϕ be any $^{\mathbb{R}}_{\mathbb{Z}}$TRIO formula.

To prove *closure under sampling*, let b be a continuous-time behavior in $[\![\chi_\circ]\!]_{\mathbb{R}}$, $\phi' = \eta_\delta^{\mathbb{R}}\{\phi\}$, and b' be the sampling $\sigma_{\delta,z}[b]$ of behavior b with the given origin and sampling period. For a generic sampling instant $t = z + k\delta$, one can show that $b(t) \models_{\mathbb{R}} \phi$ implies $b'(k) \models_{\mathbb{Z}} \phi'$, by induction on the structure of ϕ. Even if all the details are quite convoluted, the overall idea is fairly terse: assuming ϕ holds we infer that some condition ξ holds at some instant u that depends on the bounds of the time intervals involved in ϕ; then, condition $\chi_\circ$ (or $\chi_\bullet^\phi$) ensures that the truth of ξ is held at least until the previous or the next sampling instant (w.r.t. u); therefore, the discrete-time formula ϕ', whose time bounds have been relaxed by the adaptation function $\eta_\delta^{\mathbb{R}}\{\cdot\}$, matches this sampling instant and is thus shown to hold at k.

To prove *closure under inverse sampling* let b be a discrete-time behavior, $\phi' = \eta_\delta^{\mathbb{Z}}\{\phi\}$, and b' be a continuous-time behavior such that $b' \in [\![\chi_\circ]\!]_{\mathbb{R}}$ and $b = \sigma_{\delta,z}[b']$ for the given origin and sampling period. The proof is now split into two parts. The former shows that, for a generic sampling instant $t = z + k\delta$, if $b(k) \models_{\mathbb{Z}} \phi$ then $b'(t) \models_{\mathbb{R}} \phi'$, that is ϕ' holds in continuous-time at all sampling instants. Again, this involves reasoning on the intervals involved in the operators and the condition $\chi_\circ$ (or $\chi_\bullet^\phi$). Afterwards, we show that ϕ' holds in between sampling instants: if it holds at some sampling instant t, then it is either always true in the future and past, or there exist definite "changing points" in the future and past where ϕ' changes its truth value to false. These changing points, however, can be shown to correspond to changes in the truth value of some condition ξ (because in normal form we have no nesting), and thus there can be at most one of them between any two consecutive sampling instants. A final analysis shows that indeed if any ϕ' holds at *all* sampling instants, then it also holds in *between* them, since it should otherwise change its value twice between two of them. $\square$

4 Example

Let us now develop controlled reservoir whose specification was introduced in Section 2.3. Let us consider the following invariance formula, asserting that the level of liquid never goes below 1.

$$\text{Alw}(\mathsf{l} \geq 1) \tag{7}$$

We would like to prove that 7 follows from the axioms, that is (1–6)⇒(7). Notice, however, that the axioms do not refer to a uniform time domain, and thus they need to be integrated first.

A first possible strategy would first notice that Formula 5 does not use temporal operators at all and so it could be easily interpreted over the reals. Then, one would proceed to prove (1–6)⇒(7) assuming continuous time. Such a proof is possible although not very simple — at least with respect to the simplicity of the involved formulas — as it requires to deal with regularity properties of functions with real domain (such as continuity and non-Zenoness); the reader can check out [8], where a similar proof was carried out in continuous time with the aid of a compositional inference rule.

An alternative approach to achieve the verification goal is therefore to exploit the results about integration. First of all we need to identify some simpler formulas, expressible in $^{\mathbb{R}}_{\mathbb{Z}}$TRIO, which can be easily derived from the TRIO axioms (1–4) assuming $\mathbb{R}$ as time domain. Thus, let us pick a sufficiently small sampling period δ, i.e., one such that $\mathrm{r_l}\delta < \mathrm{t_l} - \mathrm{l}$. Then, we derive three formulas from the axioms (1–4) that state basic properties about the invariance of the reservoir level *over a time interval of length* δ. We omit their simple proofs.

$$\mathrm{Lasts}(\mathsf{F}, \delta) \wedge \mathsf{l} \geq \mathrm{l} \;\Rightarrow\; \mathrm{Futr}(\mathsf{l} \geq \mathrm{l}, \delta) \tag{8}$$

$$\mathsf{l} \geq \mathrm{t_l} \;\Rightarrow\; \mathrm{Futr}(\mathsf{l} \geq \mathsf{l}, \delta) \tag{9}$$

$$\mathrm{Lasts}(\neg\mathsf{F} \wedge \neg\mathsf{L}, \delta) \wedge \mathsf{l} \geq \mathrm{l} \;\Rightarrow\; \mathrm{Futr}(\mathsf{l} \geq \mathrm{l}, \delta) \tag{10}$$

Now, Formulas (8–10,5,6,7) are all well-formed $^{\mathbb{R}}_{\mathbb{Z}}$TRIO formulas, and are therefore invariant under sampling and inverse sampling. Therefore, let us interpret all of them in discrete time. While Formulas (5) and (7) do not require any adaptation, the other formulas (8–10,6) must be adapted. After translating them in normal form, applying the adaptation function $\eta^{\mathbb{R}}_{\delta}\{\cdot\}$, and writing them back with derived operators, one gets the following formulas.

$$\mathrm{Lasts_{ii}}(\mathsf{F}, 1) \wedge \mathsf{l} \geq \mathrm{l} \;\Rightarrow\; \mathrm{Futr}(\mathsf{l} \geq \mathrm{l}, 1) \tag{11}$$

$$\mathsf{l} \geq \mathrm{t_l} \;\Rightarrow\; \mathrm{Futr}(\mathsf{l} \geq \mathrm{l}, 1) \tag{12}$$

$$\mathrm{Lasts_{ii}}(\neg\mathsf{F} \wedge \neg\mathsf{L}, 1) \wedge \mathsf{l} \geq \mathrm{l} \;\Rightarrow\; \mathrm{Futr}(\mathsf{l} \geq \mathrm{l}, 1) \tag{13}$$

$$\mathrm{Som}(\mathrm{AlwP_i}(\mathsf{l} \geq \mathrm{t_l})) \tag{14}$$

All in all, we have reduced the verification goal to proving that (11–13,5,14)⇒(7), in discrete time. This is a sufficiently simple task to be totally automated; for instance, we proved it using model-checking techniques for TRIO and the SPIN model checker [16]. Finally, the invariance property (7) that has been proved in discrete time, holds in continuous time as well, for all behaviors satisfying the constraint χ. A further, complementary, analysis could then consider behaviors violating the constraints χ, if they are deemed physically meaningful; we leave this (interesting) problem to future work.

5 Related Works

The problem of formally describing in a uniform manner systems that encompass both discrete-time and continuous-time modules is particularly motivated by the recently growing interest in *hybrid systems* [3]. Although this paper does not deal with hybrid systems in the sense of *hybrid automata* [2], we also allow one to describe systems where both discrete- and continuous-time dynamics occur.

However, with respect to the hybrid automata work, our approach considers descriptive models, i.e., entirely based on temporal logic formulas, which are well suited for very high-level descriptions of a system. Moreover, our idea of integration stresses separation of concerns in the development of a specification, as the modules describing different parts of the systems, that obey different models, can be developed independently and then joined together to have a global description of the system. Thus, no *ad hoc* formalism has to be introduced, since the analysis can exploit the ideas and formalisms of temporal logics.

A straightforward way to compose continuous-time and discrete-time models is to integrate discrete models into continuous ones, by introducing some suitable conventions. This is the approach followed by Fidge in [7], with reference to timed refinement calculus. The overall simplicity of the approach is probably its main strength; nonetheless we notice that integrating everything into a continuous-time setting has some disadvantages in terms of complexity, as continuous time often introduces some peculiar difficulties that render reasoning more difficult. On the contrary, our approach aims at achieving a notion of equivalence between discrete-time and continuous-time descriptions, so that one can resort to discrete time when verifying properties, while still being able to describe naturally physical processes using a full continuous-time model.

Another approach to the definition of an equivalence between continuous- and discrete-time behaviors of a specification formula is that based on the notion of *digitization*. Henzinger et al. [12] were the first to introduce and study this notion in the context of temporal logic. Digitizability is similar to our sampling invariance in that they both define a notion of invariance between discrete- and dense-time interpretations of the same formula. However, they differ in other major aspects. First, [12] compares analog- and digital-clock models, that is behaviors consisting of a *discrete* sequences of events, each carrying a timestamp: in analog-clock behaviors the timestamp is a real value, whereas in digital-clock ones it is an integer value. On the contrary, our work considers truly continuous-time behaviors and encompasses the description of dense-valued items; these features are both needed to faithfully model continuous physical quantities. Clearly, the introduction of the behavior constraint χ does limit in practice the dynamics of the items, in order to render them ultimately discretizable and thus equivalent to a discrete sequence; nonetheless, the possibility of modeling dense-valued items is unaffected, and we believe that the explicit (syntactic) introduction of the constraint χ — required to achieve invariance — permits a clearer understanding of what we "lose" exactly by discretization, and makes it possible, whenever needed, a comparison with the unconstrained continuous-time model. The other major difference between our approach and the one presented in [12] concerns the physical motivation for the notion of invariance. In fact, sampling invariance models — albeit in an abstract and idealized way — the sampling process which really occurs in systems composed by a digital controller interacting with a physical environment. On the contrary, digitization is based on the idea of "shifting" the timestamps of the analog model to make them coincide with integer values; informally, we may say that behaviors are "stretched", without changing the relative order of the events. This weaker notion allows for a neater statement of the results about digitization; however it is more oriented towards mathematical and verification results, and less to physical reasons. A formal comparison of the two notions of discretization belongs to future work (see also [4]).

We end the presentation of related works by mentioning that several different papers on discretization and continuous/discrete equivalence have exploited similar notions of invariance as that in [12]. For the sake of space we limit ourselves to citing the work by Hung and Giang [13], where a sampling semantics for Duration Calculus (DC) is defined; the paper by Ouaknine [17], where digitization for CSP is discussed; and the paper by Ouaknine and Worrell [18], which discusses the problem of digitization for timed automata, and offers several other references to related works.

6 Conclusions

This paper introduced the notion of *sampling invariance*, which is a physically motivated way to relate the continuous-time and discrete-time behaviors of a system. Sampling invariance means that a temporal logic formula can be interpreted with a continuous-time or discrete-time model consistently and in a uniform manner. Therefore, writing sampling invariant formulas permits to achieve integration of discrete-time and continuous-time formalisms in the same specification, thus being able to verify the system in the discrete-time models, while still describing naturally physical processes with the true continuity permitted by continuous-time models.

We demonstrated how to achieve sampling invariance with $^{\mathbb{R}}_{\mathbb{Z}}$TRIO, a subset of the TRIO language. The approach involved the introduction of some suitable constraints on the possible behaviors of a system, which limit the dynamics in an appropriate way, as well as simple translation rules that adapt the meaning of time units in the two time models. We are confident that the approach can be applied to other metric temporal logics as well (and MTL in particular).

Future work will follow three main directions. Firstly, we will compare the expressiveness of $^{\mathbb{R}}_{\mathbb{Z}}$TRIO with that of other formalisms for the description of real-time and hybrid systems that admit discretization under certain constraints; in particular, we will consider timed and hybrid automata [2], logics such as MITL [1], and the AASAP semantics [6]. Secondly, we will study how to possibly extend and generalize our approach, both in terms of enriching the expressiveness of the logic language, and by attempting variants of the notion of sampling invariance. Thirdly, we will try to apply our results about sampling invariance to a *refinement* approach, as we hinted at in the Introduction. This effort should be methodological as well as technical, and it will aim at tackling, from a new perspective, the important problem of refining an abstract high-level specification to an implementable low-level one.

References

1. R. Alur, T. Feder, and T. A. Henzinger. The benefits of relaxing punctuality. *JACM*, 43(1):116–146, 1996.
2. R. Alur, T. A. Henzinger, G. Lafferriere, and G. J. Pappas. Discrete abstractions of hybrid systems. In Antsaklis [3], pages 971–984.
3. P. J. Antsaklis, editor. *Special issue on hybrid systems: theory and applications*, volume 88 of *Proceedings of the IEEE*, 2000.
4. E. Asarin, O. Maler, and A. Pnueli. On discretization of delays in timed automata and digital circuits. In *Proceedings of CONCUR'98*, volume 1466 of *LNCS*, pages 470–484, 1998.
5. E. Ciapessoni, A. Coen-Porisini, E. Crivelli, D. Mandrioli, P. Mirandola, and A. Morzenti. From formal models to formally-based methods: an industrial experience. *ACM TOSEM*, 8(1):79–113, 1999.
6. M. De Wulf, L. Doyen, and J.-F. Raskin. Almost ASAP semantics. *Formal Aspects of Computing*, 17(3):319–341, 2005.
7. C. J. Fidge. Modelling discrete behaviour in a continuous-time formalism. In *Proceedings of IFM'99*, pages 170–188. Springer, 1999.
8. C. A. Furia and M. Rossi. A compositional framework for formally verifying modular systems. volume 116 of *ENTCS*, pages 185–198. Elsevier, 2004.
9. C. A. Furia and M. Rossi. When discrete met continuous. Technical Report 2005.44, DEI, Politecnico di Milano, 2005. Available from `http://www.elet.polimi.it/upload/furia/`.
10. C. Ghezzi, D. Mandrioli, and A. Morzenti. TRIO: A logic language for executable specifications of real-time systems. *JSS*, 12(2):107–123, 1990.
11. R. L. Graham, D. E. Knuth, and O. Patashnik. *Concrete Mathematics: A foundation for computer science*. Addison-Wesley, 1994.
12. T. A. Henzinger, Z. Manna, and A. Pnueli. What good are digital clocks? In *Proceedings of ICALP'92*, volume 623 of *LNCS*, pages 545–558, 1992.
13. D. V. Hung and P. H. Giang. Sampling semantics of duration calculus. In *Proceedings of FTRTFT'96*, volume 1135 of *LNCS*, pages 188–207, 1996.
14. R. Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.
15. A. Morzenti, D. Mandrioli, and C. Ghezzi. A model parametric real-time logic. *ACM TOPLAS*, 14(4):521–573, 1992.
16. A. Morzenti, M. Pradella, P. San Pietro, and P. Spoletini. Model-checking TRIO specifications in SPIN. In *Proceedings of FME'03*, volume 2805 of *LNCS*, pages 542–561, 2003.
17. J. Ouaknine. Digisation and full abstraction for dense-time model checking. In *Proceedings of TACAS'02*, volume 2280 of *LNCS*, pages 37–51, 2002.
18. J. Ouaknine and J. Worrell. Revisiting digitization, robustness, and decidability for timed automata. In *Proceedings of LICS'03*, pages 198–207. IEEE Computer Society, 2003.

On the Computational Power of Timed Differentiable Petri Nets

Serge Haddad[1], Laura Recalde[2], and Manuel Silva[2]

[1] LAMSADE-CNRS UMR 7024, University Paris-Dauphine, France
haddad@lamsade.dauphine.fr
[2] GISED, University Zaragossa, Spain
{lrecalde, silva}@unizar.es

Abstract. Well-known hierarchies discriminate between the computational power of discrete time and space dynamical systems. *A contrario* the situation is more confused for dynamical systems when time and space are continuous. A possible way to discriminate between these models is to state whether they can simulate Turing machine. For instance, it is known that continuous systems described by an ordinary differential equation (ODE) have this power. However, since the involved ODE is defined by overlapping local ODEs inside an infinite number of regions, this result has no significant application for differentiable models whose ODE is defined by an explicit representation. In this work, we considerably strengthen this result by showing that Time Differentiable Petri Nets (TDPN) can simulate Turing machines. Indeed the ODE ruling this model is expressed by an explicit linear expression enlarged with the "minimum" operator. More precisely, we present two simulations of a two counter machine by a TDPN in order to fulfill opposite requirements: robustness and boundedness. These simulations are performed by nets whose dimension of associated ODEs is constant. At last, we prove that marking coverability, submarking reachability and the existence of a steady-state are undecidable for TDPNs.

1 Introduction

Hybrid systems. Dynamic systems can be classified depending on the way time is represented. Generally, trajectories of discrete-time systems are obtained by iterating a transition function whereas the ones of continuous-time systems are often solutions of a differential equation. When a system includes both continuous and discrete transitions it is called an *hybrid system*. On the one hand, the expressive power of hybrid systems can be strictly greater than the one of Turing machines (see for instance [12]). On the other hand, in restricted models like timed automata [1], several problems including reachability can be checked in a relatively efficient way (i.e. they are $PSPACE$-complete). The frontier between decidability and undecidability in hybrid systems is still an active research topic [8,10,4,11].

Continuous systems. A special kind of hybrid systems where the trajectories are continuous (w.r.t. standard topology) and right-differentiable functions of time have been intensively studied. They are defined by a finite number regions and associated ordinary differential equations ODEs such that inside a region r, a trajectory fulfills the equation $\dot{x}_d = f_r(x)$ where x is the trajectory and $\dot{x}_d$ its right derivative. These additional requirements are not enough to limit their expressiveness. For instance, the model of [2] has piecewise constant derivatives inside regions which are polyhedra and it is Turing equivalent if its space dimension is at least 3 (see also [3,5] for additional expressiveness results).

Differentiable systems. A more stringent requirement consists in describing the dynamics of the system by a single ODE $\dot{x} = f(x)$ where f is continuous, thus yielding continuously differentiable trajectories. We call such models, *differentiable systems*. In [6], the author shows that differentiable systems in $\mathbb{R}^3$ can simulate Turing machine. The corresponding ODE is obtained by extrapolation of the transition function of the Turing machine over every possible configuration. Indeed such a configuration is represented as a point in the first dimension of the ODE (and also in the second one for technical reasons) and the third dimension corresponds to the time evolution. The explicit local ODE around every representation of a configuration is computed from this configuration and its successor by the Turing machine. Thus the explicit equations of the ODE are piecewise defined inside *an infinite number regions* which is far beyond the expressiveness of standard ODE formalisms used for the design and analysis of dynamical systems. So the question to determine which (minimal) set of operators in an explicit expression of f is required to obtain Turing machine equivalence, is still open.

Our contribution. In this work, we (partially) answer this question by showing that Time Differentiable Petri Nets (a model close to Time Continuous Petri Nets [7,13]) can simulate Turing machines. Indeed the ODE ruling this model is particularly simple. First its expression is a linear expression enlarged with the "minimum" operator. Second, it can be decomposed into a finite number of linear ODEs $\dot{x} = \mathbf{M} \cdot x$ (with $\mathbf{M}$ a matrix) inside polyhedra.

More precisely, we present two simulations of two counter machines in order to fulfill opposite requirements: robustness (allowing some perturbation of the simulation) and boundedness of the simulating net system. Our simulation is performed by a net with a constant number of places, i.e. whose dimension of its associated ODE is constant (in $(\mathbb{R}_{\geq 0})^6$ for robust simulation and in $[0, K]^{14}$ for bounded simulation). Afterwards, by modifying the simulation, we prove that marking coverability, submarking reachability and the existence of a steady-state are undecidable for (bounded) TDPNs.

Outline of the paper. In section 2, we recall notions of dynamical systems and simulations. In section 3, we introduce TDPNs. Then we design a robust simulation of counter machines in section 4 and a bounded one in section 5. Afterwards, we establish undecidability results in section 6. At last, we conclude and give some perspectives to this work.

2 Dynamical Systems and Simulation

Notations. $\mathbb{N}$ (resp. $\mathbb{R}_{\geq 0}, \mathbb{R}_{>0}$) is the set of non negative integers (resp. non negative, positive reals).

Definition 1. *A deterministic dynamical system $(X, \mathcal{T}, f)$ is defined by:*
- *a state space X, a time space $\mathcal{T}$ ($\mathcal{T}$ is either $\mathbb{N}$ or $\mathbb{R}_{\geq 0}$),*
- *a transition function f from $X \times \mathcal{T}$ to X fulfilling:*
  $$\forall x \in X, \forall \tau_1, \tau_2 \in \mathcal{T}, f(x, 0) = x \wedge f(x, \tau_1 + \tau_2) = f(f(x, \tau_1), \tau_2)$$

In the sequel, we will only deal with deterministic systems. In a *discrete* (resp. *continous*) system $X \subseteq \mathbb{N}^d$ for some d (resp. $X \subseteq (\mathbb{R}_{\geq 0})^d$) and $\mathcal{T} = \mathbb{N}$ (resp. $\mathcal{T} = \mathbb{R}_{\geq 0}$). The simulation of a discrete system by a continuous one involves a mapping from the set of states of the discrete system to the powerset of states of the continuous systems and an observation epoch. A simulation ensures that, starting from some state in the image of an initial state of the discrete system and observing the state reached after some multiple n of the epoch, one can recover the state of the discrete system after n steps. If the continuous system evolves in some bounded subset of $(\mathbb{R}_{\geq 0})^d$, the simulation is said *bounded.*

Definition 2. *A continuous dynamical system $(Y, \mathbb{R}_{\geq 0}, g)$ simulates a discrete dynamical system $(X, \mathbb{N}, f)$ if there is a mapping ϕ from X to 2^Y and $\tau_0 \in \mathbb{R}_{>0}$ such that:*
- $\forall x \neq x' \in X, \phi(x) \cap \phi(x') = \emptyset$
- $\forall x \in X, \forall y \in \phi(x), g(y, \tau_0) \in \phi(f(x, 1))$

The simulation is said bounded *if $Y \subset [0, K]^d$ for some $K \in \mathbb{R}_{\geq 0}$.*

Roughly speaking, a *robust* simulation is insensitive to small perturbations of the simulation mapping and the observation instants. In order to define robust simulation, we refine the notion of simulation. First, a *two-level* dynamical system $(Y = Y_1 \times Y_2, \mathbb{R}_{\geq 0}, g)$ is such that g is defined by g_1 from $Y_1 \times \mathbb{R}_{\geq 0}$ to Y_1 and by g_2 from $Y \times \mathbb{R}_{\geq 0}$ to Y_2 as: $g((y_1, y_2), \tau) = (g_1(y_1, \tau), g_2((y_1, y_2), \tau))$. In words, the behaviour of the first component depends only on its local state.

Definition 3. *A two-level continuous dynamical system $(Y, \mathbb{R}_{\geq 0}, g)$* consistently *simulates a discrete dynamical system $(X, \mathbb{N}, f)$ if there is $y_0 \in Y_1$, a mapping ϕ from X to 2^{Y_2} and $\tau_0 \in \mathbb{R}_{>0}$ such that:*
- $\forall x \neq x' \in X, \phi(x) \cap \phi(x') = \emptyset,$
- $g_1(y_0, \tau_0) = y_0,$
- $\forall x \in X, \forall y \in \phi(x), g_2((y_0, y), \tau_0) \in \phi(f(x, 1)).$

Note that the first part of component is a "fixed" part of the system since its whole trajectory does not depend on the input of the simulated system.

Definition 4. *A simulation (by a two-level system) is* robust *iff there exists $\delta, \epsilon \in \mathbb{R}_{>0}$ such that:*
- $\forall x \neq x' \in X, dist(\phi(x), \phi(x')) > 2\epsilon$
- $\forall x \in X, \forall y_2 \in Y_2, \forall n \in \mathbb{N}, \forall \tau \in \mathbb{R}_{\geq 0},$
  $max(dist(y_2, \phi(x)), dist(\tau, n\tau_0)) \leq \delta \Rightarrow dist(g_2((y_0, y_2), \tau), \phi(f(y, n))) \leq \epsilon$

where $dist(Y, Y') = \inf(|y - y'|_\infty \mid y \in Y, y' \in Y')$

Thus, if the simulation is robust, starting with an initial state no more pertubated than δ and delaying or anticipating the observation of the system by no more than δ, the state of the simulated system can be recovered. For obvious reasons, the simulation of an infinite-state system cannot be *simultaneously robust and bounded.*

3 Timed Differentiable Petri Nets

Notations. Let f be a *partial* mapping then $f(x) = \perp$ means that $f(x)$ is undefined. Let $\mathbf{M}$ be a matrix whose domain is $A \times B$, with $A \cap B = \emptyset$ and $a \in A$ (resp. $b \in B$) then $\mathbf{M}(a)$ (resp. $\mathbf{M}(b)$) denotes the vector corresponding to the row a (resp. the column b) of $\mathbf{M}$.

Definition 5 (Timed Differentiable Petri Nets). *A Timed Differentiable Petri Net $\mathcal{D} = \langle P, T, \mathbf{C}, \mathbf{W} \rangle$ is defined by:*
- *P, a finite set of places,*
- *T, a finite set of transitions with $P \cap T = \emptyset$,*
- *$\mathbf{C}$, the incidence matrix from $P \times T$ to $\mathbb{Z}$, we denote by ${}^\bullet t$ (resp. $t^\bullet$) the set of input places (resp. output places) of t, $\{p \mid \mathbf{C}(p,t) < 0\}$ (resp. $\{p \mid \mathbf{C}(p,t) > 0\}$). $\mathbf{C}(t)$ is called the incidence of t.*
- *$\mathbf{W}$, the speed control matrix a partial mapping from $P \times T$ to $\mathbb{R}_{>0}$ such that:*
  - $\cdot$ $\forall t \in T, \exists p \in P, \mathbf{W}(p,t) \neq \perp$
  - $\cdot$ $\forall t \in T, \forall p \in P, \mathbf{C}(p,t) < 0 \Rightarrow \mathbf{W}(p,t) \neq \perp$

A time differentiable Petri net is a Petri net enlarged with a *speed control matrix.* In a Petri net, a state $\mathbf{m}$, called a *marking*, is a positive integer vector over the set of places (i.e. an item of $\mathbb{N}^P$) where an unit is called a token. The state change is triggered by transition *firings.* In $\mathbf{m}$, the firing of a transition t with multiplicity $k \in \mathbb{N}$ yielding marking $\mathbf{m}' = \mathbf{m} + k\mathbf{C}(t)$ is only possible if $\mathbf{m}'$ is positive. Note that in Petri nets, both the choice of the transition firing and the number of simultaneous firings are non deterministic.

In a TDPN, a marking $\mathbf{m}$, is a positive real vector over the set of places (i.e. an item of $(\mathbb{R}_{\geq 0})^P$). The non determinism of Petri nets is solved by computing at any instant the instantaneous firing rate of every transition and then applying the incidence matrix in order to deduce the infinitesimal variation of the marking. The instantaneous firing rate of transitions $\mathbf{f}(\mathbf{m})(t)$ depends on the current marking *via* the speed control matrix $\mathbf{W}$: $\mathbf{f}(\mathbf{m})(t) = min(\mathbf{W}(p,t) \cdot \mathbf{m}(p) \mid \mathbf{W}(p,t) \neq \perp)$.

The first requirement about $\mathbf{W}$ ensures that the firing rate of any transition may be determined whereas the second one ensures that the marking remains non negative since any input place p of a transition t controls its firing rate.

Definition 6 (Trajectory). *Let $\mathcal{D}$ be a TDPN, then a trajectory is a* ***continuously differentiable*** *mapping* $\mathbf{m}$ *from time (i.e. $\mathbb{R}_{\geq 0}$) to the set of markings (i.e. $(\mathbb{R}_{\geq 0})^P$) which satisfies the following differential equation system:*

$$\dot{\mathbf{m}} = \mathbf{C} \cdot \mathbf{f}(\mathbf{m}) \tag{1}$$

If $\mathbf{m}(0)$ is non negative, the requirement of non negativity is a consequence of the definition of TDPNs and one can also prove (by a reduction to the linear equation case) that given an initial marking there is always a single trajectory. Equation 1 is particularly simple since it is expressed as a linear equation enlarged with the *min* operator. We introduce the concept of *configurations*: a configuration assigns to a transition, the place that controls its firing rate.

Definition 7 (Configuration). *Let $\mathcal{D}$ be a TDPN, then a configuration cf of $\mathcal{D}$ is a mapping from T to P such that $\forall t \in T, \mathbf{W}(cf(t), t) \neq \perp$. Let cf be a configuration, then $[cf]$ denotes the following polyhedron:*

$$[cf] = \{\mathbf{m} \in (\mathbb{R}_{\geq 0})^P \mid \forall t \in T, \forall p \in P, \mathbf{W}(p,t) \neq \perp \Rightarrow \mathbf{W}(p,t) \cdot \mathbf{m}(p) \geq \mathbf{W}(cf(t),t) \cdot \mathbf{m}(cf(t))\}$$

By definition, there are $\prod_{t \in T} |\{p \mid \mathbf{W}(p,t) \neq \perp\}| \leq |P|^{|T|}$ configurations. In the sequel, we use indifferently the word configuration to denote both the mapping cf and the polyhedron $[cf]$. Inside the polyhedron $[cf]$, the differential equation ruling $\mathcal{D}$ becomes linear:

$$\forall p \in P, \dot{\mathbf{m}}(p) = \sum_{t \in T} \mathbf{C}(p,t) \cdot \mathbf{W}(cf(t),t) \cdot \mathbf{m}(cf(t))$$

Graphical notations. We extend the graphical notations of Petri nets in order to take into account matrix $\mathbf{W}$. A Petri net is a bipartite graph where places are represented by circles (sometimes with their initial marking inside) and transitions by rectangles. An arc denotes a relation between a place and a transition. Note that arcs corresponding to matrix $\mathbf{C}$ are oriented whereas arcs corresponding to matrix $\mathbf{W}$ are not oriented. There are four possible patterns illustrated in figure 1. When $\mathbf{W}(p,t) = \perp \wedge \mathbf{C}(p,t) > 0$, place p receives tokens from t and does not control its firing rate. There is an oriented arc from t to p labelled by $\mathbf{C}(p,t)$. When $\mathbf{W}(p,t) \neq \perp \wedge \mathbf{C}(p,t) < 0$, place p provides tokens to t. So it *must control* its firing rate. The non oriented arc between p and t is redundant, so we will not draw it and represent only an oriented arc from p to t both labelled by $-\mathbf{C}(p,t)$ and $\mathbf{W}(p,t)$. In order to distinguish between these two labels, $\mathbf{W}(p,t)$ will always be drawn inside a box. When $\mathbf{W}(p,t) \neq \perp \wedge \mathbf{C}(p,t) > 0$, place p receives tokens from t and controls its firing rate. There is both an oriented arc from t to p and a non oriented arc between p and t with their corresponding labels. When $\mathbf{W}(p,t) \neq \perp \wedge \mathbf{C}(p,t) = 0$, place p controls the firing rate of t and t does not modify the marking of p, so there is a non oriented arc between p and t. We omit labels $\mathbf{C}(p,t)$, $-\mathbf{C}(p,t)$ and $\mathbf{W}(p,t)$ when they are equal to 1.

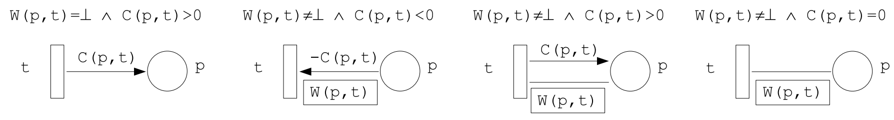


Fig. 1. Graphical notations

The net of Figure 2 illustrates TDPNs. In order to simplify the notations, when writing the differential equations, we use p as a notation for $\mathbf{m}(p)$ (the trajectory projected on p). The ODE corresponding to this net is (note that place pk holds a constant number of tokens):

$$\begin{aligned}
\dot{x}_1 &= \mathbf{f}(t_2) - \mathbf{f}(t_4) = \min\{\omega \cdot x_2, 2a\omega \cdot y_1\} - \min\{a\omega \cdot x_1, a\omega\}\\
\dot{x}_2 &= \mathbf{f}(t_1) - \mathbf{f}(t_3) = \min\{a\omega \cdot y_2, a\omega\} - \min\{2a\omega \cdot x_2, \omega \cdot x_1\}\\
\dot{y}_1 &= \mathbf{f}(t_4) - \mathbf{f}(t_2) = \min\{a\omega \cdot x_1, a\omega\} - \min\{\omega \cdot x_2, 2a\omega \cdot y_1\}\\
\dot{y}_2 &= \mathbf{f}(t_3) - \mathbf{f}(t_1) = \min\{2a\omega \cdot x_2, \omega \cdot x_1\} - \min\{a\omega \cdot y_2, a\omega\}
\end{aligned}$$

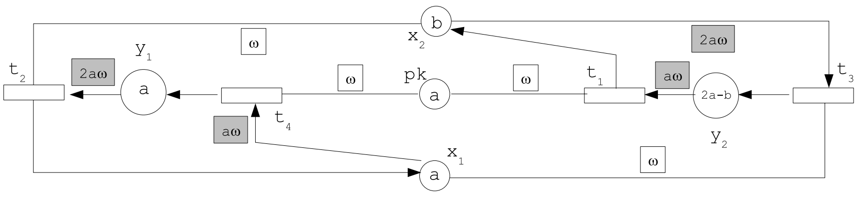


Fig. 2. A periodic TDPN

However, it can be observed that $y_1 + x_1$ and $y_2 + x_2$ are constant. Hence the system (with the initial condition described by the marking in the figure) is equivalent to:

$$\begin{aligned}
\dot{x}_1 &= \min\{\omega \cdot x_2, 2a\omega \cdot y_1\} - \min\{a\omega \cdot x_1, \omega \cdot a\},\ y_1 = 2a - x_1\\
\dot{x}_2 &= \min\{a\omega \cdot y_2, \omega \cdot a\} - \min\{2a\omega \cdot x_2, \omega \cdot x_1\},\ y_2 = 2a - x_2
\end{aligned}$$

This corresponds to a set of sixteen configurations. Let us solve the differential system with $1 \leq a \leq b \leq 2a - 1$. The linear system that applies initially is:

$$\dot{x}_1 = \omega \cdot x_2 - \omega \cdot a,\ y_1 = 2a - x_1,\ \dot{x}_2 = \omega \cdot a - \omega \cdot x_1,\ y_2 = 2a - x_2$$

In figure 2, we have represented the “inactive” items of matrix $\mathbf{W}$ in a shadowed box. In the sequel, we use this convention when it will be relevant. The solution of this system is:

$$\begin{aligned}
x_1(\tau) &= a + (b-a)\sin(\omega \cdot \tau),\ x_2(\tau) = a + (b-a)\cos(\omega \cdot \tau)\\
y_1(\tau) &= a - (b-a)\sin(\omega \cdot \tau),\ y_2(\tau) = a - (b-a)\cos(\omega \cdot \tau)
\end{aligned}$$

This trajectory stays infinitely in the initial configuration and consequently it is the behaviour of the net. Note that the dimension of the ODE may be strictly smaller than the number of places. Indeed, the existence of a linear invariant such like $\sum_{p \in P} \mathbf{m}(p) = cst$ decreases by one unit the number of dimensions. Otherwise stated, the dimension of the ODE is not $|P|$ but $\mathbf{rank}(\mathbf{C})$. Here, $|P| = 5$ and $\mathbf{rank}(\mathbf{C}) = 2$.

4 A Robust Simulation

4.1 Two Counter Machines

We will simulate two (non negative integer) counter machines (equivalent to Turing machines [9]). Their behaviour is described by a set of instructions. An instruction `I` may be one of the following kind with an obvious meaning ($\mathtt{cpt_u}$ is a counter with $u \in \{1, 2\}$):

– $\mathtt{I: goto\ I'}$;
– $\mathtt{I: increment(cpt_u); goto\ I'}$;
– $\mathtt{I: decrement(cpt_u); goto\ I'}$;
– $\mathtt{I: if\ cpt_u = 0\ then\ goto\ I'\ else\ goto\ I''}$;
– $\mathtt{I: STOP}$;

W.l.o.g. a decrementation must be preceded by a test on the counter and the (possible) successor(s) of an instruction is (are) always different from it.

4.2 Basic Principles of the Simulation

Transition pairs. In a TDPN, when a transition begins to fire, it will never stop. Thus we use *transition pairs* in order to *temporarily* either move tokens from one place to another one, or produce/consume tokens in a place.

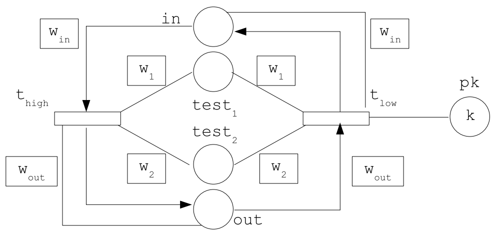


Fig. 3. A transition pair

Let us examine transitions t_{high} and t_{low} of figure 3. Their incidence is opposite. So if their firing rate is equal no marking change will occur. Let us examine $\mathbf{W}$, all the items of $\mathbf{W}(t_{high})$ and $\mathbf{W}(t_{low})$ are equal except $\mathbf{W}(pk, t_{low}) = k$ and $\mathbf{W}(pk, t_{high}) = \perp$. Thus, if any other place controls the firing rate of t_{low} it will be equal to the one of t_{high}. Place pk is a *constant* place meaning that its marking will always be k. Summarizing:

– if $w_{in}\mathbf{m}(in) > k \wedge w_{out}\mathbf{m}(out) > k \wedge w_1\mathbf{m}(test_1) > k \wedge w_2\mathbf{m}(test_2) > k$ then this pair transfers some amount of tokens from in to out,
– otherwise, there will be no marking change.

The clock subnet. The net that we build consists in two subnets: an instance of the subnet of figure 2, called in the sequel the *clock* subnet, and another subnet depending on the counter machine called the *operating* subnet. The clock subnet has k as average value, 1 as amplitude and π as period (i.e. $\omega = 2$). We recall the behavioural equations of the place markings that will be used by the operating subnet: $\mathbf{m}(x_1)(\tau) = k + sin(2\tau), \mathbf{m}(y_1)(\tau) = k - sin(2\tau)$.

Figure 4 represents the evolution of markings for x_1, y_1 and x_2 (the marking of place y_2 is symmetrical to x_2 w.r.t. the axis $\mathbf{m} = k$). Note that the mottled area is equal to 1.

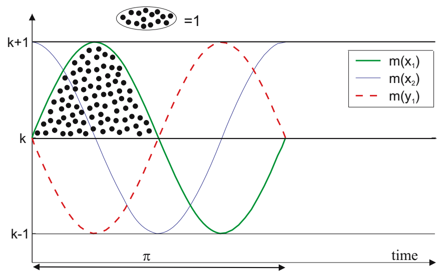


Fig. 4. The behaviour of the clock subnet

The marking changes of the operating subnet will be ruled by the places x_1 and y_1. An *execution cycle* of the net will last π. The first part of the cycle (i.e. $[h\pi, h\pi + \pi/2]$ for some $h \in \mathbb{N}$) corresponds to $\mathbf{m}(x_1) \geq k$ and the second part of the cycle (i.e. $[h\pi + \pi/2, (h+1)\pi]$) corresponds to $\mathbf{m}(y_1) \geq k$. So, the period of observation τ_0 is equal to π.

Specialisation of the transition pairs pattern
Using place x_1 (or y_1), we specialise transition pairs as illustrated in figure 5 (*pk* is the constant place of the clock subnet). In this subnet, one of the test place is x_1 and the control weights of the two test places (x_1 and *test*) are 1. First due to the periodical behaviour of $\mathbf{m}(x_1)$, no tokens transfer will occur during the second part of the cycle. Let us examine the different cases during a time interval $[h\pi, h\pi + \pi/2]$. We assume that within this interval $\mathbf{m}(test), \mathbf{m}(in)$ and $\mathbf{m}(out)$ are not modified by the other transitions.

– If $\mathbf{m}(test)(h\pi) \leq k$ then there will be no transfer of tokens.
– If $\mathbf{m}(test)(h\pi) \geq k+1 \wedge w_{in}(\mathbf{m}(in)(h\pi)-n) \geq k+1 \wedge w_{out}\mathbf{m}(out)(h\pi) \geq k+1$ then t_{high} will be controlled by x_1 and t_{low} will be controlled by *pk*. Hence (see the integral of figure 4) exactly n tokens will be transfered from *in* to *out*.
– Otherwise, some amount of tokens in $[0, n]$ will be transfered from *in* to *out*.

From a simulation point of view, one wants to avoid the last case. For the same reason, when possible, we choose w_{in} and w_{out} enough large so that it ensures that *in* and *out* will never control t_{high} and t_{low}.

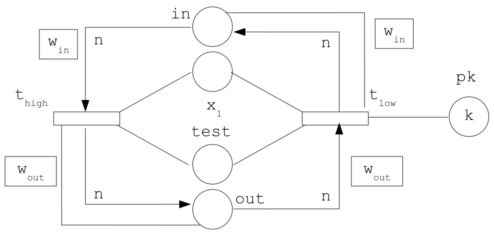


Fig. 5. A specialised transition pair

4.3 The Operating Subnet

Places of the operating subnet and the simulation mapping. Let us suppose that the counter machine has l instructions $\{\mathtt{I_1}, \ldots, \mathtt{I_l}\}$ and two counters $\{\mathtt{cpt_1}, \mathtt{cpt_2}\}$. The operating subnet has the following places: pc, qc, pn, qn, c_1, c_2. The forth first places simulate the program counter whereas the last ones simulate the counters. Furthermore by construction, the following invariants will hold for every reachable marking $\mathbf{m}$: $\mathbf{m}(pc)+\mathbf{m}(qc) = l+1$ and $\mathbf{m}(pn)+\mathbf{m}(qn) = l+1$. We now define the simulation mapping ϕ. Assume that, in a state s of the counter machine, $\mathtt{I_i}$ is the next instruction and the value of the counter $\mathtt{cpt_u}$ is v_u. Then a marking $\mathbf{m} \in \phi(s)$ iff:

- The submarking corresponding to the clock subnet is its initial marking.
- $\mathbf{m}(pn) = i$, $\mathbf{m}(qn) = l + 1 - i$,
  if $1 < i < l$ then
    $\mathbf{m}(pc) \in [i - l/k, i + l/k]$ and $\mathbf{m}(qc) \in [l + 1 - i - l/k, l + 1 - i + l/k]$
  else if $i = 1$
    $\mathbf{m}(pc) \in [1, 1 + l/k]$ and $\mathbf{m}(qc) \in [l - l/k, l]$
  else if $i = l$
    $\mathbf{m}(pc) \in [l - l/k, l]$ and $\mathbf{m}(qc) \in [1, 1 + l/k]$
- $\mathbf{m}(c_1) = k - 1 + 3v_1, \mathbf{m}(c_2) = k - 1 + 3v_2$.

Moreover, we choose $k \geq 6l^2$ for technical reasons.

Principle of the instruction simulation. The simulation of an instruction $\mathtt{I_i}$ takes exactly the time of the cycle of the clock subnet and is decomposed in two parts ($\mathbf{m}(x_1) \geq k$ followed by $\mathbf{m}(y_1) \geq k$).

The first stage is triggered by $((k + 3l)/i)\mathbf{m}(pc) \geq k + 1 \wedge ((k + 3l)/(l + 1 - i)\mathbf{m}(qc) \geq k + 1$ and performs the following tasks:

- updating $\mathbf{m}(pn)$ by producing (resp. consuming) $j - i$ tokens if $j > i$ (resp. $j < i$) where $\mathtt{I_j}$ is the next instruction; simultaneously updating $\mathbf{m}(qn)$ accordingly. If $\mathtt{I_i}$ is a conditional jump, this involves to find the appropriate j. The marking of pn will vary from i to j and the one of qn from $l + 1 - i$ to $l + 1 - j$,
- updating the counters depending on the instruction.

The second stage is triggered by $((k+3l)/j)\mathbf{m}(pn) \geq k+1 \wedge ((k+3l)/(l+1-j)\mathbf{m}(qn) \geq k + 1$ and performs the following task: updating $\mathbf{m}(pc)$ and $\mathbf{m}(qc)$ by a variable value in such a way that their marking still belong to the intervals associated with the simulation mapping.

First stage: simulation of an unconditional jump. This part of the simulation applies to both an unconditional jump, an incrementation and a decrementation. The simulation of the counter updates is straightforward once this pattern is presented. For this kind of instructions, the next instruction say $\mathtt{I_j}$ is *a priori* known.

The subnet we build depends on the relative values of i (the index of the current instruction) and j (the index of the next instruction). Here, we assume that $i < j$, the other case is similar. The transition pair of figure 6 is both triggered by pc and qc.

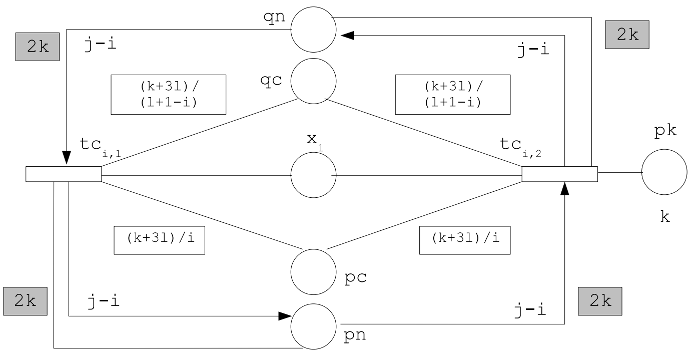


Fig. 6. First stage: simulation of an unconditional jump

– Assume that the current instruction is $\mathtt{I}_{\mathtt{i}'}$ with $i' \neq i$. If $i' < i$ then pc disables the transition pair whereas if $i' > i$ then qc disables the transition pair. We explain the first case. $\mathbf{m}(pc) \leq i' + l/k \leq i - 1 + l/k$; thus $((k+3l)/i)\mathbf{m}(pc) \leq ((k+3l)/i)(i-1+l/k) \leq k-1$ (due to our hypothesis on k).
– Assume that the current instruction is $\mathtt{I}_\mathtt{i}$. Then both $((k+3l)/i)\mathbf{m}(pc) \geq ((k+3l)/i)(i - l/k) \geq k+2$ and $((k+3l)/(l+1-i))\mathbf{m}(qc) \geq ((k+3l)/(l+1-i))((l+1-i) - l/k) \geq k+2$.

Thus in the second case, the pair is activated and transfers $j-i$ tokens from qn to pn during the first part of the cycle as required.

Note that $\mathbf{W}(pn, tc_{i,1}) = \mathbf{W}(pn, tc_{i,2}) = \mathbf{W}(qn, tc_{i,1}) = \mathbf{W}(qn, tc_{i,2}) = 2k$ ensures that places pn and qn do not control these transitions (since $2k \geq k+2$ for k enough large).

First stage: simulation of a conditional jump. The first stage for simulating the instruction $\mathtt{I}_\mathtt{i}$: `if` $\mathtt{cpt}_\mathtt{u} = 0$ `then goto` $\mathtt{I}_\mathtt{j}$ `else goto` $\mathtt{I}_{\mathtt{j}'}$; is illustrated in figure 7 in case $i < j < j'$ (the other cases are similar).

It consists in two transition pairs. Pair $tc_{i,1}, tc_{i,2}$ mimics the first stage of an unconditional jump from $\mathtt{I}_\mathtt{i}$ to $\mathtt{I}_\mathtt{j}$. It will transfer during the first part of the cycle $j-i$ tokens from qn to pn. Pair $tc_{i,3}, tc_{i,4}$ is triggered if $c_u \geq k+1$ (i.e. the counter $\mathtt{cpt}_\mathtt{u}$ is non null). If it is the case it will transfer $j'-j$ tokens from qn to pn. Thus:

– If $\mathbf{m}(c_u) = k-1$ then only the first pair is triggered and $j-i$ tokens will be transfered from qn to pn.
– otherwise $\mathbf{m}(c_u) \geq k+2$, the two pairs are simultaneously triggered and $j-i$ tokens will be transfered from qn to pn and $j'-j$ from qn to pn. Summing, $j'-i$ tokens will be transfered from qn to pn as required.

The second stage. This stage is the *difficult* part of this simulation. Due to the fact that the ODE ruling a TDPN is a linear equation inside a configuration, we cannot obtain a precise updating of $\mathbf{m}(pc)$ and $\mathbf{m}(qc)$. Roughly speaking it would require to reach a steady state in finite time which is impossible with

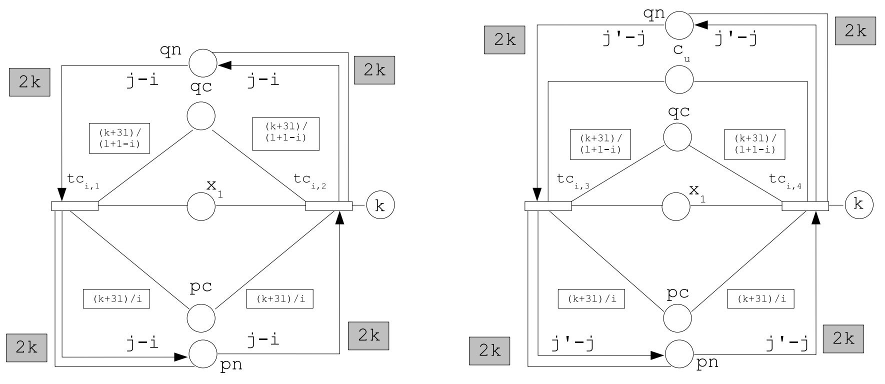


Fig. 7. The first stage of a conditional jump (places are duplicated for readability)

linear ODEs. Thus the second stage consists in trying to make the marking of pc as close as possible to j and the one of qc as close as possible to $l+1-j$.

It consists in two transition pairs depending whether the index i of the current instruction is greater or smaller than j. The first case is illustrated in figure 8 (the other case is similar).

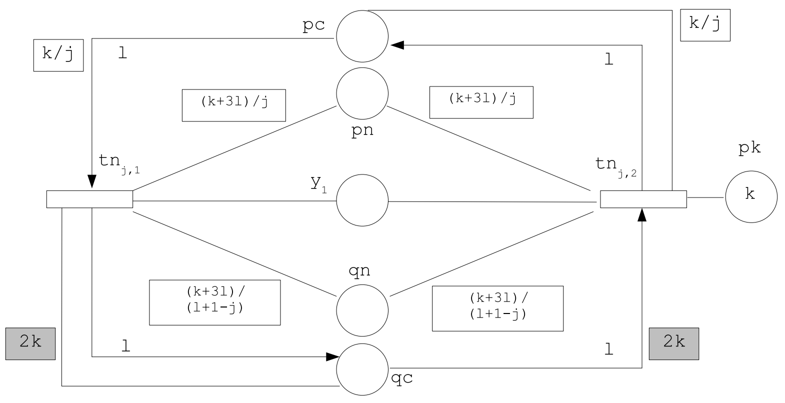


Fig. 8. The second stage

The transition pair $tn_{j,1}, tn_{j,2}$ is activated if both $\mathbf{m}(pn) = j$, $\mathbf{m}(qn) = l+1-j$ and $\mathbf{m}(pc) > j$. If the rate of transition $tn_{j,1}$ was controlled during the whole stage by y_1, pc would loose l tokens. But this means that at the beginning of the stage $\mathbf{m}(pc) > j$ and at the end $\mathbf{m}(pc) \leq j$ which is impossible since $\mathbf{m}(pc)$ must be greater than j in order to trigger the transition pair and thus cannot reach the value j (see our previous remark on linear differential equations). Thus during the second stage *pn must control* the rate of this transition. Since $\mathbf{m}(y_1) \leq k+1$, this means that, at the end of the stage, $(k/j)\mathbf{m}(pc) \leq k+1$ which implies $j \leq m(pc) \leq j+j/k \leq j+l/k$ and consequently $l+1-j-l/k \leq m(qc) \leq l+1-j$

as required by the simulation. The case $i < j$ leads, at the end of the second stage, to $j - l/k \leq m(pc) \leq j$ and consequently $l+1-j \leq m(qc) \leq l+1-j-l/k$.

Theorem 1 is a consequence of our different constructions. The dimension of the associated of ODE is obtained by recalling that the ODE of the clock subnet is 2 and that the following invariants hold in the operating subnet: $\mathbf{m}(pn) + \mathbf{m}(qn) = \mathbf{m}(pc) + \mathbf{m}(qc) = l + 1$. The proof of robustness is omitted.

Theorem 1. *Given a two counter machine $\mathcal{M}$, one can build a TDPN $\mathcal{D}$, with a constant number of places, whose size is linear w.r.t. the machine, whose associated ODE has dimension* 6 *and such that $\mathcal{D}$ robustly simulates $\mathcal{M}$.*

5 A Bounded Simulation

In this paragraph, we modify our simulation in order to obtain a bounded net. The previous net is unbounded due to the way we model the counters. So we change their management. First we will build a new lazy machine $\mathcal{M}'$ from the original one $\mathcal{M}$. We multiply by 4 the number of intructions, i.e. we create three instructions `Ai : goto Bi;`, `Bi : goto Ci;` and `Ai : goto Ii;` per instruction `Ii`. Then we modify every label in the original instructions by substituting `Ai` to `Ii`. $\mathcal{M}$ and $\mathcal{M}'$ are equivalent from a simulation point of view since they perform the same computation except that four instructions of $\mathcal{M}'$ do what does a single of instruction of $\mathcal{M}$. We then duplicate the operating subnet ($\mathcal{D}'^{(1)}$ and $\mathcal{D}'^{(2)}$) to simulate $\mathcal{M}$ *via* $\mathcal{M}'$. The only difference between the subnets is that $\mathcal{D}'^{(1)}$ simulates `Ii` of $\mathcal{M}$ by simulating `Ii` of $\mathcal{M}'$ whereas $\mathcal{N}'^{(2)}$ simulates `Ii` of $\mathcal{M}$ by simulating `Bi` of $\mathcal{M}'$. This yields a scheduling where one simulation preceedes the other one by two instructions. Superscripts $^{(1)}$ and $^{(2)}$ distinguish between places of the two subnets.

In each subnet, we add three places $pinc_u^{(s)}, pdec_u^{(s)}, d_u^{(s)}$ $(s = 1, 2)$ in addition to $c_u^{(s)}$, to manage the counter `cptu`. The marking of $pinc_u^{(s)}$ (resp. $pdec_u^{(s)}$) is equal to $k + 2$ when the current instruction is an incrementation (resp. decrementation) of `cptu` and $k - 1$ otherwise. The transition pairs managing the marking of these places are straightforward to design. Then we change our counter updates in such a way that when a counter `cptu` is equal to v then $\mathbf{m}(c_u^{(s)}) = (k+2) - 2(1/2)^v$ and $\mathbf{m}(d_u^{(s)}) = (k-1) + 2(1/2)^v$. Hence if $\texttt{cpt}_\texttt{u} = \texttt{0}$ then $\mathbf{m}(c_u^{(s)}) = k$ and if $\texttt{cpt}_\texttt{u} \geq \texttt{1}$ then $\mathbf{m}(c_u^{(s)}) \geq k + 1$ as required for the correctness of the simulation of the conditional jump.

It remains to describe the handling of incrementations and decrementations. Note that the main difficulty is that the decrement (or increment) depends on the current value of the simulated counter. If $\texttt{cpt}_\texttt{u} = \texttt{v}$ and we increment the counter, then we must produce (resp. consume) $(1/2)^v$ tokens in $c_u^{(s)}$ (resp. in $d_u^{(s)}$). If $\texttt{cpt}_\texttt{u} = \texttt{v}$ and we decrement the counter, then we must consume (resp. produce) $(1/2)^{v+1}$ tokens in $c_u^{(s)}$ (resp. in $d_u^{(s)}$). Let us observe the evolution of marking $pinc_u^{(s)}$ in the simulation (see figure 9) when one simulates the execution of instruction `Ii`, an incrementation of `cptu`. In the first part of the cycle related to $\texttt{C}_\texttt{i}$ it raises from $k-1$ to $k+2$, then holds this value during the second part and decreases in the first part of the next cycle to $k-1$. The increasing and the decreasing are not linear but they are *symmetrical*. Thus the mottled area of figure 9 is proportional to the difference between c_u and $k+2$ (equal to the difference between d_u and $k-1$). We emphasize the fact that neither the upper part of this area nor its lower part are proportional to the difference.

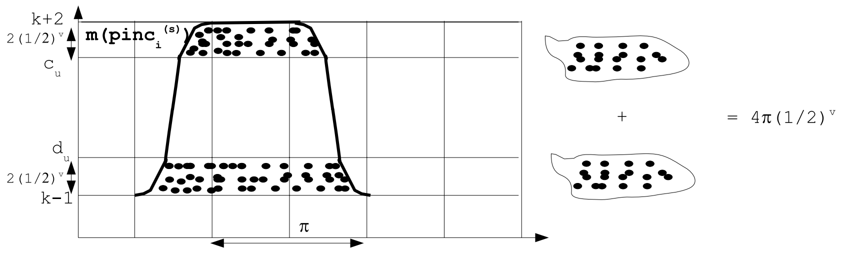


Fig. 9. A way to obtain a "proportional" increment

The subnet of figure 10 exploits this feature to simulate an incrementation of the counter. Let us detail the behaviour of this subnet. This subnet has two transition pairs $inc_{u,1}, inc_{u,2}$ and $inc_{u,3}, inc_{u,4}$. The firing rate of $inc_{u,1}$ is $(1/(4\pi))min(\mathbf{m}(c_u^{(2)}), \mathbf{m}(pinc_u^{(1)}))$ (note again that due to their speed control equal to 1, places $c_u^{(1)}$ and $c_u^{(1)}$ do not determine this rate). The rate of $inc_{u,2}$ is $(1/(4\pi))\mathbf{m}(pinc_u^{(1)})$. Thus they have different speed as long as $\mathbf{m}(pinc_u^{(1)}) > \mathbf{m}(c_u^{(2)})$. So their effect corresponds to the upper part of the mottled area of figure 9. The rate of $inc_{u,3}$ is $(1/(4\pi))min(\mathbf{m}(d_u^{(2)}), \mathbf{m}(pinc_u^{(1)}))$. The rate of $inc_{u,4}$ is $(k-1)/(4\pi)$. So their effect corresponds to the lower part of the mottled area of figure 9. The scaling factor $1/4\pi$ ensures that $1/2(\mathbf{m}(d_u^{(2)}) - (k-1))$ have been transfered from $\mathbf{m}(d_u^{(1)})$ to $\mathbf{m}(c_u^{(1)})$. The subnet managing $\mathbf{m}(d_u^{(2)})$ and $\mathbf{m}(c_u^{(2)})$ behaves similarly except that since $\mathbf{m}(c_u^{(1)})$ and $\mathbf{m}(d_u^{(1)})$ have their new value the scaling factor must be doubled in order to transfer the same amount of tokens from $\mathbf{m}(d_u^{(2)})$ to $\mathbf{m}(c_u^{(2)})$. The decrementation simulation follows a similar pattern.

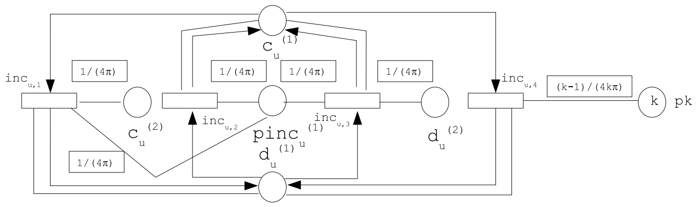


Fig. 10. Incrementing a counter (first stage)

Due to the scheduling, the places of $\mathcal{D}'^{(2)}$ modelling the counter $\mathtt{cpt_u}$ are not modified during the simulation of an instruction in $\mathcal{D}'^{(1)}$ and *vice versa.* Indeed the instruction simulations are translated and surrounded by "no-op" instructions which do not modify the counters. The correctness of this simulation yields the following theorem. The dimension of the ODE is obtained by observing that $\mathbf{m}(c_u^{(s)}) + \mathbf{m}(d_u^{(s)}) = cst$.

Theorem 2. *Given a two counter machine $\mathcal{M}$, one can build a bounded TDPN $\mathcal{D}$ with a constant number of places, whose size is linear w.r.t. the machine and whose associated ODE has dimension* 14 *such that $\mathcal{D}$ simulates $\mathcal{M}$.*

6 Undecidability Results

In this section, we apply the simulation results in order to obtain undecidability results. Proofs are omitted. Note that we cannot state the undecidability of the marking reachability problem since in the simulation, places pc and qc are not required to take precise values. However the steady-state analysis, a kind of ultimate reachability, is undecidable.

Proposition 1 (Coverability and reachability). *Let $\mathcal{D}$ be a (resp. bounded) TDPN whose associated ODE has dimension at least* 6 *(resp.* 14*), $\mathbf{m}_0, \mathbf{m}_1$ be markings, p be a place and $k \in \mathbb{N}$ then:*

– *the problem whether there is a τ such that the trajectory starting at $\mathbf{m}_0$ fulfills $\mathbf{m}(\tau)(p) = k$ is undecidable.*

– *The problem whether there is a τ such that the trajectory starting at $\mathbf{m}_0$ fulfills $\mathbf{m}(\tau)(p) \geq k$ is undecidable.*

– *The problem whether there is a τ such that the trajectory starting at $\mathbf{m}_0$ fulfills $\mathbf{m}(\tau) \geq \mathbf{m}_1$ is undecidable.*

Proposition 2 (Steady-state analysis). *Let $\mathcal{D}$ be a (resp. bounded) TDPN whose associated ODE has dimension at least* 8 *(resp.* 16*), $\mathbf{m}_0$ be a marking. Then the problem whether the trajectory $\mathbf{m}$ starting at $\mathbf{m}_0$ is such that $\lim_{\tau\to\infty} \mathbf{m}(\tau)$ exists, is undecidable.*

7 Conclusion

In this work, we have introduced TDPNs, and we have designed two simulations of counter machines in order to fulfill robustness and boundedness requirements. These simulations are performed by a net with a constant number of places, i.e. whose dimension of associated ODE is constant. We have also proved that marking coverability, submarking reachability and the existence of a steady-state are undecidable. We conjecture that the marking reachability is undecidable and we will try to prove it. In order to obtain decidability results, we also plane to introduce subclasses of TDPNs where the restrictions will be related to both the structure of the net and the associated ODE.

References

1. Rajeev Alur and David Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
2. E. Asarin, O. Maler, and A. Pnueli. Reachability analysis of dynamical systems having piecewise-constant derivatives. *Theoretical Computer Science*, 138-1:35–65, 1995.
3. Eugene Asarin and Oded Maler. Achilles and the tortoise climbing up the arithmetical hierarchy. *Journal of Computer and System Sciences*, 57(3):389–398, 1998.
4. F. Balduzzi, A. Di Febbraro, A. Giua, and C. Seatzu. Decidability results in first-order hybrid petri nets. *Discrete Event Dynamic Systems*, 11(1 and 2):41–58, 2001.
5. Olivier Bournez. Some bounds on the computational power of piecewise constant derivative systems. *Theory of Computing Systems*, 32(1):35–67, 1999.
6. Michael S. Branicky. Universal computation and other capabilities of hybrid and continuous dynamical systems. *Theoretical Computer Science*, 138(1):67–100, 1995.
7. R. David and H. Alla. *Discrete, Continuous, and Hybrid Petri Nets.* Springer-Verlag, 2004.
8. Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and Pravin Varaiya. What's decidable about hybrid automata? *Journal of Computer and System Sciences*, 57(1):94–124, 1998.
9. J.E. Hopcroft and J.D. Ullman. *Formal languages and their relation to automata.* Addison-Wesley, 1969.
10. Gerardo Lafferriere, George J. Pappas, and Sergio Yovine. Symbolic reachability computation for families of linear vector fields. *Journal of Symbolic Computation*, 32(3):231–253, 2001.
11. Venkatesh Mysore and Amir Pnueli. Refining the undecidability frontier of hybrid automata. In *FSTTCS 2005: Foundations of Software Technology and Theoretical Computer Science 25th International Conference*, volume 3821 of *LNCS*, pages 261–272, Hyderabad, India, 2005. Springer.
12. Hava T. Siegelmann and Eduardo D. Sontag. Analog computation via neural networks. *Theoretical Computer Science*, 131(2):331–360, 1994.
13. M. Silva and L. Recalde. Petri nets and integrality relaxations: A view of continuous Petri nets. *IEEE Trans. on Systems, Man, and Cybernetics*, 32(4):314–327, 2002.

Model-Checking Timed ATL for Durational Concurrent Game Structures[*]

François Laroussinie, Nicolas Markey, and Ghassan Oreiby[**]

Lab. Spécification & Vérification
ENS de Cachan & CNRS UMR 8643
61, av. Pdt. Wilson, 94235 Cachan Cedex, France
{fl, markey, oreiby}@lsv.ens-cachan.fr

Abstract. We extend the framework of ATL model-checking to "simply timed" concurrent game structures, *i.e.*, multi-agent structures where each transition carry an integral duration (or interval thereof). While the case of single durations is easily handled from the semantics point of view, intervals of durations raise several interesting questions. Moreover subtle algorithmic problems have to be handled when dealing with model checking. We propose a semantics for which we develop efficient (PTIME) algorithms for timed ATL without equality constraints, while the general case is shown to be EXPTIME-complete.

1 Introduction

Verification and model-checking. The development of embedded reactive systems is impressive (both in terms of their number and of their complexity), and their formal verification can't be ignored. Model-checking [12,7] is a well-established technique for verifying that (an automaton representing) such a system satisfies a given property. Following [21,11,22], temporal logics have been used for specifying those properties: *Linear time* temporal logics (*e.g.* LTL) expresses properties on each single execution of the model, while *branching time* temporal logics (*e.g.* CTL) deal with the computation tree of the model.

The model-checking technique has been extended to also handle quantitative measurement of time. In that framework, automata are equipped with real-valued clocks [2], and temporal logics are extended to also express quantitative constraints on the flow of time [1]. Again, this framework is now well understood, but the algorithms are noticeably more complex.

In order to lower the complexity of those algorithms, less expressive models and logics have been developed [14,9,17]. Those models are less expressive, but can be handled very efficiently, especially through symbolic model-checking algorithms using BDD techniques [8,20,9,19].

[*] This work is partly supported by ACI Sécurité & Informatique CORTOS, a program of the French ministry of research.

[**] This author is supported by a PhD grant from Région Ile-de-France.

Verification and control. In the late 80's, a new framework has been developed in the field of verification: control (and controller synthesis) [23]. The goal is now to build a controller that should prevent the (model of the) system from having unwanted behaviors.

This problem is closely related to (multi-player) games: solving such a game amounts to compute a strategy (if it exists) for a player so that he surely reaches a state where he is declared the winner. In that case, the underlying model is not a simple automaton, but rather a "concurrent game structure" (CGSs) [5], in which several agents concurrently decide on the behavior of the system. In order to reason with strategies, a new flavor of temporal logics has been defined: *alternating time* temporal logics (ATL) [4,5]. This logic allows to express, for instance, that a coalition of agents has a strategy in order to always reach a winning location, or to always avoid reaching a bad locations. When the concurrent game structure is defined explicitly, ATL enjoys polynomial-time model-checking algorithms.

Our contribution. The goal of this paper is to extend the framework of ATL to (simply) timed systems. To that aim, we introduce *durational* CGSs (DCGSs), in which each transition is labeled with an interval of possible (integer) durations. Those durations are assumed to be atomic, *i.e.*, there are no intermediate state, and the complete duration elapses in one step.

We propose a semantics for DCGSs where we assume that each transition is associated with an extra agent, who is in charge of selecting the duration of that transition within the interval it is labeled with. We believe that this semantics is really interesting, as it allows to finely select which durations can be controlled by a coalition. Moreover, we show that it still enjoys polynomial-time quantitative model-checking algorithms in the case when no equality constraint is involved.

Related work. Our discrete-time extension of CGSs to DCGSs is inspired by that of [17], where efficient quantitative model-checking algorithms are proposed. Several other extensions of games with time have been proposed in the recent literature, *e.g.* [18,3,6,10]. The semantics assumed there uses dense-time where players choose either to wait for a delay or to fire an action-transition. In [13], another dense-time semantics is proposed, working (roughly) as follows: each player chooses a (strictly positive) delay and a transition, and the game follows the player with the shortest delay. With this semantics, each player can take the others by surprise.

Those papers only deal with qualitative control objectives. In [24], Schobbens proposes a quantitative extension of ATL over timed CGSs (with a semantics of time similar to that of [13]). The resulting logic, ARTL*, a mixture of ATL and MITL, is shown decidable.

2 Definitions

2.1 Tight Durational CGS (TDCGS)

We extend the model of CGSs, see [5,16].

Definition 1. *A TDCGS is a* 6*-tuple* $\mathcal{A} = \langle \mathsf{Loc}, \mathsf{Agt}, \mathsf{AP}, \mathsf{Lab}, \mathsf{Mv}, \mathsf{Edg}\rangle$ *where:*

- Loc *is the (finite) set of* locations*;*
- $\mathsf{Agt} = \{a_1, \dots, a_k\}$ *is a (finite) set of* agents *(or* players*);*
- AP *the set of* atomic propositions*;*
- $\mathsf{Lab}\colon \mathsf{Loc} \to 2^{\mathsf{AP}}$ *the labelling function;*
- $\mathsf{Mv}\colon \mathsf{Loc} \times \mathsf{Agt} \to \mathcal{P}(\mathbb{N}) \smallsetminus \{\varnothing\}$ *gives the set of possible moves at a given location for a given agent;*
- $\mathsf{Edg}\colon \mathsf{Loc} \times \mathbb{N}^k \to \mathsf{Loc} \times \mathbb{N}^{>0}$ *is* the transition table*, that is a partial function assigning a successor location and a duration for a move of all agents.*

The difference with classical CGSs is that each transition of a TDCGS carries a positive[1] integer, representing the duration (or cost) of that transition. Given a transition $\mathsf{Edg}(q, c_1, \dots, c_k) = (q', t)$, we use $\mathsf{Edg}_\ell(q, c_1, \dots, c_k)$ (resp. $\mathsf{Edg}_\tau(q, c_1, \dots, c_k)$) to denote the location q' (resp. the duration t).

The semantics of a TDCGS is similar to that of classical CGSs: a *move* of agent a in location q is an integer c such that $c \in \mathsf{Mv}(q, a)$. Once each agent a_i has selected a move $c_i \in \mathsf{Mv}(q, a_i)$, the transition table Edg indicates the transition to be fired, namely $\mathsf{Edg}(q, c_1, \dots, c_k)$.

Definition 2. *An* execution *of a TDCGS* $\mathcal{A} = \langle \mathsf{Loc}, \mathsf{Agt}, \mathsf{AP}, \mathsf{Lab}, \mathsf{Mv}, \mathsf{Edg}\rangle$ *from a location* $q_0 \in \mathsf{Loc}$ *is an infinite sequence* $\rho = (q_0, d_0) \dots (q_i, d_i) \dots$ *such that:*

- $d_0 = 0$*;*
- *for each* i*, there exists a set of moves* $c_1^i, \dots, c_k^i$ *such that*

$$(q_{i+1}, d_{i+1} - d_i) = \mathsf{Edg}(q_i, c_1^i, \dots, c_k^i).$$

For an execution $\rho = (q_0, d_0) \dots (q_i, d_i) \dots$, the integer d_i is the *date* when arriving in q_i.

The interesting point with ATL, compared to standard temporal logics, is that it allows quantifications on *strategies* of (coalitions of) agents. A coalition is a subset of the set of agents. Now we introduce the notions of strategy and outcome:

Definition 3. *Let* $\mathcal{A} = \langle \mathsf{Loc}, \mathsf{Agt}, \mathsf{AP}, \mathsf{Lab}, \mathsf{Mv}, \mathsf{Edg}\rangle$ *be a TDCGS.*

- *Let* $a \in \mathsf{Agt}$*. A* strategy σ_a *for* a *is a mapping that associates, with any finite prefix* $(q_0, d_0) \dots (q_i, d_i)$ *of any execution, a possible move for agent* a *in* q_i*, i.e.* $\sigma_a((q_0, d_0) \dots (q_i, d_i)) \in \mathsf{Mv}(q_i, a)$*.*
- *Let* $A \subseteq \mathsf{Agt}$ *be a coalition. A move for* A *from a location* q *is a family* $(c_a)_{a \in A}$ *: one move for each agent in* A*. We write* $\mathsf{Mv}(q, A)$ *to represent the set of all possible moves for* A *from* q*. Moreover a strategy* σ_A *for* A *is a family* $(\sigma_a)_{a \in A}$*.*
  We write $\overline{A}$ *for* $\mathsf{Agt} \smallsetminus A$*. Given a move* $c \in \mathsf{Mv}(q, A)$ *and* $\overline{c} \in \mathsf{Mv}(q, \overline{A})$*, we write* $\mathsf{Edg}(q, c \cdot \overline{c})$ *for the transition corresponding to these choices.*

[1] We require in this paper that the durations be non-zero. The case of zero-durations makes some of our algorithms slightly more difficult, and will be handled in a long version of this paper.

- *Let $A \subseteq \mathsf{Agt}$ be a coalition and σ_A be a strategy for A. An execution $\rho = (q_0, d_0) \ldots (q_i, d_i) \ldots$ is an* outcome *of σ_A from q_0 if, for any i, writing $c^i = \sigma_A((q_0, d_0) \ldots (q_i, d_i))$, there exists $\overline{c}^i \in \mathsf{Mv}(q_i, \overline{A})$ s.t.*

$$(q_{i+1}, d_{i+1} - d_i) = \mathsf{Edg}(q_i, c \cdot \overline{c}).$$

We denote by $Out^{\mathcal{A}}(q, \sigma_A)$ the set of all outcomes of σ_A from q (we omit the superscript $\mathcal{A}$ when it is clear from the context).

Size of a TDCGS. The size $|\mathcal{A}|$ of $\mathcal{A}$ is the sum of the sizes of Loc and Edg. The size of Edg is defined as follows: $|\mathsf{Edg}| = \sum_{q \in \mathsf{Loc}} \sum_{c \in \mathsf{Mv}(q,\mathsf{Agt})} (1 + \lfloor \log(\mathsf{Edg}_\tau(q, c) \rfloor)$.

2.2 Timed ATL (TATL)

Definition 4. *The syntax of* TATL *is defined by the following grammar:*

$$\begin{aligned} \mathsf{TATL} \ni \varphi_s, \psi_s &::= \top \mid P \mid \neg \varphi_s \mid \varphi_s \vee \psi_s \mid \langle\!\langle A \rangle\!\rangle\, \varphi_p \\ \varphi_p &::= \mathbf{X}\, \varphi_s \mid \varphi_s\, \mathbf{U}_{\sim\zeta}\, \psi_s \mid \varphi_s\, \mathbf{R}_{\sim\zeta}\, \psi_s \end{aligned}$$

with $P \in \mathsf{AP}$, $A \subseteq \mathsf{Agt}$, $\sim \in \{<, \leq, =, \geq, >\}$, and $\zeta \in \mathbb{N}$.

We also define the usual shorthands, such as $\bot \stackrel{\text{def}}{\equiv} \neg\top$, $\langle\!\langle A \rangle\!\rangle\, \mathbf{F}_{\sim\zeta}\, \varphi_s \stackrel{\text{def}}{\equiv} \langle\!\langle A \rangle\!\rangle\, \top\, \mathbf{U}_{\sim\zeta}\, \varphi_s$, and $\langle\!\langle A \rangle\!\rangle\, \mathbf{G}_{\sim\zeta}\, \varphi_s \stackrel{\text{def}}{\equiv} \langle\!\langle A \rangle\!\rangle\, \bot\, \mathbf{R}_{\sim\zeta}\, \varphi_s$.

TATL formulae are interpreted over states of TDCGSs. Intuitively, the state-formula $\langle\!\langle A \rangle\!\rangle\, \varphi_p$ holds in q iff there exists a strategy for coalition A in order to enforce the path-formula φ_p along all the outcomes. Formally:

Definition 5. *The following clauses define when a location q (resp. an execution $\rho = (q_0, d_0)(q_1, d_1) \ldots$) of a TDCGS $\mathcal{A}$ satisfies a* TATL *formula φ_s (resp. a path-formula φ_p), written $q \models_{\mathcal{A}} \varphi_s$ (resp. $\rho \models_{\mathcal{A}} \varphi_p$), by induction over the formula (semantics of boolean operators and atomic propositions are omitted):*

$$\begin{aligned} q \models_{\mathcal{A}} \langle\!\langle A \rangle\!\rangle\, \varphi_p \quad &\Leftrightarrow \quad \exists \sigma_A.\ \forall \rho \in Out(q, \sigma_A).\ \rho \models_{\mathcal{A}} \varphi_p \\ \rho \models_{\mathcal{A}} \mathbf{X}\, \varphi_s \quad &\Leftrightarrow \quad q_1 \models_{\mathcal{A}} \varphi_s \\ \rho \models_{\mathcal{A}} \varphi_s\, \mathbf{U}_{\sim\zeta}\, \psi_s \quad &\Leftrightarrow \quad \exists i \in \mathbb{N}.\ q_i \models_{\mathcal{A}} \psi_s,\ d_i \sim \zeta \\ &\qquad\qquad \textit{and } q_j \models_{\mathcal{A}} \varphi_s \textit{ for any } 0 \leq j < i \\ \rho \models_{\mathcal{A}} \varphi_s\, \mathbf{R}_{\sim\zeta}\, \psi_s \quad &\Leftrightarrow \quad \rho \models_{\mathcal{A}} \neg(\neg\varphi_s\, \mathbf{U}_{\sim\zeta}\, \neg\psi_s) \end{aligned}$$

When $\mathcal{A}$ is clear from the context, we just write $q \models \varphi$. Note that, contrary to usual definitions of ATL [4,5], we include the "release" modality $\mathbf{R}$, as we proved in [16] that modality $\langle\!\langle A \rangle\!\rangle\mathbf{R}$ cannot be expressed using only $\langle\!\langle A \rangle\!\rangle\mathbf{U}$ and $\langle\!\langle A \rangle\!\rangle\mathbf{G}$. Intuitively, $\varphi_s\, \mathbf{R}_{\sim\zeta}\, \psi_s$ requires that ψ_s must hold when the condition "$\sim \zeta$" is fulfilled, but this global requirement is released as soon as φ_s holds. Formally:

$$\begin{aligned} \rho \models_{\mathcal{A}} \varphi_s\, \mathbf{R}_{\sim\zeta}\, \psi_s \quad \Leftrightarrow \quad &\forall i \in \mathbb{N}.\ (d_i \sim \zeta \Rightarrow q_i \models_{\mathcal{A}} \psi_s) \\ &\quad \text{or } \exists j < i.\ q_j \models_{\mathcal{A}} \varphi_s. \end{aligned}$$

We use $\mathsf{TATL}_{\leq,\geq}$ to denote the fragment of TATL where subscripts "$= \zeta$" are not allowed in timing constraints for $\mathbf{U}$ and $\mathbf{R}$.

3 Model Checking TATL

The complexity of model-checking an ATL formula over a CGS has been shown to be linear in both the size of the structure and the size of the formula [5]. On the other hand, TCTL model checking on DKSs is PTIME-complete when timing constraints contain no equality, while it is Δ_2^{P}-complete otherwise [17].

We present in this section our model-checking algorithm for TATL over TD-CGSs. We explain how to handle modalities $\mathbf{U}_{\sim\zeta}$ and $\mathbf{R}_{\sim\zeta}$. We then gather up those algorithms in a global labeling algorithm, which is PTIME-complete when no equality constraint is involved, and EXPTIME-complete otherwise.

3.1 Modalities $\mathbf{U}_{\leq\zeta}$ and $\mathbf{R}_{\leq\zeta}$

First we consider the case of formula $\langle\!\langle A\rangle\!\rangle\, \varphi_1\, \mathbf{U}_{\leq\zeta}\, \varphi_2$, that is, when the coalition A aims at reaching φ_2 within ζ time units (and verifying φ_1 in the intermediate states). We assume that states have already been labeled with φ_1 and φ_2, which can therefore be seen as atomic propositions. We have:

Lemma 6. *Let $\mathcal{A}$ be a TDCGS, and $\varphi = \langle\!\langle A\rangle\!\rangle\, P_1\, \mathbf{U}_{\leq\zeta}\, P_2$ be a* TATL *formula (with $P_1, P_2 \in \mathsf{AP}$). Then we can compute in time $O(|\mathsf{Loc}| \cdot |\mathsf{Edg}|)$ the set of locations of $\mathcal{A}$ where φ holds.*

Proof. For this proof, we define the extra modality $\mathbf{U}^{\leq i}_{\leq n}$, with the following semantics:

$$\rho \models_{\mathcal{A}} P_1\, \mathbf{U}^{\leq i}_{\leq n}\, P_2 \quad\Leftrightarrow\quad \exists j \leq i.\ q_j \models_{\mathcal{A}} P_1,\ d_j \leq n, \\ \text{and } q_k \models_{\mathcal{A}} P_2 \text{ for any } 0 \leq k < j$$

This modality requires that the right-hand side formula be satisfied within at most i steps. It is clear that, for any $n \in \mathbb{N}$,

$$q \models \langle\!\langle A\rangle\!\rangle\, P_1\, \mathbf{U}_{\leq n}\, P_2 \quad\Leftrightarrow\quad q \models \langle\!\langle A\rangle\!\rangle\, P_1\, \mathbf{U}^{\leq|\mathsf{Loc}|}_{\leq n}\, P_2. \tag{1}$$

Indeed, if all the outcomes of a strategy satisfy $P_1\, \mathbf{U}_{\leq n}\, P_2$, it is possible to adapt that strategy so that each location is visited at most once along each outcome.

We now define functions $v_i(q)$, for $i \in \mathbb{N}$ and $q \in \mathsf{Loc}$, by the following recursive rules:

$$\begin{cases} \text{if } q \models P_2 : \quad v_0(q) = 0 \\ \text{if } q \models \neg P_2 : v_0(q) = +\infty \end{cases}$$

$$\begin{cases} \text{if } q \models P_2 :\ v_{i+1}(q) = 0 \\ \text{if } q \models \neg P_1 \wedge \neg P_2 :\ v_{i+1}(q) = +\infty \\ \text{otherwise} : v_{i+1}(q) = \min\limits_{c \in \mathsf{Mv}(q,A)}\ \max\limits_{\overline{c} \in \mathsf{Mv}(q,\overline{A})} \Big(\mathsf{Edg}_\tau(q, c\cdot\overline{c}) + v_i(\mathsf{Edg}_\ell(q, c\cdot\overline{c}))\Big) \end{cases}$$

Our proof now amounts to showing the following lemma:

Lemma 7. *For any $i \in \mathbb{N}$, for any $n \in \mathbb{N}$ and any $q \in \mathsf{Loc}$, we have:*

$$n \geq v_i(q) \quad\Leftrightarrow\quad q \models \langle\!\langle A\rangle\!\rangle\, P_1\, \mathbf{U}^{\leq i}_{\leq n}\, P_2.$$

The proof is by induction on i: The base case is straightforward, as well as the induction step when $q \models P_2$ and when $q \models \neg P_1 \wedge \neg P_2$. We thus only focus on the last case, when $q \models P_1 \wedge \neg P_2$: Assume the induction hypothesis holds up to level i. If $n \geq v_{i+1}(q)$, then (by definition of $v_{i+1}(q)$) there exists a move $c \in \mathsf{Mv}(q, A)$ such that, for any move $\overline{c} \in \mathsf{Mv}(q, \overline{A})$, we have

$$n \geq \mathsf{Edg}_\tau(q, c \cdot \overline{c}) + v_i(\mathsf{Edg}_\ell(q, c \cdot \overline{c})).$$

By i.h., for any $\overline{c} \in \mathsf{Mv}(q, \overline{A})$, we have

$$\mathsf{Edg}_\ell(q, c \cdot \overline{c}) \models \langle\!\langle A \rangle\!\rangle\, P_1 \,\mathbf{U}^{\leq i}_{\leq n - \mathsf{Edg}_\tau(q, c\cdot\overline{c})}\, P_2.$$

The strategy σ_A witnessing that property, combined with the move $c \in \mathsf{Mv}(q, A)$, yields a strategy for enforcing $P_1 \,\mathbf{U}^{\leq i+1}_{\leq n}\, P_2$ from q, as required.

The converse implication follows the same lines: given a strategy σ_A enforcing $P_1 \,\mathbf{U}^{\leq i+1}_{\leq n}\, P_2$ from q, we let $c = \sigma_A(q)$, and deduce that n satisfies the same inequalities as above.

From Equation (1), it suffices to compute $v_{|\mathsf{Loc}|}(q)$, for each $q \in \mathsf{Loc}$, to deduce the set of locations where φ holds. This algorithm thus runs in time $O(|\mathsf{Loc}| \cdot |\mathsf{Edg}|)$. □

The release modality is handled similarly: we define $v'_i(q)$ as follows:

$$\begin{cases} \text{if } q \models \neg P_2 : v'_0(q) = 0 \\ \text{if } q \models P_2 : \quad v'_0(q) = +\infty \end{cases}$$

$$\begin{cases} \text{if } q \models \neg P_2 : v'_{i+1}(q) = 0 \\ \text{if } q \models P_1 \wedge P_2 : \ v'_{i+1}(q) = +\infty \\ \text{otherwise} : \ v'_{i+1}(q) = \max\limits_{c \in \mathsf{Mv}(q,A)} \ \min\limits_{\overline{c} \in \mathsf{Mv}(q,\overline{A})} \Big(\mathsf{Edg}_\tau(q, c \cdot \overline{c}) + v'_i(\mathsf{Edg}_\ell(q, c\cdot\overline{c}))\Big) \end{cases}$$

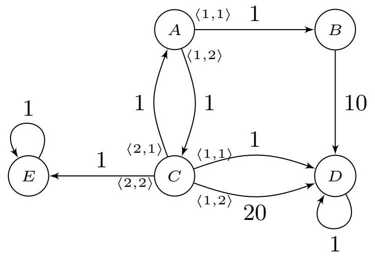
and $\mathbf{R}^{\leq i}_{\leq n}$ as the dual of $\mathbf{U}^{\leq i}_{\leq n}$. Then an equivalence similar to Equation (1) holds, and we have the following lemma (proof omitted):

Lemma 8. *For any $i \in \mathbb{N}$, for any $n \in \mathbb{N}$ and any $q \in \mathsf{Loc}$, we have:*

$$n < v'_i(q) \quad \Leftrightarrow \quad q \models \langle\!\langle A \rangle\!\rangle\, P_1 \,\mathbf{R}^{\leq i}_{\leq n}\, P_2.$$

Example 1. Consider the example depicted on Figure 1. On that TDCGS, the duration is the integer written in the middle of each transition. The tuples that are written close to the source location indicates the choices of the agents for firing that transition (for instance, $\langle 2, 1\rangle$ means that player a_1 chooses move 2 and player a_2 chooses move 1). They are omitted when each agent has a single choice.

The valuations of atomic propositions are given in the table on the right of the figure. This table shows the computation of $v_i(q)$, for each location. This computation converges in three steps. For instance, that $v_3(A) = 21$ indicates that $A \models \langle\!\langle a_1 \rangle\!\rangle\, P_1 \,\mathbf{U}_{\leq 21}\, P_2$ holds, but $A \not\models \langle\!\langle a_1 \rangle\!\rangle\, P_1 \,\mathbf{U}_{\leq 20}\, P_2$.



$v_i(q)$	0	1	2	3
$A\ (P_1,\ \neg P_2)$	$+\infty$	$+\infty$	21	21
$B\ (P_1,\ P_2)$	0	0	0	0
$C\ (P_1,\ \neg P_2)$	$+\infty$	20	20	20
$D\ (\neg P_1,\ P_2)$	0	0	0	0
$E\ (P_1,\ \neg P_2)$	$+\infty$	$+\infty$	$+\infty$	$+\infty$

Fig. 1. The algorithm for $\mathbf{U}_{\leq\zeta}$

3.2 Modalities $\mathbf{U}_{\geq\zeta}$ and $\mathbf{R}_{\geq\zeta}$

We now consider formula $\langle\!\langle A\rangle\!\rangle\, \varphi_1\, \mathbf{U}_{\geq\zeta}\, \varphi_2$ expressing that coalition A has a strategy for staying at least ζ time units in φ_1-states before reaching φ_2. We have:

Lemma 9. *Let $\mathcal{A}$ be a TDCGS, and $\varphi = \langle\!\langle A\rangle\!\rangle\, P_1\, \mathbf{U}_{\geq\zeta}\, P_2$ be a* TATL *formula (with $P_1, P_2 \in \mathsf{AP}$). Then we can compute in time $O(|\mathsf{Loc}| \cdot |\mathsf{Edg}|)$ the set of locations of $\mathcal{A}$ where φ holds.*

Proof. The idea is similar to that of the proof of Lemma 6. We introduce the following modality:

$$\begin{aligned}\rho \models_{\mathcal{A}} p\, \mathbf{U}^{\geq i}\, q \quad \Leftrightarrow \quad & \exists j \geq i.\ q_j \models_{\mathcal{A}} q \\ & \text{and } q_k \models_{\mathcal{A}} p \text{ for any } 0 \leq k < j\end{aligned}$$

We then compute a sequence of values, defined by the following recursive rules:

$$\begin{cases} \text{if } q \models \neg \langle\!\langle A\rangle\!\rangle\, P_1\, \mathbf{U}\, P_2 :\ v_0(q) = -\infty \\ \text{if } q \models \langle\!\langle A\rangle\!\rangle\, P_1\, \mathbf{U}\, P_2 \wedge \neg \langle\!\langle A\rangle\!\rangle\, P_1\, \mathbf{U}^{\geq 1}\, P_2 :\ v_0(q) = 0 \\ \text{if } q \models \langle\!\langle A\rangle\!\rangle\, P_1\, \mathbf{U}^{\geq 1}\, P_2 : v_0(q) = +\infty \end{cases}$$

$$\begin{cases} \text{if } q \models \neg \langle\!\langle A\rangle\!\rangle\, P_1\, \mathbf{U}\, P_2 :\ v_{i+1}(q) = -\infty \\ \text{if } q \models \langle\!\langle A\rangle\!\rangle\, P_1\, \mathbf{U}\, P_2 \wedge \neg \langle\!\langle A\rangle\!\rangle\, P_1\, \mathbf{U}^{\geq 1}\, P_2 :\ v_{i+1}(q) = 0 \\ \text{if } q \models \langle\!\langle A\rangle\!\rangle\, P_1\, \mathbf{U}^{\geq 1}\, P_2 : \\ \qquad\qquad v_{i+1}(q) = \max\limits_{c \in \mathsf{Mv}(q,A)}\ \min\limits_{\overline{c} \in \mathsf{Mv}(q,\overline{A})} \Big(\mathsf{Edg}_\tau(q, c\cdot\overline{c}) + v_i(\mathsf{Edg}_\ell(q, c\cdot\overline{c}))\Big) \end{cases}$$

This computation requires that we first compute the set of locations satisfying $\langle\!\langle A\rangle\!\rangle\, P_1\, \mathbf{U}\, P_2$ and those satisfying $\langle\!\langle A\rangle\!\rangle\, P_1\, \mathbf{U}^{\geq 1}\, P_2$. This can be done in time $O(|\mathsf{Edg}|)$ using standard ATL model-checking algorithms. Thus, computing $v_i(q)$ for each $q \in \mathsf{Loc}$ and each $i \leq |\mathsf{Loc}|$ can be achieved in time $O(|\mathsf{Loc}| \cdot |\mathsf{Edg}|)$.

Those values satisfy the following lemma:

Lemma 10. *For any $i \in \mathbb{N}$, for any $n \in \mathbb{N}$ and $q \in \mathsf{Loc}$, we have*

$$n \leq v_i(q) \quad \Leftrightarrow \quad q \models \langle\!\langle A\rangle\!\rangle \big[(P_1\, \mathbf{U}_{\geq n}\, P_2) \vee (P_1\, \mathbf{U}^{\geq i+1}\, P_2)\big].$$

This will conclude the proof of Lemma 9, thanks to the following equivalence:

$$q \models \langle\langle A\rangle\rangle\, P_1\, \mathbf{U}_{\geq n}\, P_2 \;\Leftrightarrow\; q \models \langle\langle A\rangle\rangle\, \big[(P_1\, \mathbf{U}_{\geq n}\, P_2) \vee (P_1\, \mathbf{U}^{\geq |\mathsf{Loc}|+1}\, P_2)\big].$$

This equivalence relies on the fact that all durations are strictly positive: if some outcome of the strategy satisfies $P_1\, \mathbf{U}^{\geq |\mathsf{Loc}|+1}\, P_2$, then one location is visited twice, and it is possible to adapt the strategy so that it is visited n times (thus with total duration larger than n) before visiting P_2.

We now prove Lemma 10: we omit the easy cases, and only focus on the inductive step in the case when $q \models \langle\langle A\rangle\rangle\, P_1\, \mathbf{U}^{\geq 1}\, P_2$. In particular, we have $q \models P_1$. First, pick some $n \leq v_{i+1}(q)$, assuming the result holds up to step i. By definition of $v_{i+1}(q)$, there exists $c \in \mathsf{Mv}(q, A)$ such that, for any $\overline{c} \in \mathsf{Mv}(q, \overline{A})$, we have

$$n \leq \mathsf{Edg}_\tau(q, c \cdot \overline{c}) + v_i(\mathsf{Edg}_\ell(q, c \cdot \overline{c})).$$

From the induction hypothesis, this means that

$$\mathsf{Edg}_\ell(q, c \cdot \overline{c}) \models \langle\langle A\rangle\rangle\, \big[(P_1\, \mathbf{U}_{\geq n - \mathsf{Edg}_\tau(q, c\cdot\overline{c})}\, P_2) \vee (P_1\, \mathbf{U}^{\geq i+1}\, P_2)\big].$$

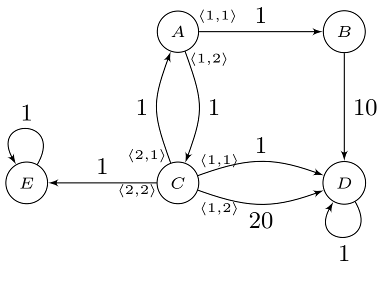
The strategy witnessing that property, combined with move $c \in \mathsf{Mv}(q, A)$, yields a strategy witnessing that

$$q \models \langle\langle A\rangle\rangle\, \big[(P_1\, \mathbf{U}_{\geq n}\, P_2) \vee (P_1\, \mathbf{U}^{\geq i+2}\, P_2)\big].$$

The converse implication follows the same (reversed) steps. □

We omit the case of the release modality, which is very similar.

Example 2. In Figure 2, we apply that algorithm to the same TDCGS as before. This table shows the computation of $v_i(q)$, for each location. This computation converges in three steps. For instance, that $v_3(A) = 2$ indicates that $A \models \langle\langle a_1\rangle\rangle\, P_1\, \mathbf{U}_{\geq 2}\, P_2$ holds, but $A \not\models \langle\langle a_1\rangle\rangle\, P_1\, \mathbf{U}_{\geq 3}\, P_2$.



$v_i(q)$	0	1	2	3
A (P_1, $\neg P_2$)	$+\infty$	$+\infty$	2	2
B (P_1, P_2)	$+\infty$	10	10	10
C (P_1, $\neg P_2$)	$+\infty$	1	1	1
D ($\neg P_1$, P_2)	0	0	0	0
E (P_1, $\neg P_2$)	$-\infty$	$-\infty$	$-\infty$	$-\infty$

Fig. 2. The algorithm for $\mathbf{U}_{\geq \zeta}$

3.3 Modalities $\mathbf{U}_{=\zeta}$ and $\mathbf{R}_{=\zeta}$

Lemma 11. *Let $\mathcal{A}$ be a TDCGS, and $\varphi = \langle\!\langle A \rangle\!\rangle\, P_1 \,\mathbf{U}_{=\zeta}\, P_2$ be a* TATL *formula. We can compute in time $O(\zeta \cdot |\mathsf{Edg}|)$ the set of locations of $\mathcal{A}$ where φ holds.*

Since ζ is encoded in binary, this algorithm runs in time exponential in the size of the formula.

Proof. We use dynamical programming, and recursively build a table $T\colon \mathsf{Loc} \times \{0, \ldots, \zeta\} \to \{\top, \bot\}$ such that

$$T(q,i) = \top \quad \Leftrightarrow \quad q \models_{\mathcal{A}} \langle\!\langle A \rangle\!\rangle\, \varphi_1 \,\mathbf{U}_{=i}\, \varphi_2. \tag{2}$$

When $i = 0$, letting $T(q, 0) = \top$ if, and only if, $q \models P_2$ clearly fulfills equation (2) (since all durations are non-zero). Now, pick $i < \zeta$, and assume all the $T(q, j)$ have been computed for $j \leq i$. Then

$$\begin{aligned} T(q, i+1) = \top \quad \Leftrightarrow \quad & q \models P_1 \text{ and } \exists c \in \mathsf{Mv}(q, A).\ \forall \overline{c} \in \mathsf{Mv}(q, \overline{A}). \\ & \qquad \mathsf{Edg}(q, c \cdot \overline{c}) = (q', t) \text{ with } T(q', i - t) = \top. \end{aligned}$$

This computation can be done since all durations are non-zero. It is achieved by running through the transition table, and is thus in time linear in the size of Edg. It is clear that equation (2) is preserved by this construction, so that in the end, $q \models \varphi$ iff $T(q, \zeta) = \top$. This algorithm runs in time $O(\zeta \times |\mathsf{Edg}|)$. □

A similar algorithm can be defined for handling $\langle\!\langle A \rangle\!\rangle\, \varphi_1 \,\mathbf{R}_{=\zeta}\, \varphi_2$: the table $T'(q, i)$ is initialized in the same way (i.e., $T'(q, 0) = \top \ \Leftrightarrow\ q \models P_2$), and each step is computed according to the following rule:

$$\begin{aligned} T'(q, i) = \top \quad \Leftrightarrow \quad & q \models P_1 \vee \exists c \in \mathsf{Mv}(q, A).\ \forall \overline{c} \in \mathsf{Mv}(q, \overline{A}). \\ & \qquad \mathsf{Edg}(q, c \cdot \overline{c}) = (q', t) \text{ with } T'(q', i - t) = \top. \end{aligned}$$

3.4 Results for TATL and $\mathsf{TATL}_{\leq,\geq}$

From Lemmas 6, 9 and 11, we can deduce procedures to handle all modalities and this gives the following result:

Theorem 12. *Model checking a* TATL *formula φ over a TDCGS $\mathcal{A}$ can be achieved in time $O(|\mathcal{A}|^2 \cdot |\varphi| \cdot \zeta_{\max})$, where $\zeta_{\max}$ is the maximal constant appearing in φ. It is thus in* EXPTIME.

Note that this algorithm is polynomial in the size of the TDCGS, and is exponential only because of the binary encoding of the constants that appear in the formula. This complexity blow-up cannot be avoided:

Theorem 13. *Model-checking* TATL *over TDCGSs is* EXPTIME*-complete.*

Proof. This result is based on the fact that deciding the *countdown games* is EXPTIME-hard [15]. A countdown game is a two-player game. It consists of a weighted graph (V, E) where V is the set of vertices and $E \subseteq V \times \mathbb{N} \times V$ is the weighted transition relation. A configuration of a countdown game (V, E) is a pair $(v, C) \in V \times \mathbb{N}$. At every turn, Player 1 chooses, from the current configuration (v, C), a duration $1 \leq d \leq C$ s.t. (1) $0 < d \leq C$ and (2) there exist at least one transition $(v, d, v') \in E$. Then Player 2 chooses one of these transitions (issued from v and whose duration is d) and then the new configuration is $(v', C-d)$. Any configuration $(v, 0)$ is terminal and it is a winning configuration for Player 1. Any configuration (v, C) s.t. (1) $C > 0$ and (2) there is no transition $(v, d, -)$ with $d \leq C$ is terminal and it is a winning configuration for Player 2. Deciding whether Player 1 has a wining strategy for a configuration (v, C) is an EXPTIME-hard problem [15].

We can easily build a TDCGS $\mathcal{A} = \langle \mathsf{Loc}, \mathsf{Agt}, \mathsf{AP}, \mathsf{Lab}, \mathsf{Mv}, \mathsf{Edg} \rangle$ corresponding to the countdown game (V, E). We let $\mathsf{Agt} = \{a_1, a_2\}$. The set of locations $\mathsf{Loc} \subseteq V \cup V \times \mathbb{N}$ is defined as follows: $v \in \mathsf{Loc}$ if $v \in V$, and $(v, t) \in \mathsf{Loc}$ if there exists a transition $(v, t, -)$ in E. This is a turn-based game: agent a_1 plays (i.e., has several possible moves) in locations v, and agent a_2 plays in locations (v, t). From v, the moves of agent a_1 are the weights t s.t. there exist some transitions $(v, t, -)$ in E, and the moves of agent a_2 from (v, t) are the possible successors v' s.t. $(v, t, v') \in E$. The transition table is defined in a natural way and the duration associated with transitions leading from v to (v, t) or from (v, t) to v' is t. Then deciding whether the configuration (v, C) with $C \in \mathbb{N}$ is winning for Player 1 reduces to a model checking problem: $v \models_{\mathcal{A}} \langle\!\langle a_1 \rangle\!\rangle \mathbf{F}_{=2C} \top$. □

Still, we can have efficient algorithm if we restrict to the fragment $\mathsf{TATL}_{\leq,\geq}$:

Theorem 14. *Model-checking a* $\mathsf{TATL}_{\leq,\geq}$ *formula* φ *over a TDCGS* $\mathcal{A}$ *can be achieved in time* $O(|\mathcal{A}|^2 \cdot |\varphi|)$*, and is thus in* PTIME.

This is an immediate consequence of Lemmas 6 and 9. PTIME-hardness follows from that of CTL model-checking over Kripke structures. Thus:

Corollary 15. *Model checking* $\mathsf{TATL}_{\leq,\geq}$ *over TDCGSs is* PTIME*-complete.*

3.5 Unitary TDCGSs

Unitary TDCGSs are TDCGSs where all durations equal 1. Intuitively, unitary TDCGSs are easier to handle because it is possible to verify $\varphi = \langle\!\langle A \rangle\!\rangle \varphi_1 \mathbf{U}_{=c} \varphi_2$ by dichotomy, i.e., by verifying $\langle\!\langle A \rangle\!\rangle \varphi_1 \mathbf{U}_{=\lfloor c/2 \rfloor} (\langle\!\langle A \rangle\!\rangle \varphi_1 \mathbf{U}_{=c-\lfloor c/2 \rfloor} \varphi_2)$, and so on [14]. That way, model-checking an $\mathbf{U}_{=\zeta}$ formula can be achieved in time $O(\log(\zeta) \cdot |\mathcal{A}|)$. In the end:

Theorem 16. *Model checking a* TATL *formula* φ *over a unitary TDCGS* $\mathcal{A}$ *can be achieved in time* $O(|\mathcal{A}|^2 \cdot |\varphi|)$*, and is thus* PTIME*-complete.*

4 Durational CGSs

We now propose an extension of the models above that now allows transitions labeled with intervals (instead of a single integer). That way, agents don't know in advance the duration of the transitions. Special agents, one per transition, decide for the durations. This gives a very expressive framework, in which coalitions can mix "classical" agents with "time" agents.

4.1 Definition

We write $\mathcal{I}$ for the set of intervals with bounds in $\mathbb{N}^{>0} \cup \{+\infty\}$.

Definition 17. *A durational CGS (DCGS) is a* 6*-tuple* $\mathcal{S} = \langle \mathsf{Loc}, \mathsf{Agt}, \mathsf{AP}, \mathsf{Lab}, \mathsf{Mv}, \mathsf{Edg} \rangle$ *such that:*

- Loc*,* Agt*,* AP*,* Lab *and* Mv *have the same characteristics as in Definition 1;*
- $\mathsf{Edg}\colon \mathsf{Loc} \times \mathbb{N}^k \to \mathsf{Loc} \times \mathcal{I}$ *is the transition table, associating with each transition an interval containing its possible durations.*

The size of the transition table is again the space needed to write it in binary notation, and the size of a DCGS is $|\mathsf{Loc}| + |\mathsf{Edg}|$. Again, we use $\mathsf{Edg}_\tau(q, c)$ to denote the interval of durations of the transition $\mathsf{Edg}(q, c)$, and $\mathsf{Edg}_\ell(q, c)$ to denote its location.

While the syntax is not very different to that of TDCGSs, the semantics is rather more involved: the crucial point is that the agents must select the transition the system will fire, but they must also choose the duration of that transition within the interval it's labeled with. This last part is achieved by special "time-agents": we consider one time-agent $ta_{q,c}$ per location q and move c in $\mathsf{Mv}(q, \mathsf{Agt})$. Formally the semantics of $\mathcal{S}$ is defined as a TDCGS $\mathcal{A}[\mathcal{S}] = \langle \mathsf{Loc}, \mathsf{Agt}', \mathsf{AP}, \mathsf{Lab}, \mathsf{Mv}', \mathsf{Edg}' \rangle$ with:

- $\mathsf{Agt}' = \mathsf{Agt} \cup \{ta_{q,c} \mid q \in \mathsf{Loc} \text{ and } c \in \mathsf{Mv}(q, \mathsf{Agt})\}$,
- $\mathsf{Mv}'(q, a) = \mathsf{Mv}(q, a)$ for any $a \in \mathsf{Agt}$; $\mathsf{Mv}'(q, ta_{q,c}) = \mathsf{Edg}_\tau(q, c)$ for any $ta_{q,c} \in \mathsf{Agt}'$, and $\mathsf{Mv}'(q, ta_{q',c}) = \{0\}$ for any $ta_{q,c} \in \mathsf{Agt}'$ with $q' \neq q$,
- $\mathsf{Edg}'(q, c, t_{q_0,c_0}, \ldots, t_{q_n,c_m}) = (q', t)$ iff $c \in \mathsf{Mv}'(q, \mathsf{Agt})$ for any i, $t = t_{q,c}$ and $t_{q,c} \in \mathsf{Mv}'(q, ta_{q,c})$, and $t_{q',c'} = 0$ when $q' \neq q$ or $c \neq c'$.

As for TDCGSs, we use the notions of coalition of agents (including the time-agents), strategy and outcome.

Note that the transition table of the corresponding TDCGS $\mathcal{A}[\mathcal{S}]$ is infinite when there exist infinite intervals in the definition of $\mathcal{S}$. And when it is finite, $|\mathsf{Edg}'|$ is bounded by $|\mathsf{Edg}| \cdot b_M$ where b_M is the maximal constant occurring in the intervals of durations in $\mathcal{S}$: in Edg', we replace each entry (q, c) of Edg by $(b - a) + 1$ entries when $\mathsf{Edg}_\tau(q, c) = [a; b]$. Thus the size of $\mathcal{A}[\mathcal{S}]$ is potentially exponential in $|\mathcal{S}|$ (due to the binary encoding). Note that in every entry of Edg', only one time-agent may have more than one possible move.

Moreover time agents can be used in TATL modalities in order to express the existence of strategies for coalitions that may control the duration of a subset of transitions. Given a DCGS $\mathcal{S}$, a location q, and a TATL formula φ, we write $q \models_{\mathcal{S}} \varphi$ when $q \models_{\mathcal{A}[\mathcal{S}]} \varphi$. We might omit the subscript if it raises no ambiguity.

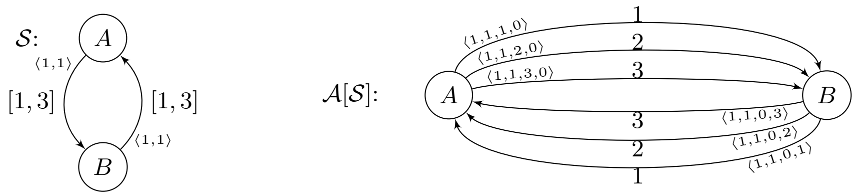


Fig. 3. A DCGS for the simplified Nim game, and its associated TDCGS

Example 3. We illustrate our models by a simple example: the simplified Nim game. In that game, a set of N matches are aligned on a table, and each player, in turn, picks between 1 and 3 matches. The players who takes the last match is declared the winner. This game can easily be encoded as the DCGS (where "durations" are in fact the number of matches taken by the players) depicted on the left of Figure 3. Player A wins iff formula $\langle\!\langle A, t_A \rangle\!\rangle\, \mathbf{F}_{=N}\, B$ holds.

Time agents. The motivation for using time-agents is that the time elapsing should not be controlled by the same player along an execution. Depending on the state or the transition, it is convenient to be able to specify who decides the duration of an event. Note that assigning one time-agent per transition is more general than assigning one time-agent per location: indeed, in the former approach, a time-agent ta_q (for controlling the duration of all the transitions issued from q) can be easily simulated by the coalition $\{ta_{q,c_1}, \ldots, ta_{q,c_m}\}$ containing all the time-agents of the transitions leaving q.

4.2 Model Checking $\mathsf{TATL}_{\le,\ge}$

When verifying TATL formulae containing modalities with timing constraints of the form "$\le c$" or "$\ge c$", we do not need to consider all the transitions of $\mathcal{A}[\mathcal{S}]$. We can restrict the analysis to an abstraction of $\mathcal{A}[\mathcal{S}]$:

Definition 18. *Let* $\mathcal{S} = \langle \mathsf{Loc}, \mathsf{Agt}, \mathsf{AP}, \mathsf{Lab}, \mathsf{Mv}, \mathsf{Edg}\rangle$ *be a DCGS and let* $\mathcal{A}[\mathcal{S}] = \langle \mathsf{Loc}, \mathsf{Agt}', \mathsf{AP}, \mathsf{Lab}, \mathsf{Mv}', \mathsf{Edg}'\rangle$ *be the TDCGS corresponding to the semantics of* $\mathcal{S}$*. Given an integer* C*, we define the* C*-abstraction of* $\mathcal{S}$ *as the TDCGS* $\mathcal{A}[\mathcal{S}]_C = \langle \mathsf{Loc}, \mathsf{Agt}', \mathsf{AP}, \mathsf{Lab}, \mathsf{Mv}'', \mathsf{Edg}''\rangle$ *with:*

- $\mathsf{Mv}''(q, ta_{q,c})$ *is* $\{a, b\}$ *(resp.* $\{a; C+1\}$*) if* $\mathsf{Edg}_\tau(q, c) = [a, b]$ *(resp.* $\mathsf{Edg}_\tau(q, c) = [a, +\infty)$*); and* Mv'' *coincides with* Mv' *for other cases (*$a \in \mathsf{Agt}$ *or* $ta_{q',c}$ *with* $q' \neq q$*).*
- Edg'' *is defined as* Edg' *but with* Mv'' *instead of* Mv'*.*

In the TDCGS $\mathcal{A}[\mathcal{S}]_C$, we replace the set of transitions corresponding to all durations in an interval λ by two transitions: a short one —the left-end value of λ— and a long one: either the right-end value of λ if λ is finite, or $C + 1$ (or the left-end of λ if $C + 1 \notin \lambda$). Indeed, an open interval is interesting for the truth value of some properties because it may allow arbitrary long durations.

But delaying for $C+1$ t.u. is always enough when considering $\mathsf{TATL}_{\leq,\geq}$ formulae with constants less than C:

Lemma 19. *Let $\mathcal{S}$ be the DCGS and C be an integer. For any $\zeta \leq C$ and $q \in \mathsf{Loc}$, we have:*

$$q \models_{\mathcal{A}[\mathcal{S}]} \langle\!\langle A \rangle\!\rangle\, P_1 \,\mathbf{U}_{\leq\zeta}\, P_2 \quad \Leftrightarrow \quad q \models_{\mathcal{A}[\mathcal{S}]_C} \langle\!\langle A \rangle\!\rangle\, P_1 \,\mathbf{U}_{\leq\zeta}\, P_2 \tag{3}$$
$$q \models_{\mathcal{A}[\mathcal{S}]} \langle\!\langle A \rangle\!\rangle\, P_1 \,\mathbf{U}_{\geq\zeta}\, P_2 \quad \Leftrightarrow \quad q \models_{\mathcal{A}[\mathcal{S}]_C} \langle\!\langle A \rangle\!\rangle\, P_1 \,\mathbf{U}_{\geq\zeta}\, P_2 \tag{4}$$
$$q \models_{\mathcal{A}[\mathcal{S}]} \langle\!\langle A \rangle\!\rangle\, P_1 \,\mathbf{R}_{\leq\zeta}\, P_2 \quad \Leftrightarrow \quad q \models_{\mathcal{A}[\mathcal{S}]_C} \langle\!\langle A \rangle\!\rangle\, P_1 \,\mathbf{R}_{\leq\zeta}\, P_2 \tag{5}$$
$$q \models_{\mathcal{A}[\mathcal{S}]} \langle\!\langle A \rangle\!\rangle\, P_1 \,\mathbf{R}_{\geq\zeta}\, P_2 \quad \Leftrightarrow \quad q \models_{\mathcal{A}[\mathcal{S}]_C} \langle\!\langle A \rangle\!\rangle\, P_1 \,\mathbf{R}_{\geq\zeta}\, P_2 \tag{6}$$

Proof (sketch). There are more behaviors in $\mathcal{A}[\mathcal{S}]$ than in $\mathcal{A}[\mathcal{S}]_C$, but these additional executions do not change the truth value of $\mathsf{TATL}_{\leq,\geq}$ formulae.

Indeed let $\rho = (q_0, d_0) \ldots (q_i, d_i) \ldots$ be an execution in $\mathcal{A}[\mathcal{S}]$ and let c_i^a (resp. c_i^t) be the $i+1$-st move of the agents Agt (resp. the time-agents) along ρ [2]. We can change the move of the agent ta_{q_i,c_i} and obtain another run ρ' with the *same prefix and the same suffix* as ρ: the duration spent in q_i has changed (and thus the global dates of actions) but not the time spent in other locations. This property allows to make local changes on delays without changing the sequence of visited states.

Consider a strategy σ_A for the coalition A in $\mathcal{A}[\mathcal{S}]$ to ensure $\psi = \mathsf{P}_1 \,\mathbf{U}_{\leq\zeta}\, P_2$. From σ_A, we can build a strategy σ'_A ensuring ψ in $\mathcal{A}[\mathcal{S}]_C$. Indeed the only changes we have to make are for the moves of time-agents when, in q with a move c for Agt, σ_A requires to wait for $t_{q,c}$ while $t_{q,c}$ is not in the restricted set of moves of $\mathcal{A}[\mathcal{S}]_C$ (*i.e.* $t_{q,c} \notin \mathsf{Mv}''(q, ta_{q,c})$). In that case, the strategy σ'_A can propose the minimal duration in $\mathsf{Mv}''(q, ta_{q,c})$: the ending state satisfying ψ will be reached sooner than along ρ and then ψ will be true. If the formula to be verified was $\mathbf{U}_{\geq c}$ then the maximal duration will be enough to ensure the formula. The same holds for the release operators.

Now consider a strategy σ'_A for the coalition A in $\mathcal{A}[\mathcal{S}]_C$ to ensure $\psi = \mathsf{P}_1 \,\mathbf{U}_{\leq\zeta}\, P_2$. This strategy can be completed for ensuring ψ in the full TDCGS. Consider a finite execution ρ in $\mathcal{A}[\mathcal{S}]$, we can define a corresponding execution $\bar{\rho}$ in $\mathcal{A}[\mathcal{S}]_C$ where any move of time-agent $ta_{q,c}$ from the location q is either left unchanged if $ta_{q,c} \in A$ —this is an "A-controllable" time-agent and having applied σ_A from the beginning of the execution ensures that its move is in $\mathcal{A}[\mathcal{S}]_C$—, or replaced by the maximal duration in $\mathsf{Mv}''(q, ta_{q,c})$ if $ta_{q,c}$ is not an A-controllable time-agent. Then it is sufficient to define $\sigma_A(\rho)$ as $\sigma'_A(\bar{\rho})$.

Of course, if we consider $\psi = \mathsf{P}_1 \,\mathbf{U}_{\leq\zeta}\, P_2$, we build $\bar{\rho}$ differently and consider minimal durations. □

Note that the size of $\mathcal{A}[\mathcal{S}]_C$ is bounded by $2 \cdot |\mathcal{S}|$. Thus:

Theorem 20. *Model checking $\mathsf{TATL}_{\leq,\geq}$ over DCGSs is PTIME-complete.*

[2] *i.e.* $(q_{i+1}, t_{i+1} - t_i) = \mathsf{Edg}'(q_i, c_i^a \cdot c_i^t)$ with $c_i^a \cdot c_i^t \in \mathsf{Mv}(q_i, \mathsf{Agt}')$.

4.3 Model Checking TATL

For full TATL, we also reduce the problem to that of finite TDCGSs. Given a DCGS $\mathcal{S}$ and a formula $\varphi = \langle\!\langle A \rangle\!\rangle\, P_1 \,\mathbf{U}_{=\zeta}\, P_2$, we explicitly add one extra agent per transition, with moves in $[a, b]$ if the corresponding transition is labeled with $[a, b]$, and moves in $[a, \max(a, \zeta + 1)]$ if it is labeled with $[a, +\infty)$. This restriction makes the corresponding TDCGS to be finite, its size is in $O(|\mathcal{S}| \cdot \max(b_M, \zeta))$ where b_M is the maximal integer appearing as a bound of an interval in $\mathcal{S}$. And applying the algorithm of Theorem 12 to that TDCGS, we get an EXPTIME algorithm for model-checking φ on our DCGS $\mathcal{S}$. The algorithm is similar for the release modality. This must be repeated a polynomial number of times for verifying a TATL formula, yielding an algorithm in time $O(|\mathcal{S}|^2 \cdot b_M \cdot \zeta_{\max} \cdot |\varphi|)$ (where $\zeta_{\max}$ is the largest constant in the formula φ), that is, in time exponential in both the structure and the formula. Note that the blow-up is only due to the binary notation for the integers. Combined to Theorem 13, this gives:

Theorem 21. *Model-checking* TATL *over DCGSs is* EXPTIME-*complete.*

5 Conclusion

We have introduced a new family of models of concurrent game structures, in which transitions carry a (set of) durations. The semantics of those models involve "time-agents", i.e., agents that decide the duration of the transition that will be fired. This allows to break the symmetry in time games, allowing some coalition to decide the duration of *some* of the transitions.

We proved that those models enjoy efficient quantitative model-checking algorithms, as soon as no timing constraint $= \zeta$ is involved. Equality constraints yield an exponential blow-up, which we saw cannot be avoided.

As future work, we plan to extend this "time agent"-semantics to concurrent game structures with clocks, similar to timed automata [2]. As a first step, it would be nice if we could extend the algorithms presented here to the "continuous" semantics (as defined in [17]) of DCGSs.

References

1. R. Alur, C. Courcoubetis, and D. L. Dill. Model-checking in dense real-time. *Information & Computation*, 104(1):2–34, 1993.
2. R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
3. R. Alur and T. A. Henzinger. Modularity for timed and hybrid systems. In *Proc. 8th Intl Conf. on Concurrency Theory (CONCUR'97)*, volume 1243 of *LNCS*, pages 74–88. Springer, 1997.
4. R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. In *Proc 38th Annual Symp. on Foundations of Computer Science (FOCS'97)*, pages 100–109. IEEE Comp. Soc. Press, 1997.
5. R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *Journal of the ACM*, 49(5):672–713, 2002.

6. E. Asarin, O. Maler, A. Pnueli, and J. Sifakis. Controller synthesis for timed automata. In *Proc IFAC Symp. on System Structure and Control*. Elsevier, 1998.
7. B. Bérard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, and Ph. Schnoebelen. *Systems and Software Verification. Model-Checking Techniques and Tools*. Springer, 2001.
8. R. E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992.
9. S. Campos and E. M. Clarke. Real-time symbolic model checking for discrete time models. In *Theories and Experiences for Real-Time System Development*, volume 2 of *AMAST Series in Computing*, pages 129–145. World Scientific, 1995.
10. F. Cassez, A. David, E. Fleury, K. G. Larsen, and D. Lime. Efficient on-the-fly algorithms for the analysis of timed games. In M. Abadi and L. de Alfaro, editors, *Proc. 16th Intl Conf. on Concurrency Theory (CONCUR'05)*, volume 3653 of *LNCS*, pages 66–80. Springer, Aug. 2005.
11. E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proc. Workshop Logics of Programs 1981*, volume 131 of *LNCS*, pages 52–71. Springer, 1981.
12. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 1999.
13. L. de Alfaro, M. Faella, T. A. Henzinger, R. Majumdar, and M. Stoelinga. The element of surprise in timed games. In R. Amadio and D. Lugiez, editors, *Proc. 14th Intl Conf. on Concurrency Theory (CONCUR'03)*, volume 2761 of *LNCS*, pages 142–156. Springer, Aug. 2003.
14. E. A. Emerson, A. K.-L. Mok, A. P. Sistla, and J. Srinivasan. Quantitative temporal reasoning. *Real-Time Systems*, 4:331–352, 1992.
15. M. Jurdziński. Countdown games, Mar. 2006. Personal communication.
16. F. Laroussinie, N. Markey, and G. Oreiby. Expressiveness and complexity of ATL. Research Report LSV-06-03, Laboratoire Spécification et Vérification, ENS Cachan, France, Feb. 2006.
17. F. Laroussinie, N. Markey, and Ph. Schnoebelen. Efficient timed model checking for discrete-time systems. *Theoretical Computer Science*, 353(1-3):249–271, 2006.
18. O. Maler, A. Pnueli, and J. Sifakis. On the synthesis of discrete controllers for timed systems. In *Proc. 12th Annual Symp. on Theoretical Aspects of Computer Science (STACS'95)*, volume 900 of *LNCS*, pages 229–242. Springer, 1995.
19. N. Markey and Ph. Schnoebelen. Symbolic model checking of simply-timed systems. In Y. Lakhnech and S. Yovine, editors, *Proc. Joint Conf. Formal Modelling and Analysis of Timed Systems (FORMATS'04) and Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'04)*, volume 3253 of *Lecture Notes in Computer Science*, pages 102–117, Grenoble, France, Sept. 2004. Springer.
20. K. L. McMillan. *Symbolic Model Checking: An Approach to the State Explosion Problem*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 1992.
21. A. Pnueli. The temporal logic of programs. In *Proc. 18th IEEE Symp. Foundations of Computer Science (FOCS'77)*, pages 46–57. IEEE Comp. Soc. Press, 1977.
22. J.-P. Queille and J. Sifakis. Specification and verification of concurrent systems in CESAR. In *Proc. 5th Intl Symp. on Programming*, volume 137 of *LNCS*, pages 337–351. Springer, 1982.
23. P. J. Ramadge and W. M. Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77(1):81–98, 1989.
24. P.-Y. Schobbens and Y. Bontemps. Real-time concurrent game structures, Dec. 2005. Personal communication.

A Dose of Timed Logic, in Guarded Measure

Kamal Lodaya[1] and Paritosh K. Pandya[2]

[1] The Institute of Mathematical Sciences
CIT Campus, Chennai 600113, India
[2] Tata Institute of Fundamental Research
Colaba, Mumbai 400005, India

Abstract. We consider interval measurement logic *IML*, a sublogic of Zhou and Hansen's interval logic, with measurement functions which provide real-valued measurement of some aspect of system behaviour in a given time interval. We interpret *IML* over a variety of time domains (continuous, sampled, integer) and show that it can provide a unified treatment of many diverse temporal logics including duration calculus (DC), interval duration logic (IDL) and metric temporal logic (MTL). We introduce a fragment *GIML* with restricted measurement modalities which subsumes most of the decidable timed logics considered in the literature.

Next, we introduce a guarded first-order logic with measurements *MGF*. As a generalisation of Kamp's theorem, we show that over arbitrary time domains, the measurement logic *GIML* is expressively complete for it. We also show that *MGF* has the 3-variable property.

In addition, we have a preliminary result showing the decidability of a subset of *GIML* when interpreted over timed words.

The importance of reasoning about timed systems has led to considerable research on models and logics for timed behaviours. We consider a slightly more general situation where, in addition to time, we can use other measurement functions as well. For instance, instead of saying "during the last 24 hours, the rainfall was 100 mm," we can say that "the time elapsed for the last 100 mm of rainfall was over 4 months." We can also have measurements of quantities like "mean value" of a proposition within a time interval. Guelev has shown how probabilities might be incorporated into such a framework [Gue00].

Unlike data languages [BPT03], there is no finite state mechanism associated with the measurement functions. Thus we are in the setting of the interval logic with measurements defined by Zhou and Hansen [ZH04].

There exists quite a menagerie of timed and duration logics. In Section 1 below, we review the literature and define our logic $\chi IML[\Sigma]$ over a signature Σ of measurement functions, and parameterised by a set of primitive comparisons χ dependent on Σ. We show that it can provide a unified treatment of many diverse temporal logics including duration calculus (DC), interval duration logic (IDL) and metric temporal logic (MTL).

In Section 2, we consider an enrichment of Kamp's $FO[<]$ with measurements. The undecidability of this logic motivates us to formulate and investigate a fragment $\chi MGF[<, \Sigma]$ with χ-guarded measurement quantifiers. The next two sections show that $\chi GIML$ is expressively complete for χMGF. Kamp's syntactic techniques were used by Venema [Ven91], and we extend these as well as the pebble games of Immerman and Kozen [IK87] in our proofs. As in Kamp's result, we show along the way that χMGF has the three-variable property.

Thus the expressiveness of our logic is reasonably delineated. We would have liked to have established a connection to aperiodic languages [Bac03] but this must remain future work.

We now turn to decidability. We find that *IML* and *GIML* are in general undecidable, but for a set of weak comparisons (which disallow equality tests between measurements and constants), we use a result by Hirshfeld and Rabinovich [HR99] and our expressive completeness to show that *Weak*-$GIML[\ell]$ is decidable for continous time. We also prove by translation into one-clock alternating timed automata [LW05, OW05], decidability over timed words of a sublogic *Punct-FgIML*$[\ell]$ of $GIML[\ell]$, which only has nesting-free forward guarding.

1 A Classification of Timed Behaviours and Logics

Timed logics describe the evolution of system behaviour in time. For us, time is a linear order $(T, <)$, and we will further assume that T is a subset of the non-negative reals (which we designate $\Re$) with $<$ the usual ordering. $Intv(T)$, the set of (closed) intervals of T, is $\{[b,e] \in T \times T \mid b \leq e\}$. A time frame $TF = (T, <, d)$ is a subset of the real order $(\Re, <)$ with d giving the absolute value of the distance on the real line between two real numbers, i.e. $d[b,e] = |b - e|$.

Zhou and Hansen have proposed an interesting interval logic [ZH04] where the variables (measurement functions) denote real-valued measurements of system behaviour in a given time interval. Formally we have a signature $\Sigma = \{m_1, \ldots, m_n\}$ of measurement function symbols (of arity 2), and we assume that it contains the distinguished function ℓ which measures the length of the interval. We will often abbreviate the signature $\{\ell\}$ to ℓ.

Zhou and Hansen's logic allows first order real arithmetic over such measurements. In this section, we introduce a restricted version of this logic where a measurement may only be compared with an integer constant. We call this logic **interval measurement logic**, *IML*.

Let $Pvar$ be a finite set of propositional variables. A behaviour of a system over TF is a pair of maps $\theta : (Pvar \to T \to \{0,1\}) \times (\Sigma \to Intv \to \Re)$, where Σ might depend on $Pvar$. For convenience we write $\theta(p)$ as a boolean function of time and $\theta(m)[b,e]$, for $m \in \Sigma$, as giving the value of the measurement function m on the interval $[b,e]$. Moreover, we require that the measurement ℓ is always interpreted as length of the interval, i.e. $\theta(\ell)[b,e] = d[b,e] = |b - e|$. An interval model is a pair $\theta, [b,e]$.

It is useful to consider several classes of time frames $TF = (T, <, d)$ where $T \subseteq \Re$. In the literature, we find a variety of timed logics which use these different classes as time frames. Some such logics are listed in the next section.

Continuous infinite time. $T = \Re$.

Continuous time finitely variable behaviours. We call a continuous time behaviour θ finitely variable if for any p in $Pvar$, $\theta(p)$ changes only finitely often within a finite interval.

Continuous time prefix behaviours. $T = [0, r]$ for some $r \in \Re$ where $[0, r]$ denotes the set of reals between 0 and r. Let $max(T) = r$ give the maximum time-point upto which the behaviour is captured.

Sampled time infinite behaviours. T has the form $\{r_0, r_1, \ldots\}$, the countably infinite set of sampling points where $r_0 = 0$ and $r_0, r_1, \ldots$ forms an unbounded increasing sequence within $\Re$. These behaviours are also called timed ω-words.

Sampled time prefix behaviours. T has the form $\{r_0, r_1, \ldots, r_k\}$, the finite set of sampling points where $r_0 = 0$ and $r_i \in \Re$. Also $r_i < r_{i+1}$. Let $max(T) = r_k$. These behaviours are also called finite timed words.

Discrete time. This is a subclass of sampled time behaviour (infinite or prefix) where all sampling points have integer values.

1.1 Interval Measurement Logic

The formulae of interval measurement logic $\chi IML[\Sigma]$ are parameterised by a set χ of atomic measurement comparisons. For concreteness, let us fix $Punct(\Sigma)$ to be the countable set of comparisons $m \sim c$, for all $m \in \Sigma$, $\sim$ in $\{<, =, >\}$ and c in $\mathbf{Z}$, the set of integers. Since punctuality is a strong requirement [AFH96], we also define $Weak(\Sigma)$ to be the subset of weak comparisons made only using the $<$ symbol, and $Test(\Sigma)$ to be the set of comparisons of the form $m = 0$.

Boolean combinations of the propositional variables $Pvar$ and $0, 1$ (denoting *false* and *true* respectively) are called propositions, $Prop$. Let P, Q range over propositions, $m \sim c$ over comparisons from a set χ and D_1, D_2 over formulae. The formulae of $\chi IML[\Sigma]$ have the syntax

$$\lceil P \rceil^0 \ \mid \ \lceil P \rceil \ \mid \ m \sim c \ \mid \ D_1 \frown D_2 \ \mid \ D_1 \overset{\frown}{+} D_2 \ \mid \ D_1 \overset{\frown}{-} D_2 \ \mid \ D_1 \wedge D_2 \ \mid \ \neg D$$

When we write $IML[\Sigma]$ we mean that χ is the full set of comparisons $Punct(\Sigma)$.

Semantics of IML. For a proposition P and time point t, $\theta, t \models P$ is defined inductively as usual. Let $\theta, [b, e] \models D$ denote that the formula $D \in IML$ evaluates to true in the behaviour θ at interval $[b, e] \in Intv(\theta)$. Omitting the boolean cases, this is defined as follows.

$\theta, [b, e] \models \lceil P \rceil^0$ iff $b = e$ and $\theta, b \models P$
$\theta, [b, e] \models \lceil P \rceil$ iff $b < e$ and for all $t: b < t < e.\ \theta, t \models P$
$\theta, [b, e] \models m \sim c$ iff $\theta(m)[b, e] \sim c$
$\theta, [b, e] \models D_1 \frown D_2$ iff for some $z: b \leq z \leq e.\ \theta, [b, z] \models D_1$ and $\theta, [z, e] \models D_2$
$\theta, [b, e] \models D_1 \overset{\frown}{+} D_2$ iff for some $z: e \leq z.\ \theta, [b, z] \models D_1$ and $\theta, [e, z] \models D_2$
$\theta, [b, e] \models D_1 \overset{\frown}{-} D_2$ iff for some $z: z \leq b.\ \theta, [z, e] \models D_1$ and $\theta, [z, b] \models D_2$

Derived operators. Note that $\lceil 1 \rceil^0$ holds for all point intervals whereas $\lceil 1 \rceil$ holds for all extended intervals. The formula $\lceil\lceil P \rceil \stackrel{\text{def}}{=} \lceil P \rceil^0 \frown \lceil P \rceil$ states that P must hold invariantly over the interval, except possibly at the last point.

– $\Diamond D \stackrel{\text{def}}{=} true \frown D \frown true$ holds provided D holds for some subinterval.
– $\Box D \stackrel{\text{def}}{=} \neg\Diamond\neg D$ holds provided D holds for all subintervals.
– $\overrightarrow{\Diamond}\, D \stackrel{\text{def}}{=} D \overset{\frown}{+} true$ holds provided some forward extension of the interval satisfies D. Symmetrically $\overleftarrow{\Diamond}\, D \stackrel{\text{def}}{=} true \overset{\frown}{-} D$.

Validity. As usual D is valid iff for all behaviours θ, $\theta \models D$, where

– For prefix behaviours, $\theta \models D$ iff $\theta, [0, max(\theta)] \models D$
– For infinite behaviours, $\theta \models D$ iff $\theta, [0, e] \models D$ for all $e \in dom(\theta)$.

Example 1. The formula $\Box(\lceil\lceil P \rceil \Rightarrow \ell \leq c)$ states that P can be continuously true for at most c time units.

Various sublogics of *IML* have appeared in the literature. We use different signatures to relate our work to a few of these. (The original versions of some of these logics do not include the modalities $D_1 \overset{\frown}{+} D_2$ and $D_1 \overset{\frown}{-} D_2$ which were introduced by Venema [Ven90].)

Duration calculi. Let the signature $Duration(Pvar) = \{\ell\} \cup \{\int P \mid P \in Prop\}$. The term $\int P$ is interpreted to measure the accumulated amount of time for which proposition P is true in an interval. Thus, we obtain the logic *Punct-IML*$[Duration(Pvar)]$. This logic is called duration calculus, DC, when interpreted over continuous time finitely variable models [ZH04]; interval duration logic, IDL, when interpreted over sampled time prefix models [Pan02]; and DDC when interpreted over integer time prefix models.

Mean value calculus. Let the signature $Mean(Pvar) = \{\ell\} \cup \{\overline{P} \mid P \in Prop\}$. The term $\overline{P}$ is interpreted to measure the mean value of proposition P in an interval $[b, e]$. The logic *Punct-IML*$[Mean(Pvar)]$ is the mean value calculus, MVC, interpreted over continuous time finitely variable models [ZL94].

CDT. Consider a signature Σ of measurement functions *without* ℓ. If we only allow comparisons with zero—that is, χ is $Test(\Sigma)$—effectively we are restricting from real-valued measurements to boolean-valued ones. Such a measurement function is nothing but an atomic proposition (such as "did it rain?") evaluated at every interval. This is an idea which has been long studied by philosophers of time. The corresponding logic *Test-IML*$[\Sigma]$ was called CDT [Ven91], as the modalities $\frown$, $\overset{\frown}{-}$, $\overset{\frown}{+}$ are named C, D and T respectively.

Interval length logic. On the other hand, we can consider the signature $\{\ell\}$ without any other measurement functions. The logic *Punct-IML*$[\ell]$ is called interval length logic. As in most real-time logics, it only includes the measurement of time distance using the distinguished function ℓ.

Interval temporal logic. Finally, the trivial logic *IML*$[\emptyset]$ with the empty signature is called interval temporal logic, ITL. This logic has been studied over all the classes of models discussed above.

The discrete time logic DDC has been shown to be decidable using an automata-theoretic decision procedure [Pan02]. The general situation is bleaker.

Proposition 1. *The logic Punct-IML[ℓ] is undecidable for continuous, finitely variable and sampled behaviours whether infinite or prefix. Test-IML[Σ] is undecidable for infinite time.*

Proof. As in the undecidability proof for DC [ZH04], for each 2-counter machine M we can define a formula $D(M)$ of *Punct-IML*[ℓ] which is satisfiable iff M has a halting run. The nonhalting problem is encoded using a very narrow subset of CDT, with unary modalities definable from $\frown$, in [Lod00].

1.2 Guarded Measurement

Faced with the strong undecidability results described above, we restrict the logic by permitting only *guarded* use of measurement formulae. A **χ-guarded modality** has the syntax

$$G \rightarrow D \mid G \leftarrow D,$$

where the guard G is a boolean formula over the set of comparisons χ. The meaning of guarded modalities is as follows:

$$\theta,[b,e] \models G \rightarrow D \text{ iff } b=e \text{ and for some } z : b \leq z.\ \theta,[b,z] \models G \wedge D$$
$$\theta,[b,e] \models G \leftarrow D \text{ iff } b=e \text{ and for some } z : z \leq b.\ \theta,[z,b] \models G \wedge D$$

Formally, $G \rightarrow D \overset{\text{def}}{=} \lceil 1 \rceil^0 \wedge \overset{\rightarrow}{\Diamond} (G \wedge D)$ and $G \leftarrow D \overset{\text{def}}{=} \lceil 1 \rceil^0 \wedge \overset{\leftarrow}{\Diamond} (G \wedge D)$.

Example 2. The formula $\Box(\neg(\ell \leq c) \rightarrow \neg[\lceil P \rceil)$ is *Weak*-guarded and states that P can be continuously true only for at most c time units.

χ*GIML*[Σ] is the sublogic of *IML*[Σ] where measurements only appear in χ-guards. Thus, *Punct-GIML*[Σ] guards use boolean combinations of comparisons from *Punct*(Σ). If only forward (resp. backward) measurement guards are used, we call the logic *FGIML* (resp. *BGIML*). If in the modality $G \rightarrow D$ of *FGIML*, we do not allow guarded modalities in D, we get a logic with nesting-free forward guarding, which we denote *Punct-FgIML*. Guarded modalities exist in the literature, though not in direct fashion.

Relative distance. A subset of interval duration logic *IML*[*Duration*(*Pvar*)] where measurements only occur within the guard G of a modality $P \rightsquigarrow G$ (originally due to [Wil94]) has been called LIDL [Pan02]. This logic can be encoded in the backwards guarded logic *BGIML*[*Duration*(*Pvar*)] by encoding the LIDL formula $P \rightsquigarrow G$ as the *BGIML* formula $G \leftarrow \lceil P \rceil^0 \frown \lceil \neg P \rceil$. All the other constructs of LIDL are already available in *BGIML*.

Metric temporal logic. The logic MTL [Koy90] can be encoded in *Punct-FGIML*[ℓ] as follows (see [Pan96] for details). For every MTL formula ϕ we define a translation $\alpha(\phi)$: Let $\alpha(p) = \lceil p \rceil^0 \frown true$. Let $BP(D) \overset{\text{def}}{=}$

$(\lceil 1 \rceil^0 \wedge \overrightarrow{\Diamond} D)^\frown true$. Then, the constrained until modality of MTL is encoded as follows.

$$\alpha(\phi\, \mathcal{U}_I\, \psi) = (I(\ell) \rightarrow \neg(true^\frown BP(\alpha(\phi)^\frown true))^\frown BP(\alpha(\psi)))^\frown true$$

Here $I(\ell)$ is the constraint corresponding to the interval I, e.g. for $[3,5)$ we get $\ell \geq 3 \wedge \ell < 5$. It can be shown that for all θ, b, e, ϕ we have $\theta, b \models_{mtl} \phi$ iff $\theta, [b,e] \models_{iml} \alpha(\phi)$. By a variation of this construction, we can model MTL with both past and future modalities in *Punct-GIML*$[\ell]$.

Guarding in first order logic has been an important tool for obtaining decidability. Unfortunately, we can show that in the presence of punctual measurements guarding does not guarantee decidability. MTL with future operators is undecidable over continuous infinite time and MTL with past and future operators is undecidable over sampled prefix time [OW05] and the second author and Vijay Suman have recently shown that LIDL is undecidable over sampled time, prefix or infinite. (However, LIDL with only length measurements is decidable over sampled prefix time [Pan02].) All these logics can be encoded within fragments of *GIML* giving the following results.

Proposition 2. *1. The logic Punct-FGIML*$[\ell]$ *is undecidable for continuous infinite time.*
2. The logic Punct-GIML$[\ell]$ *is undecidable for sampled time.*
3. The logic Punct-BGIML$[Duration(Pvar)]$ *is undecidable for sampled time.*

2 First Order Logics with Measurement

In place of interval measurement logic, we can specify a behaviour θ using the first order logic with measurement $MFO[\widehat{\Sigma}]$. This is the first order logic with equality over the signature $\widehat{\Sigma} = (Pvar, \{<\}, \Sigma)$ where each $p \in Pvar$ denotes a monadic predicate.

Example 3. The formula $\forall x, y.\ x < y \wedge (\forall z.\ x < z < y \Rightarrow P(z)) \Rightarrow \ell(x,y) \leq c$ states that P cannot be true continuously for more than c time units.

We can associate a classical first order structure $\underline{\theta}$ interpreting $\widehat{\Sigma}$ with a given behaviour θ. The domain of $\underline{\theta}$ is T with linear order $<$. For each $p \in Pvar$ there is a monadic predicate $p(x)$ which is interpreted as the set $\theta(p)$. The functions $m \in \Sigma$ are interpreted as $\theta(m)$. The semantics of $MFO[\widehat{\Sigma}]$ is given as usual and omitted here.

Proposition 3. *There is a bijection* $(\theta, \underline{\theta})$ *between the IML*$[\Sigma]$ *behaviours and the first-order structures interpreting* $\widehat{\Sigma}$.

While the monadic theory of linear order $MonFO[<]$ is decidable [LL66], introduction of even the basic measurement $\Sigma = \{\ell\}$ makes the logic $MFO[\widehat{\Sigma}]$

undecidable [ZH04]. Hence we resort to a notion of guarded use of measurements. Let $\chi MGF[\Sigma]$ be the measurement-guarded fragment of $MFO[\widehat{\Sigma}]$ which extends $MonFO[<]$ by the **χ-guarded quantifier** $\phi(t_0) = \exists t(G(t_0,t) \wedge \psi(t_0,t))$, where ψ is a formula with at most two free variables t_0 and t, and the guard G is a boolean combination of comparisons from the set χ over the signature Σ. Thus the measurement terms appear in a very restricted context.

We now translate our interval measurement logics into measurement guarded first order logics.

The notation $\phi(x,y)$ indicates a formula with at most two free variables x and y. We will use notation such as $FO(x,y)$ to indicate a logic with formulas with at most two free variables x and y. Superscripts as in FO^k designate k-variable fragments of a logic (now including bound as well as free variables).

$$\begin{aligned}
ST_z(\lceil P \rceil^0)(x,y) &\stackrel{\text{def}}{=} x = y \wedge P(x) \\
ST_z(\lceil P \rceil)(x,y) &\stackrel{\text{def}}{=} x < y \wedge \forall z(x < z < y \Rightarrow P(z)) \\
ST_z(G \rightarrow D)(x,y) &\stackrel{\text{def}}{=} x = y \wedge \exists z(ST(G)(y,z) \wedge y \leq z \wedge ST_x(D)(y,z)) \\
ST_z(G \leftarrow D)(x,y) &\stackrel{\text{def}}{=} x = y \wedge \exists z(ST(G)(z,x) \wedge z \leq x \wedge ST_y(D)(z,x)) \\
ST_z(D_1 \frown D_2)(x,y) &\stackrel{\text{def}}{=} \exists z(x \leq z \leq y \wedge ST_y(D_1)(x,z) \wedge ST_x(D_2)(z,y)) \\
ST_z(D_1 \overset{\frown}{-} D_2)(x,y) &\stackrel{\text{def}}{=} \exists z(z \leq x \wedge ST_y(D_1)(z,x) \wedge ST_x(D_2)(z,y)) \\
ST_z(D_1 \overset{\frown}{+} D_2)(x,y) &\stackrel{\text{def}}{=} \exists z(y \leq z \wedge ST_y(D_1)(x,z) \wedge ST_x(D_2)(y,z))
\end{aligned}$$

The translation of guards is obvious: $ST(m \sim c)(x,y) = m(x,y) \sim c$. The translation uses the standard trick of reusing variables. Thus $ST_z(D)(x,y)$ produces a $MGF^3(x,y)$ formula using at most the variables $\{x,y,z\}$.

Proposition 4. *There is a standard translation from $\chi GIML[\Sigma]$ to $\chi MGF^3[\Sigma]$ which has the property that $\theta,[b,e] \models D$ iff $\underline{\theta} \models ST_z(D)[b/x, e/y]$.*

3 Expressive Completeness of *GIML* for MGF^3

Without loss of generality we assume the logic χMGF^3 consists of formulae with variables x_1, x_2, x_3. In this section, following the proof of Kamp's theorem [Kamp68] as used by Venema [Ven91], we show that the measurement logic $\chi GIML$ has the same expressive power as χMGF^3.

The first lemma is routine [GO]. Let $L_{i,j}$, $i \neq j,\ i,j \in \{1,2,3\}$, be the subset of $\chi MGF^3(x_i,x_j)$ consisting of boolean combinations of quantifier-free formulas of MGF^3 and quantified MGF^3 formulas with one free variable in $\{x_i,x_j\}$. $L_{i,j}$ is the same as $L_{j,i}$.

Lemma 1. *Any MGF^3 formula is equivalent to a boolean combination of formulae from $L_{1,2} \cup L_{2,3} \cup L_{3,1}$.*

Now we translate $L_{i,j}$ to *GIML*. Following [Ven91], we use a forward translation $\alpha^+ : L_{i,j} \rightarrow GIML$ and a backward translation $\alpha^- : L_{i,j} \rightarrow GIML$. The boolean cases are routine. We assume the measurement functions are symmetric.

$$
\begin{array}{ll}
\alpha^+(x = x) = \mathit{true} & \alpha^-(x = x) = \mathit{true} \\
\alpha^+(x_i = x_j) = \lceil 1 \rceil^0 & \alpha^-(x_i = x_j) = \lceil 1 \rceil^0 \\
\alpha^+(x_i < x_j) = \neg \lceil 1 \rceil^0 & \alpha^-(x_i < x_j) = \mathit{false} \\
\alpha^+(x_j < x_i) = \mathit{false} & \alpha^-(x_j < x_i) = \neg \lceil 1 \rceil^0 \\
\alpha^+(x < x) = \mathit{false} & \alpha^-(x < x) = \mathit{false} \\
\alpha^+(P(x_i)) = \lceil P \rceil^0 \frown \mathit{true} & \alpha^-(P(x_i)) = \mathit{true} \frown \lceil P \rceil^0 \\
\alpha^+(P(x_j)) = \mathit{true} \frown \lceil P \rceil^0 & \alpha^-(P(x_j)) = \lceil P \rceil^0 \frown \mathit{true} \\
\alpha^+(m(x,y) \sim c) = m \sim c & \alpha^-(m(x,y) \sim c) = m \sim c \\
\alpha^+(\phi_1(x_i,x_j) \wedge \phi_2(x_i,x_j)) & \alpha^-(\phi_1(x_i,x_j) \wedge \phi_2(x_i,x_j)) \\
\quad = \alpha^+(\phi_1(x_i,x_j) \wedge \alpha^+(\phi_2(x_i,x_j)) & \quad = \alpha^-(\phi_1(x_i,x_j)) \wedge \alpha^-(\phi_2(x_i,x_j)) \\
\alpha^+(\neg\phi(x_i,x_j)) = \neg\alpha^+(\phi(x_i,x_j)) & \alpha^-(\neg\phi(x_i,x_j)) = \neg\alpha^-(\phi(x_i,x_j))
\end{array}
$$

The translation of a quantifier uses the fact that the $\frown$, $\overset{\frown}{-}$, $\overset{\frown}{+}$ modalities cover all cases in which a third time point can be oriented with respect to two points.

$$
\begin{array}{l}
\alpha^+(\exists x_k.\ \phi_1(x_i,x_k) \wedge \phi_2(x_k,x_j)) = \alpha^+(\phi_1(x_i,x_k)) \frown \alpha^+(\phi_2(x_k,x_j)) \quad \vee \\
\quad \alpha^+(\phi_1(x_i,x_k)) \overset{\frown}{+} \alpha^-(\phi_2(x_k,x_j)) \ \vee \ \alpha^-(\phi_1(x_i,x_k)) \overset{\frown}{-} \alpha^+(\phi_2(x_k,x_j)) \\
\alpha^-(\exists x_k.\ \phi_1(x_i,x_k) \wedge \phi_2(x_k,x_j)) = \alpha^-(\phi_2(x_k,x_j)) \frown \alpha^-(\phi_1(x_i,x_k)) \quad \vee \\
\quad \alpha^-(\phi_2(x_k,x_j)) \overset{\frown}{+} \alpha^+(\phi_1(x_i,x_k)) \ \vee \ \alpha^+(\phi_2(x_k,x_j)) \overset{\frown}{-} \alpha^-(\phi_1(x_i,x_k))
\end{array}
$$

The translation of a measurement guarded formula uses the forward and backward guarded modalities to cover the way the quantified variable is oriented with respect to the free variable of the formula.

$$
\begin{array}{l}
\alpha^+(\exists x_k.\ G(x_i,x_k) \wedge \zeta(x_i,x_k)) = \\
\quad [\alpha^+(G(x_i,x_k)) \rightarrow \alpha^+(\zeta(x_i,x_k)) \ \vee \ \alpha^-(G(x_i,x_k)) \leftarrow \alpha^-(\zeta(x_i,x_k))] \frown \mathit{true} \\
\alpha^-(\exists x_k.\ G(x_i,x_k) \wedge \zeta(x_i,x_k)) = \\
\quad \mathit{true} \frown [\alpha^+(G(x_i,x_k)) \rightarrow \alpha^+(\zeta(x_i,x_k)) \ \vee \ \alpha^-(G(x_i,x_k)) \leftarrow \alpha^-(\zeta(x_i,x_k))] \\
\alpha^+(\exists x_k.\ G(x_j,x_k) \wedge \zeta(x_j,x_k)) = \\
\quad \mathit{true} \frown [\alpha^+(G(x_j,x_k)) \rightarrow \alpha^+(\zeta(x_j,x_k)) \ \vee \ \alpha^-(G(x_j,x_k)) \leftarrow \alpha^-(\zeta(x_j,x_k))] \\
\alpha^-(\exists x_k.\ G(x_j,x_k) \wedge \zeta(x_j,x_k)) = \\
\quad [\alpha^+(G(x_j,x_k)) \rightarrow \alpha^+(\zeta(x_j,x_k)) \ \vee \ \alpha^-(G(x_j,x_k)) \leftarrow \alpha^-(\zeta(x_j,x_k))] \frown \mathit{true}
\end{array}
$$

By a careful case analysis over the syntax of $L_{i,j}$, we can show that the translations α^+ and α^- preserve the semantics in the expected way.

Lemma 2. *For all θ and all $[b,e] \in \mathit{Intv}(\theta)$,*
$\theta, [b,e] \models \alpha^+(\zeta(x_i,x_j))$ iff $\underline{\theta} \models \zeta[b/x_i, e/x_j]$ and
$\theta, [b,e] \models \alpha^-(\zeta(x_i,x_j))$ iff $\underline{\theta} \models \zeta[e/x_i, b/x_j]$.

By combining Lemmas 1 and 2, and observing that the translation above can be parameterised by the set of guards χ, we get the following theorem.

Theorem 1. *The logic $\chi\mathit{GIML}[\Sigma]$ is expressively complete for the three-variable measurement guarded fragment with two free variables $\chi\mathit{MGF}^3[\Sigma](x,y)$.*

A referee reminded us that our proof works for an even larger family of logics: $\chi\mathit{IML}[\Sigma]$ is expressively complete for $\chi\mathit{FO}^3[\Sigma](x,y)$.

4 Games and the 3-Variable Property

Next we would like to show that the full logic χMGF has the three-variable property, that is, χMGF^3 is expressively equivalent to it. To do this, we set up Ehrenfeucht-Fraïssé games for the k-variable guarded fragments, which are an extension of the k-pebble games for FO^k [IK87].

Our game is played for n rounds by two players, Spoiler and Duplicator, on a board consisting of a pair of structures $\mathcal{A}$ and $\mathcal{B}$. Spoiler is trying to distinguish the two structures, Duplicator to hide the distinctions. Each player uses k pebbles for the syntactic restriction to k variables and a measuring tape and meters to check lengths and measurement values. These devices work in integer units.

A k-configuration of consists of the positions of the k pebbles on each structure, which we represent by a pair of partial functions (which are defined where the corresponding pebbles are on the board) $\alpha : \{1..k\} \to \mathcal{A}$ and $\beta : \{1..k\} \to \mathcal{B}$. The k-pebble n-round game on structures $\mathcal{A}, \mathcal{B}$ with k-configurations α, β is denoted $G^k_n(\mathcal{A}, \alpha, \mathcal{B}, \beta)$.

Two configurations are said to be order isomorphic if the sequence of pebble positions, seen as linear orders, are order isomorphic. More precisely, Spoiler's pebble i is on the board on one structure if and only if Duplicator's pebble i is present on the other structure, and for each pair of pebbles i, j on the board, both structures satisfy the same formulas from the set $\{i < j, i = j, i > j\}$. By linearity, they will satisfy exactly one formula from this set. Two configurations are said to be χ-measurement isomorphic if they are order isomorphic and for each pair of pebbles i, j on the board, both structures satisfy the same measurement formulas from the set χ.

If α, β are not order isomorphic, Spoiler wins $G^k_n(\mathcal{A}, \alpha, \mathcal{B}, \beta)$ immediately (after 0 rounds). Each round has one of two kinds of moves.

In a pebble move, Spoiler can place his pebble i on an element of one of the structures. Duplicator responds by placing her pebble i on an element of the other structure. After the move, if the resulting configurations α', β' are not order isomorphic, Spoiler wins the game.

In a measuring move, Spoiler removes all pebbles but one, say pebble i (we call this the free pebble of this move), then places another pebble j, using the measuring tape and meters to set its length and other measurement functions to some desired value. Duplicator has to follow suit on the other structure: she removes all pebbles except pebble i, then places her pebble j using the measuring tape and meters. After the move, if the resulting configurations α', β' are not measurement isomorphic, Spoiler wins the game.

If Spoiler has not won the game after any of the n rounds, Duplicator wins $G^k_n(\mathcal{A}, \alpha, \mathcal{B}, \beta)$.

Following [Imm98], we now relate our games to logical types. The proof relies on the fact that the set of measurement formulas $\{m(i, j) \sim c \mid c \in Z\}$ satisfied by a configuration is logically equivalent to a finite conjunction of such formulas, since each value is either the point c or inside an open interval $(c, c + 1)$.

Theorem 2 (E-F characterization). *Given two time frames $\mathcal{A}, \mathcal{B}$ and a k-configuration α_0, β_0 over them, Duplicator wins an n-move game $G_n^k(\mathcal{A}, \alpha_0, \mathcal{B}, \beta_0)$ if and only if the configurations $(\mathcal{A}, \alpha_0)$ and $(\mathcal{B}, \beta_0)$ are indistinguishable by a χMGF^k formula of quantifier depth n.*

We now show that for time domains, three variables suffice to express all *MGF* properties. The proof closely follows the (admittedly tricky) one of [Imm98, Theorem 6.32], which combines winning strategies from simpler games. The measuring move does not yield any difficulty since it always reduces the board to 2-configurations for which a winning strategy exists by supposition.

Theorem 3 (3-variable property). *Every χMGF formula is equivalent to an χMGF^3 formula over time domains.*

Putting together the 3-variable property with the expressive completeness result of the previous section, we get a proper generalization of Kamp's theorem. Venema has shown that $\chi FO[\Sigma]$ does not have the 3-variable property [Ven90], so the result cannot be extended to full first order logic with measurements.

Corollary 1. *The logic $\chi GIML[\Sigma]$ is expressively complete for $\chi MGF[\Sigma]$.*

Hirshfeld and Rabinovich conjectured that there is no finite expressively complete temporal logic for a logic L_2 which subsumes *Weak-MGF*$[\ell]$ by having a more generous set of comparisons [HR99]. We observe that since our logic uses countably many constants c, it is not finite according to their definition.

5 Decidability

One of the main motivations for considering the sublogic *GIML* of *IML* is the hope of getting reasonable decidability properties. In this section we restrict our attention to only the measurement of length, that is, the signature $\Sigma = \{\ell\}$.

Theorem 4. *Over sampled as well as continuous time infinite models, the logic Punct-MGF$[\ell]$ is undecidable and the logics Weak-MGF$[\ell]$ and Weak-GIML$[\ell]$ are decidable.*

Proof. It is shown in the previous section that *Punct-GIML*$[\ell]$ and *Weak-GIML*$[\ell]$ are expressively equivalent to *Punct-MGF*$[\ell]$ and *Weak-MGF*$[\ell]$ and can be translated to these logics. The undecidability of *Punct-MGF*$[\ell]$ for sampled and continuous time infinite models follows from that of *FGIML*$[\ell]$ (Proposition 2). *Weak-GIML*$[\ell]$ is decidable since *Weak-MGF*$[\ell]$ is subsumed by the logic L_2, which was shown decidable over continuous time infinite models [HR99].

The exact decidability border between *Punct-GIML*$[\ell]$ and *Weak-GIML*$[\ell]$ is not clear. As a preliminary result, we show that for sampled time prefix models (i.e. timed words), the logic *Punct-FgIML*$[\ell]$, the nesting-free subset of *FGIML*$[\ell]$, is decidable by reduction to alternating timed automata.

5.1 Alternating Timed Automata

Let C be a finite set of clock variables (more briefly, clocks). A clock valuation v is a function $C \to \Re$. The clock valuation $v + t$ is defined by adding t to each clock value, and the valuation $v[r := 0]$, for $r \subseteq C$, is defined by resetting all the clocks in r to zero. The initial valuation v_0 has all clocks set to zero.

A clock constraint is a boolean combination of conditions $x \sim c$ where x is a clock. Let $Cons(C)$ be the constraints over clocks in C.

Definition 1 (Lasota and Walukiewicz, Ouaknine and Worrell). *An **alternating timed automaton** over the alphabet A and clocks C is a tuple $M = (Q, \delta, I, F)$, where Q is a finite set of states, $I, F \subseteq Q$ are the initial and final states respectively, and $\delta : Q \times A \times Cons(C) \to \mathcal{B}^+(Q \times \wp(C))$ a finite partial transition function, satisfying the partition condition: for every $q \in Q$ and $a \in A$, the set of constraints $\{[\sigma] \mid \delta(q, a, \sigma)$ is defined$\}$ partitions the set of clock valuations $C \to \Re$. ($\mathcal{B}^+(X)$ is positive boolean formulas over X.)*

A timed word over A is a sequence over $A \times \Re$. The second (time) component gives the time elapsed between reading the previous letter and the current one.

The acceptance game $G_{M,w}$ between two players Pathfinder and Automaton is defined as follows. Automaton's objective is to accept $w = (a_1, t_1) \ldots (a_n, t_n)$, Pathfinder's is to reject. A play starts at the configuration (q_0, v_0) and proceeds for $n = |w|$ rounds. Suppose the configuration reached at the end of the i'th round is (q_i, v_i). Let σ be the unique constraint satisfied by the valuation $v_i + t_{i+1}$ and $\delta(q_i, a_{i+1}, \sigma)$ is the formula b. Now there are three cases: if b is a conjunction, Pathfinder chooses one of the conjuncts; if b is a disjunction, Automaton chooses one of the disjuncts; and if $b = q$, the round ends with $q_{i+1} = q$ and $v_{i+1} = (v_i + t_{i+1})[x := 0, x \in \rho(q_{i+1})]$. Automaton wins the game if q_n is a final state, otherwise Pathfinder wins.

The automaton M accepts the timed word w if and only if Automaton has a winning strategy in the game $G_{M,w}$.

The languages accepted by ATA are closed under boolean operations. The papers [LW05, OW05] showed that the emptiness problem for ATA with one clock is decidable. It follows from [AD94] that the problem is undecidable for ATA with two clocks. It is known [LW05] that ATA, even with one clock, are incomparable in expressive power to the usual nondeterministic timed automata of Alur and Dill [AD94].

In order to accept timed languages defined by formulas, automata have to work over models of these formulas. Typically an alphabet 2^{Pvar} is used.

5.2 Decidability of the Nesting-Free Logic *Punct-FgIML*[ℓ]

Recall that *Punct-FgIML*[ℓ] is the subset of *Punct-GIML*[ℓ] with only forward guarded modalities $G \to D$ and where no guarded modality occurs in D. Moreover, we will confine ourselves to sampled time prefix models (finite timed words). The usual Alur-Dill timed automata [AD94], as well as the ATA introduced in the previous section, are recognisers for such timed words. They have decidable emptiness checking. We give an automata-theoretic decision procedure for *Punct-FgIML*$[\ell]$ by reduction to emptiness of ATA. This logic can already express properties not recognised by any timed automaton.

Example 4. Consider behaviour over a single propositional variable. The following property says that there are no pairs of positions exactly one time unit apart: $\neg(\mathit{true}^\frown(\ell = 1 \rightarrow \mathit{true})^\frown \mathit{true})$.

Theorem 5. *For each $D \in$ Punct-FgIML$[\ell]$ over Pvar, we can construct ATA $A(D)$ over alphabet 2^{Pvar} with a single clock x such that $\theta \models D$ iff $\theta \in L(A(D))$.*

Corollary 2. *Punct-FgIML$[\ell]$ is decidable for sampled time prefix models.*

Proof (of Theorem 5). We give the construction of $A(D)$ inductively as follows. Some specific features of our automata are first outlined: These automata have a unique starting state which is never accepting. Our automata never accept the empty word. The automata may also have two distinguished states $\top, \bot$ where $\top$ is accepting and $\bot$ is rejecting. We assume that $\delta(\top, P, \mathit{true}) = \top$ and $\delta(\bot, P, \mathit{true})$ for all P. The symbol x denotes that the unique clock x is reset and $\overline{x}$ that it is not reset.

(i) In the base case we have $D \in ITL$ without any measurements. Then, we can straightforwardly construct a DFA $A(D)$ recognising the models of D. Note that the models of D satisfiable at an interval $[b, e]$ with $b = e$ are accepted by $A(D)$ with a transition from an initial state to a final state.

(ii) Next, we construct one-clock ATA for $GQ = G \rightarrow D$. By the nesting-free property, D is a pure ITL formula without measurements. Let $A(D) = (Q, \delta, q_0, F)$ be the DFA for D. Then $A(GQ) = (Q \cup \{\top, q_0'\}, \delta', q_0', \{\top\})$. The transition relation δ' is defined as follows.

- Let $\delta(q_0, P) = q$. If $G[0/\ell]$ evaluates to true and $q \in F$, then $\delta'(q_0', P, \mathit{true}) = (\top, \overline{x})$. Otherwise, $\delta'(q_0', P, \mathit{true}) = (q, x)$.
- Let $\delta(q, P) = q'$. If $q' \notin F$, then $\delta'(q, P, \mathit{true}) = (q', \overline{x})$. Otherwise, $q' \in F$ and we have $\delta'(q, P, G[x/\ell]) = (\top, \overline{x})$ and $\delta'(q, P, \neg G[x/\ell]) = (q', \overline{x})$.

Claim: $A(GQ)$ accepts all nonempty words θ such that $\theta, [0, 0] \models GQ^\frown \mathit{true}$.

To prove the claim, by the property mentioned in *(i)*, the models of length 0 are obtained by the immediate transitions to the final state. For the other models, the automaton moves off from the initial state (the clock condition for these initial transitions is *true*) and reaches the final state after checking the guard. Since by our convention $\delta(\top, P, \mathit{true}) = \top$, the automaton will continue accepting an extension of a GQ behaviour.

$A(GQ)$ is a deterministic one-clock timed automaton (without alternation). Let $\neg A(GQ)$ denote the complement of $A(GQ)$ where all $\wedge, \vee$ are exchanged and $\top, \bot$ are exchanged and the final states are Q. (That is, the final states are complemented, but since the empty word is not to be accepted, we remove the initial state q_0' from the final states.) It is clear that $\neg A(GQ)$ is a one-clock ATA recognising the complement of the language accepted by $A(GQ)$.

(iii) Next, let D be a formula having syntactic occurrences $GQ_1, \dots, GQ_k$ of guarded modalities. Using from above the ATAs $A(GQ_i)$ and $\neg A(GQ_i)$, for each i, we construct the one-clock ATA $A(D)$.

Let $GW = \{p_1, \dots, p_k\}$ be fresh witness propositions. Replace each occurrence of a guarded modality GQ_i by witness formula $\lceil p_i \rceil^0$ in D to get the ITL formula DW (without measurements) such that $D = DW[GQ_i/\lceil p_i \rceil^0, i = 1..k]$.

Let $A(DW) = (Q, \delta, q_0, F)$ be the deterministic finite automaton accepting finite nonempty words over the extended alphabet $2^{Pvar \cup GW}$ which are models of DW. Let $A(GQ_i) = (Q_i, \delta_i, s_i, F_i)$ and $\neg A(GQ_i) = (Q_i', \delta_i', s_i', F_i')$. We construct the one-clock ATA $A(D) = (Q'', \delta'', q_0'', F'')$ as follows. The states Q'' are a disjoint union of the states Q of $A(DW)$ together with the states Q_i and Q_i' of $A(GQ_i)$ and $\neg A(GQ_i)$. The initial state is $q_0'' = q_0$. Final states are also unions of the final states of the component automata $A(DW)$, $A(GQ_i)$ and $\neg A(GQ_i)$.

For each $P \subseteq Pvar$ we have in $A(D)$:

$$\delta''(q, P, true) = \bigvee_{P \subseteq S \subseteq P \cup GW} \left((\delta(q, S), \overline{x}) \wedge \bigwedge_{p_i \in S} \delta_i(s_i, P, true) \wedge \bigwedge_{p_i \notin S} \delta_i'(s_i', P, true) \right)$$

By induction on the number of occurrences of guarded modalities k, we now show that $A(D)$ recognises the θ satisfying D.

For the base case, if there are no guarded modalities and no witnesses, then $A(D) = A(DW)$ does accept models of $DW = D$.

For the induction step, consider $D = DW[GQ/\lceil p \rceil^0]$ for an additional guarded modality GQ and witness proposition p.

Suppose θ is a model of D. Then there is a corresponding model θ_p of DW over the alphabet $2^{Pvar \cup \{p\}}$ which is accepted by $A(DW)$, determining a subset S at every point along the word. The transition function $\delta(q, S)$ determines $\delta''(q, S \cap Pvar)$, which ensures that the corresponding constraint is checked by $A(GQ)$ or $\neg A(GQ)$ and the substituted model θ is accepted by $A(D)$.

Conversely, suppose θ is a word accepted by $A(D)$. At each point of the word, we can ask whether the point formula GQ holds or not. Substituting GQ by p, this defines a set of models of DW over the alphabet $2^{Pvar \cup \{p\}}$. Each such model θ_p is accepted by $A(DW)$, and the $\wedge$-branches in the transition function δ'' ensure, using the automata $A(GQ)$ and $\neg A(GQ)$, that the corresponding timing constraint holds so that θ_p with the valuation of p removed, i.e. θ, is a model of $DW[GQ/\lceil p \rceil^0] = D$.

Acknowledgements. Four FORMATS referees commented on this paper. After reading what they said, we wondered whether we should have called this paper an *overdose*! This is not to criticise the referees: we found their detailed reports constructive and helpful, and would like to thank them. We have removed some of the proofs from this conference version and are preparing a full version.

References

[AD94] R. Alur and D. Dill. A theory of timed automata, *TCS* 126, 1994, 183–236.

[AFH96] R. Alur, T. Feder and T. Henzinger. The benefits of relaxing punctuality, *J.ACM* 43(1), 1996, 116–146.

[Bac03] M. Baclet. Logical characterization of aperiodic data languages, Research Report LSV-03-12 (ENS Cachan, 2003), 16 pp.

[BPT03] P. Bouyer, A. Petit and D. Thérien. An algebraic approach to data languages and timed languages, *Inf. Comput.* 182(2), 2003, 137–162.

[GO] V. Goranko and M. Otto. Model theory of modal logic, in *Handbook of modal logic*, in preparation.

[Gue00] D.P. Guelev. Probabilistic neighbourhood logic, *Proc. FTRTFT*, Pune, *LNCS* 1926 (M. Joseph, ed.), 2000, 264–275.

[HR99] Y. Hirshfeld and A. Rabinovich. A framework for decidable metrical logics, *Proc. ICALP*, Prague, *LNCS* 1644 (J. Wiedermann, P. van Emde Boas and M. Nielsen, eds.), 1999, 422–432.

[IK87] N. Immerman and D. Kozen. Definability with bounded number of bound variables, *Proc. LICS*, Ithaca (IEEE, 1987).

[Imm98] N. Immerman. *Descriptive complexity* (Springer, 1998).

[Kamp68] J.A.W. Kamp. *Tense logic and the theory of linear order*, PhD thesis (UCLA, 1968).

[Koy90] R. Koymans. Specifying real-time properties with metric temporal logic, *Real-time systems* 2(4), 1990, 255–299.

[LW05] S. Lasota and I. Walukiewicz. Alternating timed automata, *Proc. Fossacs*, Edinburgh, *LNCS* 3441 (V. Sassone, ed.), 2005, 250–265.

[LL66] H. Läuchli and J. Leonard. On the elementary theory of linear order, *Fund. Math.* 59, 1966, 109–116.

[Lod00] K. Lodaya. Sharpening the undecidability of interval temporal logic, *Proc. Asian*, Penang, *LNCS* 1961 (J. He and M. Sato, eds.), 2000, 290–298.

[OW05] J. Ouaknine and J. Worrell. On the decidability of metric temporal logic, *Proc. LICS*, Chicago (IEEE, 2005), 188–197.

[Pan96] P.K. Pandya. Weak chop inverses and liveness in mean-value calculus, *Proc. FTRTFT*, Uppsala, *LNCS* 1135 (B. Jonsson and J. Parrow, eds.), 1996, 148–167.

[Pan02] P.K. Pandya. Interval duration logic: expressiveness and decidability, *Proc. TPTS*, Grenoble, *ENTCS* 65(6) (E. Asarin, O. Maler and S. Yovine, eds.), 2002, 19 pp.

[Ven90] Y. Venema. Expressiveness and completeness of an interval tense logic, *Notre Dame J.FL* 31(4), 1990, 529–547.

[Ven91] Y. Venema. A modal logic for chopping intervals, *J. Logic Comput.* 1(4), 1991, 453–476.

[Wil94] T. Wilke. Specifying timed state sequences in powerful decidable logics and timed automata, *Proc. FTRTFT*, Lübeck, *LNCS* 863 (H. Langmaack, W.P. de Roever and J. Vytopil, eds.), 1994, 694–715.

[ZH04] Zhou C. and M.R. Hansen. *Duration calculus* (Springer, 2004).

[ZL94] Zhou C. and Li X. A mean value calculus of durations, in A.W. Roscoe (ed.), *A classical mind: Essays in honour of C.A.R. Hoare* (Prentice-Hall, 1994), 431–451.

From MITL to Timed Automata*

Oded Maler[1], Dejan Nickovic[1], and Amir Pnueli[2,3]

[1] Verimag, 2 Av. de Vignate, 38610 Gières, France
{Dejan.Nickovic, Oded.Maler}@imag.fr
[2] Weizmann Institute of Science, Rehovot 76100, Israel
[3] New York University, 251 Mercer St. New York, NY 10012, USA
Amir.Pnueli@cs.nyu.edu

Abstract. We show how to transform formulae written in the real-time temporal logic MITL into timed automata that recognize their satisfying models. This compositional construction is much simpler than previously known and can be easily implemented.

Prediction is very difficult, especially about the future.
Niels Bohr

1 Introduction

Decision procedures for model-checking of temporal logic formulae [MP91, MP95] play a central role in algorithmic verification [CGP99]. Such procedures are based, in the linear-time context, on deriving from a formula φ an automaton-like device that accepts exactly sequences of states or events that satisfy it [VW86]. For discrete-time models, used for functional verification of software or synchronous hardware, the logical situation is rather mature. Logics like LTL (linear-time temporal logic) or CTL (computation-tree logic) are commonly accepted and incorporated into verification tools. LTL admits a variety of efficient algorithms for translating a formula into an equivalent automaton [GPVW95, SB00, GO01, KP05] and it even underlies the industrial standard PSL [KCV04, HFE04].

When considering *timed* models and specification formalisms whose semantics involves the time domain $\mathbb{R}_+$ rather than $\mathbb{N}$, the situation is somewhat less satisfactory [A04]. Many variants of real-time logics [Koy90, AH92a, Hen98, HR04] as well as timed regular expressions [ACM02] have been proposed but the correspondence between simply-defined logics and variants of timed automata (automata with auxiliary clock variables [AD94]) is not as simple and canonical as for the untimed case, partly, of course, due to the additional complexity of the timed model. Consequently, existing verification tools for timed automata rarely use temporal properties other than safety. One of the most popular dense-time extensions of LTL is the logic MITL introduced in [AFH96] as a restriction of the logic MTL [Koy90]. The principal modality of MITL is the *timed until* $\mathcal{U}_I$ where I is some non-singular interval. A

* This work was partially supported by the European Community project IST-2003-507219 PROSYD (Property-based System Design).

formula $p\mathcal{U}_{[a,b]}\, q$ is satisfied by a model at any time instant t that admits q at some $t' \in [t+a, t+b]$, and where p holds continuously from t to t'. The decidability of MITL was established in [AFH96] and it was, together with MTL, subject to further investigations [AH92b, RSH98, HRS98, OW05]. However, the automaton construction in [AFH96] is very complicated (11 pages) and, to the best of our knowledge, has never been implemented. The only logic that has been integrated into a real-time model-checking tool was the timed version of CTL, TCTL [HNSY94], used in the tool KRONOS [Y97].[4]

In this paper we remedy this situation somewhat by presenting a simple construction of timed automata that accept exactly models of MITL formulae. The construction is based on two ideas, the first being the *modular construction* of property testers for untimed formulae [KP05] and the other is the observation, similar to the one already made in [AFH96], that the evolution over time of the satisfiability of a formula of the form $p\mathcal{U}_{[a,b]}\, q$ is of bounded variability, regardless of the variability of p and q.

The rest of the paper is organized as follows. In Section 2 we illustrate the modular construction of testers for (untimed) LTL formulae. The logic MITL is presented in Section 3 together with its semantic domain (Boolean signals) and timed automata. The main result, the construction of property testers for MITL, is presented in Section 4, followed by a brief discussion of the differences between the version of MITL that we use and the one presented in [AFH96].

2 Temporal Testers for LTL

In this section we discuss some of the problems associated with the construction of automata that accept models of LTL formulae, and describe the modular procedure of [KP05] which we later extend for MITL. We feel that, independently of its timed generalization, this construction, based on composition of *acausal sequential transducers*, improves our understanding of temporal logic and can serve as a unifying framework for both verification and monitoring. We assume familiarity with basic notions of LTL, whose syntax is defined according to the grammar

$$\varphi := p \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \bigcirc \varphi \mid \varphi_1 \mathcal{U} \varphi_2$$

where p belongs to a set $P = \{p_1, \ldots, p_n\}$ of propositions. LTL is interpreted over n-dimensional Boolean ω-sequences of the form $\xi : \mathbb{N} \rightarrow \mathbb{B}^n$. We abuse p to denote the projection of the sequence ξ on p. The semantics of LTL formulae is typically given via a recursive definition of the relation $(\xi, t) \models \varphi$ indicating that the sequence ξ satisfies φ at position t. The satisfaction of a compound formula $op(\varphi_1, \varphi_2)$, where op is a temporal or propositional operator, by a sequence ξ is an op-dependent function of the satisfaction of the sub formulae φ_1 and φ_2 by ξ. Functionally speaking, the satisfaction of φ by arbitrary sequences can be viewed as a *characteristic function* χ^φ which maps sequences over $\mathbb{B}^n$ into Boolean sequences such that $\beta = \chi^\varphi(\xi)$

[4] An efficient emptiness checking algorithm for timed Büchi automata has been proposed and implemented in [TYB05] but without an upstream translation from a logical formalism.

means that for every i, $\beta[i] = 1$ iff $(\xi, i) \models \varphi$. The semantics of LTL can thus be formulated[5] as a recursive definition of χ^φ:

$$\begin{array}{lcl} \chi^p[t] & = & p[t] \\ \chi^{\neg\varphi}[t] & = & \neg\chi^\varphi[t] \\ \chi^{\varphi_1\vee\varphi_2}[t] & = & \chi^{\varphi_1}[t] \vee \chi^{\varphi_2}[t] \\ \chi^{\bigcirc\varphi}[t] & = & \chi^\varphi[t+1] \\ \chi^{\varphi_1\mathcal{U}\varphi_2}[t] & = & \bigvee_{t'\geq t}(\chi^{\varphi_2}[t'] \wedge \bigwedge_{t''\in[t,t']} \chi^{\varphi_1}[t'']) \end{array} \tag{1}$$

Given a formula φ, its characteristic function is defined as a composition of sequential functions, following the pattern of its parse tree, as illustrated in Figure 1. So what remains to be done is to build an automaton that realizes the appropriate sequential function for each LTL operator and use these building blocks, that we call *temporal testers*, to construct a mechanism that computes the characteristic function for arbitrary formulae, but some problem related to causality should be settled first.

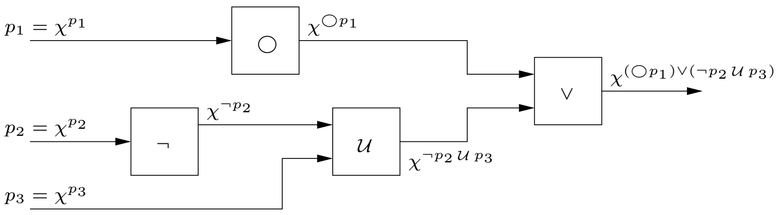


Fig. 1. Computing the characteristic function of $(\bigcirc p_1) \vee (\neg p_2\,\mathcal{U}\,p_3)$

A sequential function $f : A^\omega \to B^\omega$ is said to be *causal* if for every $u \in A^*$, $v, v' \in B^*$ such that $|u| = |v| = |v'|$ and every $\alpha \in A^\omega$ and $\beta \in B^\omega$

$$f(u \cdot \alpha) = v \cdot \beta \text{ and } f(u \cdot \alpha') = v' \cdot \beta' \text{ implies } v = v'.$$

In other words, the value of $f(\alpha)$ at time t may depend only on the values $\{\alpha[t'] : t' \leq t\}$. Causal functions are realized naturally by deterministic automata with output (sequential synchronous transducers). However, unlike the semantics of the *past* fragment of LTL which is causal (see [HR02]), the semantics of future LTL is not. Looking closely at (1) we can see that while the propositional operators define causal sequential functions, the future temporal operators are acausal. The output of the *next* operator at time t depends on the input at $t + 1$ and, even worse, the output of the *until* operator at t may depend on input values at some t' which may be arbitrarily further in the future.

One can think of two ways to realize acausal sequential functions. The first approach, which works well for operators with a *bounded* level of acausality, such as

[5] Throughout the paper we use a variant of *until* in which $p\,\mathcal{U}\,q$ requires that p holds also at the same time instant when q becomes true, which can be expressed as $p\,\mathcal{U}(p\wedge q)$ using the standard interpretation. For LTL this variation just simplifies the corresponding tester while for MITL it avoids certain anomalies.

the *next* operator or several nestings of which (denoted by $\bigcirc^d$) is to dissociate the time scales of the input and the output. That is, let the automaton ignore the first d input symbols, and then let $\beta[t] = \alpha[t+d]$. Unfortunately this does not always work for unbounded acausality.

Using the second approach the transducer responds to the input *synchronously* but since at time t the information might not be sufficient for determining the output, it will have to "guess" the output non-deterministically and split the computation into two runs, one that predicts 0 and one that predicts 1. Each of the runs needs to remember its predictions until sufficient input information is accumulated to abort all but one run. The automaton for operators with acausality of depth d, may need to memorize up to 2^d predictions. The first approach is more intuitive[6] but since we do not know how to extend it to unbounded acausality, we use the second one. The automaton that computes the characteristic function of $\bigcirc$ is depicted in Figure 2 along with a sample run.[7] State s_0 indicates that the prediction made in the previous step was 0, hence at this state, observing p contradicts the prediction and the run is aborted. Input $\overline{p}$ confirms the prediction and the run splits by generating two predictions for the next value and so on. For every infinite input sequence ξ, only *one* infinite run survives and its output is $\chi^{\bigcirc p}(\xi)$. The automaton for $\bigcirc^d p$ follows the same pattern and is nothing but an output-driven shift register of depth d (see Figure 2). States s_{00} and s_{01} of this automaton represent predictions for unsatisfiability at $t-2$ and hence admit only $\overline{p}$-labeled transitions. It is worth mentioning that these

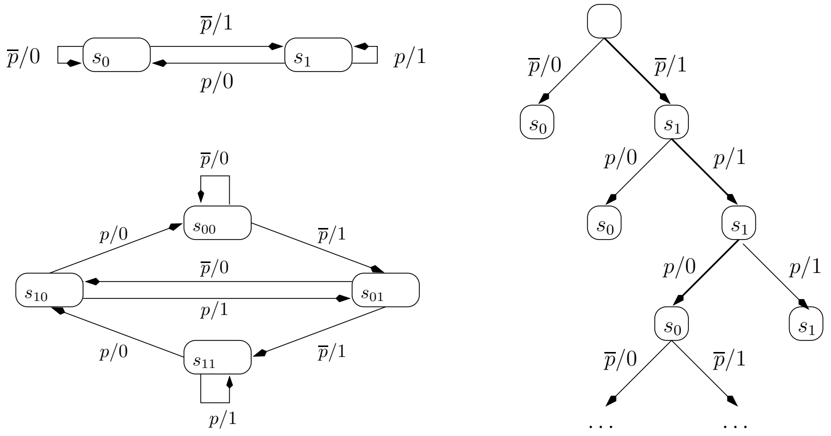


Fig. 2. The testers for $\bigcirc p$ and for $\bigcirc(\bigcirc p)$; An initial fragment of the behavior for the $\bigcirc p$-tester while computing $\chi^{\bigcirc}(\overline{p}pp\overline{p}\cdots) = 110\cdots$

[6] After all, it is more natural to reply "I don't know yet" rather than saying "the answer now could be either 0 or 1, but only in the next step I will tell you which of them is actually true".

[7] To ease readability we use $\{p,\overline{p}\}$ instead of $\{0,1\}$ as the input alphabet for unary operators and $\{\overline{pq},\overline{p}q,p\overline{q},pq\}$ for the binary ones. Moreover, we use p as a shorthand for $\{p\overline{q},pq\}$.

automata are output-deterministic and can be obtained by reversing the transitions in the ordinary input-driven shift registers that correspond to the past temporal operator *previously*, also known as the delay operator z^{-1}.

The situation with $p\mathcal{U}q$ is more involved because a priori, due to the unbounded horizon, one might need to generate and memorize 2^ω predictions. However the semantics of *until* implies that at most *two* confirmable predictions may co-exist simultaneously.

Lemma 1. *Let $\beta = \chi^{p\mathcal{U}q}(\xi)$. Then for every t such that $\xi[t] = \xi[t+1] = p\overline{q}$, $\beta[t] = \beta[t+1]$.*

Proof. There are three possibilities: 1) The earliest $t' > t+1$ such that $\beta[t'] \neq p\overline{q}$ satisfies $\beta[t'] = pq$. In that case, the property is satisfied at t and $t+1$; 2) The same t' satisfies $\beta[t'] = \overline{p}$ and the property is falsified at both t and $t+1$; 3) $\beta[t'] = p\overline{q}$ for every $t' > t+1$ and the property is falsified from both time points.[8] ∎

This fact is reflected by the tester of Figure 3. At time instants where $\overline{p}$ is observed, the value of the output is determined to be 0. Likewise when pq is observed the output is determined to be 1. The only situation that calls for non-determinism is when $p\overline{q}$ occurs and we do not know in which of the three cases of Lemma 1 we will eventually find ourselves. Hence we split the run into positive and negative predictions (states $s_{p\overline{q}}$ and $\overline{s}_{p\overline{q}}$, respectively). The only input sequences that will lead to *two* infinite runs are those ending with an infinite suffix of $p\overline{q}$'s. To choose the correct one among the two we add a Büchi condition requiring infinitely many visits in the set $\{s_{\overline{p}}, \overline{s}_{p\overline{q}}, s_{pq}\}$ which amounts to rejecting runs that stay forever in $s_{p\overline{q}}$. With these two testers (and simple testers for the Boolean operators) one can build testers for arbitrary LTL formulae. Note that in both the *next* and *until* testers, all transitions entering a state are decorated by the same output symbols so, in fact, one might view outputs as associated with states rather than with transitions, as is done in [KP05].

The notion of compositional temporal testers, as introduced in [KP05], is based on several key ideas. One of them is the allocation of an individual Boolean variable to each sub-formula of an LTL formula. This idea, to which we refer in [KP05] as *statification* of the sub-formula, has been considered first in [BCM$^+$92] in the context of symbolic implementation of a *tableau* construction. The observation that such a Boolean variable can replace the sub-formula itself in the context of model checking has been considered in [CGH94]. It is worth mentioning the translation of LTL to *alternating automata* [Var96] which, like the temporal testers approach, works inductively on the structure of the formula but not in a compositional manner.

3 Signals, Their Temporal Logic and Timed Automata

Extending the construction of temporal testers from discrete to dense time requires adaptations of the semantic domain, the logic and the automata. The interaction

[8] This is the "strong" interpretation of *until*. For the weak version both t and $t+1$ will satisfy the property.

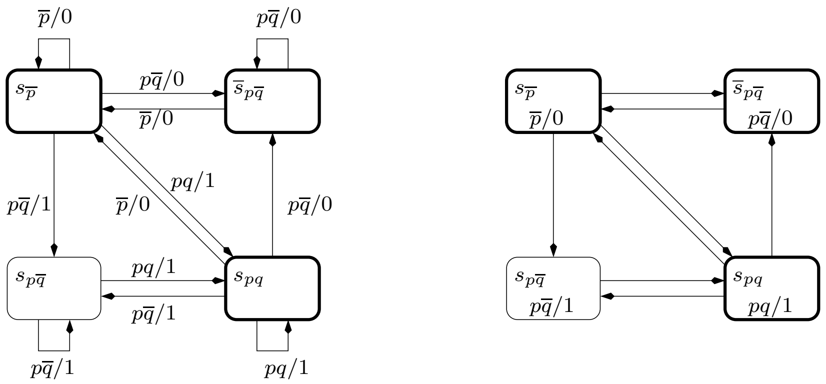


Fig. 3. The tester for $p\,\mathcal{U}\,q$ for sequences (left) and for signals (right). All states except $s_{p\overline{q}}$ are accepting. All transitions in the signal tester should be decorated by $z > 0/z := 0$, with z being an auxiliary clock, to force signals to maintain each value for a non-punctual duration.

between discrete transitions and dense time may give rise to certain anomalies and complications that we avoid by deviating slightly from the original definitions of [AFH96].

3.1 Signals

Two basic semantic domains for describing timed behaviors have been introduced in an algebraic form in [ACM02, A04]. The first semantic objects are the *time-event sequences*, elements of the free product (shuffle) of the monoids Σ^* and $\mathbb{R}_+$. Time-event sequences consist of instantaneous events (or finite sequences of events) separated by numbers that represent time durations. The other semantic domain is that of *signals*, elements of the free product of several copies of the monoid $\mathbb{R}_+$, a copy for each alphabet symbol. Signals are thus viewed as concatenations of "typed" real numbers, where $\sigma_1^5 \cdot \sigma_2^3$ represents a σ_1 period of duration 5 followed by a σ_2 period of duration 3. Zero is the identity element of each of the monoids and it is absorbed when signals are transformed to a canonical form, that is $\sigma_1^{t_1} \cdot \sigma^0 \cdot \sigma_2^{t_2} = \sigma_1^{t_1} \cdot \sigma_2^{t_2}$. A *signal-event monoid* containing both can be defined as well [ACM02].

The transformation of these objects into the form of functions from the time domain to the alphabet is not painless. For time-event sequences a "super-dense" time [MMP92] had to be invented, a subset of the product of $\mathbb{R}_+$ and $\mathbb{N}$, where a sequence of discrete events, all taking place at time t, is mapped to the sequence $(t,1),(t,2),\ldots$ of generalized time instants. The timed traces of [AD94] constitute another representation of the same objects. Signals, which are the natural domain for MITL, are more well-behaving in this respect and can be mapped bijectively to functions from $\mathbb{R}_+$ to the alphabet if one accepts some restrictions. The one we adopt is to view a signal of the form $\sigma_1^{t_1} \cdot \sigma_2^{t_2}$ as a function whose value is σ_1 in the *left-closed right-open interval* $[0,t_1)$ and σ_2 in the interval $[t_1, t_1+t_2)$. Since such intervals cannot be punctual without being empty, we exclude from the discussion signals that may have a distinct value in some isolated point.[9] Since our construction is based on characteristic functions and signal transducers, we need this property of signals to be preserved by the temporal operators, hence we deviate from [AFH96] by restricting the quantitative temporal modalities to closed intervals.

Definition 1 (Signals). *Let $\mathbb{R}_+$ be the set of non-negative real numbers. A Boolean signal is a function $\xi : \mathbb{R}_+ \to \mathbb{B}^n$ such that for every $\sigma \in \mathbb{B}^n$, $\xi^{-1}(\sigma)$ is a union of* left-closed right-open *intervals.*

A partial signal is a restriction of a signal to some interval $[a, b)$. We will sometimes refer to a partial signal of the form $\sigma_1^{t_1} \cdot \sigma_2^{t_2} \cdots \sigma_k^{t_k}$ as a $\sigma_1 \cdot \sigma_2 \cdots \sigma_k$-segment of the signal.

3.2 Real-Time Temporal Logic

The syntax of MITL is defined by the grammar

$$\varphi := p \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \mathcal{U}_{[a,b]} \varphi_2 \mid \varphi_1 \mathcal{U} \varphi_2$$

where p belongs to a set $P = \{p_1, \ldots, p_n\}$ of propositions and $b > a \geq 0$ are rational numbers (in fact, using normalization, it is sufficient to consider integers). From basic MITL operators one can derive other standard Boolean and temporal operators, in particular the time-constrained *eventually* and *always* operators

$$\Diamond_{[a,b]}\varphi = \mathrm{T}\ \mathcal{U}_{[a,b]}\varphi \quad \text{and} \quad \Box_{[a,b]}\varphi = \neg\Diamond_{[a,b]}\neg\varphi.$$

We interpret MITL over n-dimensional Boolean signals and define the satisfiability relation via characteristic functions, which for the propositional operators and the untimed *until* are defined exactly as for LTL. The semantics of timed *until* is given by

$$\chi^{\varphi_1 \mathcal{U}_{[a,b]} \varphi_2}[t] \;=\; \bigvee_{t' \in [t+a,t+b]} (\chi^{\varphi_2}[t'] \wedge \bigwedge_{t'' \in [t,t']} \chi^{\varphi_1}[t''])$$

The difference with respect to untimed *until* is that t' ranges over the bounded (but dense) interval $[t + a, t + b]$ rather than over the unbounded set $\{t, t + 1, \ldots\}$ of integers. Our interpretation of the timed *until* slightly deviates from [AFH96] by requiring φ_1 to hold also at the moment t' when φ_2 becomes true, and not only in the open interval (t, t'). A signal ξ satisfies the formula φ iff $\chi^{\varphi}(\xi)[0] = 1$.

The following lemma, proved also in [DT04], shows that timed *until* can be expressed by a combination of untimed *until* and timed *eventually* whose characteristic function is

$$\chi^{\Diamond_{[a,b]}\varphi} \;=\; \bigvee_{t' \in [t+a,t+b]} \chi^{\varphi}[t'].$$

Lemma 2 (Timed Until is Redundant). *For every signal ξ,*

$$\xi \models p\mathcal{U}_{[a,b]}q \;\leftrightarrow\; \xi \models \Box_{[0,a]}(p\,\mathcal{U}\,q) \wedge \Diamond_{[a,b]}q.$$

[9] Other possibilities are left-open right-closed intervals with the exclusion of time zero, or a non-bijective representation which may map the algebraic object into two time functions that differ on t_1.

Proof: One direction of the equivalence follows directly from the semantics of timed *until*, so we will consider only the other direction which is proved via the following observations:

1. If $\xi \models \Box_{[0,a]} p \mathcal{U} q$ then p is continuously true throughout $[0, a]$.
2. If $\xi \models \Box_{[0,a]} p \mathcal{U} q$, then $p \mathcal{U} q$ has to hold at a and hence there exists $t' \geq a$ such that q is true at t' and p holds during $[a, t']$.
3. Formula $\Diamond_{[a,b]} q$ requires the existence of $t' \in [a, b]$ such that q holds at t'

Combining these observations we can see that $\xi \models \Box_{[0,a]}(p \mathcal{U} q) \wedge \Diamond_{[a,b]} q$ implies that there exists $t' \in [a, b]$ such that q is true at t' and p holds continuously during $[0, t']$, which is exactly the semantic definition of $p \mathcal{U}_{[a,b]} q$. ▟

For the sake of completeness, let us mention that the special case $p \mathcal{U}_{[0,b]} q$ can be written as $p \mathcal{U} q \wedge \Diamond_{[0,b]} q$, and that $p \mathcal{U}_{[a,\infty)} q$, which is not allowed by our syntax, can be written as $\Box_{[0,a]}(p \mathcal{U} q)$.

3.3 Timed Automata

We use a variant of timed automata which differs slightly from the classical definitions [AD94, HNSY94, Alu99] as it reads multi-dimensional *dense-time* Boolean signals, and outputs Boolean signals. Hence the input and output alphabet letters are associated with *states* rather than with *transitions*. We also extend the domain of clock values to include the special symbol $\perp$ indicating that the clock is currently *inactive*[10] and extend the order relation on $\mathbb{R}_+$ accordingly by letting $\perp < v$ for every $v \in \mathbb{R}_+$. For a set $A \subseteq \mathbb{R}^n$ we use $cl(A)$ to denote its closure (in the topological sense).

The set of valuations of a set $\mathcal{C} = \{x_1, \ldots, x_n\}$ of clock variables, each denoted as $v = (v_1, \ldots, v_n)$, defines the clock space $\mathcal{H} = (\mathbb{R}_+ \cup \{\perp\})^n$. A *configuration* of a timed automaton is a pair of the form (q, v) with q being a discrete state. For a clock valuation $v = (v_1, \ldots, v_n)$, $v + t$ is the valuation $(v'_1, \ldots, v'_n)$ such that $v'_i = v_i$ if $v_i = \perp$ and $v'_i = v_i + t$ otherwise. A *clock constraint* is a conjunction of conditions of the form $x \leq d$, $x < d$, $x \geq d$ or $x > d$ for some integer d.

Definition 2 (Timed Signal Transducer). *A timed signal transducer is a tuple $\mathcal{A} = (\Sigma, Q, \Gamma, \mathcal{C}, \lambda, \gamma, I, \Delta, q_0, F)$ where Σ is the input alphabet, Q is a finite set of discrete states, Γ is the output alphabet and $\mathcal{C}$ is a set of clock variables. The input labeling function $\lambda : Q \to \Sigma$ associates an input letter to every state while the output function $\gamma : Q \to \Gamma$ assigns output letters. The staying condition (invariant) I assigns to every state q a subset I_q of $\mathcal{H}$ defined by a conjunction of inequalities of the form $x \leq d$ or $x < d$, for some clock x and integer d. The transition relation Δ consists of elements of the form (q, g, ρ, q') where q and q' are discrete states, the transition guard g is a subset of $\mathcal{H}$ defined by a clock constraint and ρ is the update function, a transformation of $\mathcal{H}$ defined by an assignment of the form $x := 0$ or*

[10] This is syntactic sugar since clock inactivity in a state can be encoded implicitly by the fact that in all paths emanating from the state, the clock is reset to zero before being tested [DY96].

$x := \bot$. *Finally* q_0 *is the initial state and* $\mathcal{F} \subseteq 2^Q$ *is a generalized Büchi acceptance condition.*

The behavior of the automaton as it reads a signal ξ consists of an alternation between time progress periods where the automaton stays in a state q as long as $\xi[t] = \lambda(q)$ and I_q holds, and discrete instantaneous transitions guarded by clock conditions. Formally, a *step* of the automaton is one of the following:

- A time step: $(q, v) \xrightarrow{\sigma^t/\tau^t} (q, v+t)$, $t \in \mathbb{R}_+$ such that $\sigma = \lambda(q)$, $\tau = \gamma(q)$, and[11] $v + t \in cl(I_q)$.
- A discrete step: $(q, v) \xrightarrow{\delta} (q', v')$, for some transition $\delta = (q, g, \rho, q') \in \Delta$, such that $v \in g$ and $v' = \rho(v)$

A *run* of the automaton starting from a configuration (q_0, v_0) is a finite or infinite sequence of alternating time and discrete steps of the form

$$\xi : \quad (q_0, v_0) \xrightarrow{\sigma_1^{t_1}/\tau_1^{t_1}} (q_0, v_0 + t_1) \xrightarrow{\delta_1} (q_1, v_1) \xrightarrow{\sigma_2^{t_2}/\tau_2^{t_2}} (q_1, v_1 + t_2) \xrightarrow{\delta_2} \cdots,$$

such that $\sum t_i$ diverges. A run is accepting if for every $F \in \mathcal{F}$, the set of time instants in which it visits states in F is unbounded. The input signal carried by the run is $\sigma_1^{t_1} \cdot \sigma_2^{t_2} \cdots$ and the output is $\tau_1^{t_1} \cdot \tau_2^{t_2} \cdots$ The automaton realizes a sequential relation $f_{\mathcal{A}}$ over its input and output alphabets defined by $f_{\mathcal{A}}(\xi) = \xi'$ iff the accepting run induced by the input signal ξ, outputs the signal ξ'.

4 Main Result

In this section we show how to build for every MITL formula φ a property tester, a timed signal transducer $\mathcal{A}_\varphi$ whose associated sequential function coincides with the characteristic function of φ, that is, $f_{\mathcal{A}_\varphi} = \chi^\varphi$. The construction follows the pattern of the untimed one by composing testers corresponding to operators that appear in the formula. The tester for the untimed *until* can be easily adapted to signals by associating input and output symbols with states and removing self loops (see Figure 3). The only missing link is the following proposition.

Proposition 1. *One can construct a timed signal transducer that realizes* $\chi^{\Diamond_{[a,b]}p}$.

The construction used to prove this proposition follows the lines of the untimed one, based on generating output predictions non-deterministically and aborting them when they are contradicted by actual values of p in ξ. Dense time, however poses new problems because the set of potential predictions of bounded duration includes signals with an arbitrary number of switchings between true and false, and such predictions cannot be memorized by a finite-state timed device. We first show in the following lemma, also used in [AFH96], that predictions that switch too frequently cannot be true for any sub formula of type $\Diamond_{[a,b]}p$. A similar property was used in [MNP05] to show that past MITL is deterministic. We use the auxiliary variable u for the output of the tester.

[11] Note that we have chosen I_q to be "backward-closed" so that $v + t \in cl(I_q)$ implies $v + t' \in I_q$ for every t', $0 \leq t' < t$.

Lemma 3. *Let β be a Boolean signal satisfying $\beta = \chi^{\Diamond_{[a,b]}p}(\xi)$ for $0 \leq a < b$ and some arbitrary ξ. Then for any factorization $\beta = v \cdot 0^{r_1} \cdot 1^{r_2} \cdot 0^{r_3} \cdot w$ we have $r_2 \geq b - a$.*

Proof: The following observation concerning the constraints on the values of u and p at every t follows from the definitions: if p holds at $t+b$, u must hold throughout the interval $[t, t+b-a]$. Let $[t_1, t_2)$ be the corresponding interval for 1^{r_2}. Since t_1 is the first instant where u holds, p must start holding at t_1+b and not before that, because otherwise this would imply that u holds before t_1, contrary to our assumptions. Following the observation, u has to hold during the entire interval $[t_1, t_1 + b - a]$. Consequently, $t_2 \geq t_1 + b - a$ and we are done. ▟

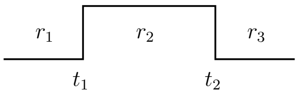


Fig. 4. A signal $\chi^{\Diamond_{[a,b]}p}(\xi)$ for an arbitrary ξ

The importance of this property is that it bounds the variability of any realizable prediction and constrains the relation between the number of changes and the duration of candidate signals. Let $d = b - a$ and $m = \lceil b/d \rceil + 1$. Since every[12] 10 or 01 segment of β has at least d duration, any acceptable prediction of the form $(01)^m$ or $(10)^m$ has a duration beyond b and, hence, its initial segment can be forgotten. Consequently, $2m$ clocks suffice to memorize relevant predictions.

Let us first explain our construction in discrete time where $\Diamond_{[a,b]}$ is just syntactic sugar:

$$\Diamond_{[a,b]}p \equiv \bigvee_{i \in [a,b]} \bigcirc^i p$$

As this operator is bounded by $d = b - a$, a tester will need to remember at most 2^d predictions, which can be encoded explicitly as states that correspond to elements of $\mathbb{B}^d$. For obvious reasons, this approach will not extend to dense time. Instead, we can encode such sequences by the length (duration) of their 0 and 1 segments as is done in data compression. For example, the sequence 00011011 can be encoded as $0^3 1^2 0^1 1^2$. This information can be memorized by an automaton augmented with additional discrete clocks (counters), each reset upon a change between 0 and 1, and incremented at each step. For this example, assuming x_1 is reset at the first 0, y_1 reset at the subsequent occurrence of 1, and so on, we reach the end of the sequence with $x_1 = 8$, $y_1 = 5$, $x_2 = 3$ and $y_2 = 2$. With such a "symbolic" representation, the relevant past predictions at any time instant are identified via conditions on the clocks, which are used to admit or reject actual values of the input.

Our construction does exactly that in dense time. Due to the symbolic encoding of states, it is simpler to decompose the tester into two parts, as depicted in Figure 5: a *prediction generator* (Figure 6) which generates output signals non-deterministically

[12] To be more precise, the first 10 segment can be arbitrarily small.

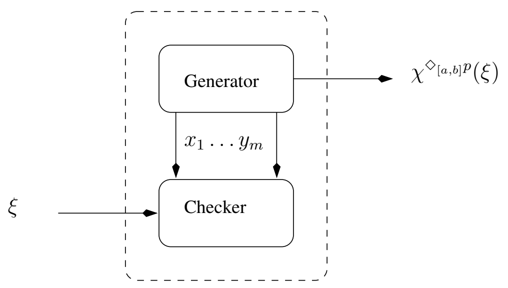


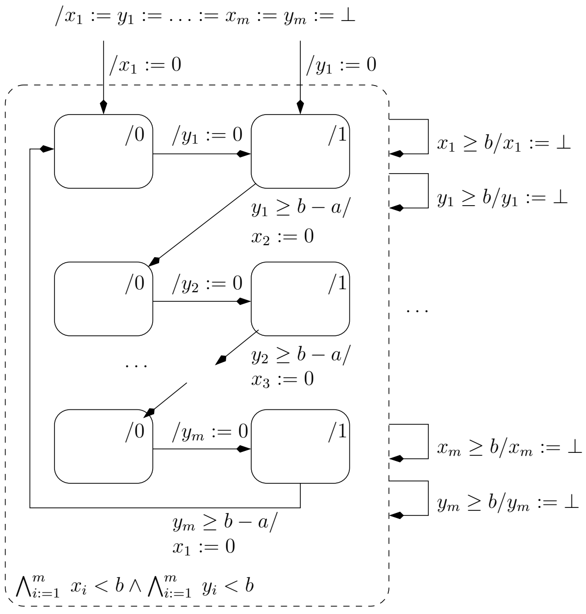
Fig. 5. The architecture of the tester for $\Diamond_{[a,b]}p$

and the *checker* (Figure 7) which checks whether actual inputs confirm these predictions.

The generator simply generates arbitrary Boolean signals subject to the sole restriction of bounded variability, according to Lemma 3. This condition is imposed via the guards of the form $y_i \geq b - a$ on all transitions from 1 to 0. The values of the $2m$ clocks $x_1, \ldots x_m$ and $y_1 \ldots y_m$ represent at time t the form of the prediction segment at the time window $[t - b, t]$. Clocks whose value cross b become inactive and can be re-used to memorize new events. This way the denotation of the clocks shifts circularly, and the value of the active clocks always represents the boundaries of the relevant prediction segments.

To see that the checker indeed aborts all but correct predictions observe a prefix of its behavior as it moves through states $\{s_1, s_2, s_3\}$ trying to match prediction to input (see Figure 8). The negative prediction segment at $[t_0, t_1)$ forbids p anywhere in the interval $t_0 + a, t_1 + b)$ and this is guaranteed by forcing the checker to be at the $\overline{p}$-state s_1 from the time $x_1 \geq a$ until the time $y_1 = b$. At time $t_1 + b$, p must be observed and a transition to s_2 is taken. The positive prediction interval requires p to hold during the interval $[t_1 + b, t_2 + a)$ (which corresponds to the clock conditions $y_1 \geq b$ and $x_2 < a$) except, possibly, for short episodes of $\overline{p}$ that last less than $b - a$ time. This is reflected in the transition from s_2 to s_3 which resets the clock z. If $z = b - a$ and we are still in s_3, the run is aborted. When $x_2 \geq a$ we arrive to a new negative prediction segment and move to the next identical segment of the automaton. This bounded operator requires no Büchi condition, and the proof of Proposition 1 is concluded. ■

To complete the construction for MITL we just need to compose the testers for the propositional, untimed and timed operators according to the structure of the formula. The parallel composition of transducers is fairly standard and we give only the definition of an input-output composition of signal transducers $\mathcal{A}^1 \triangleright \mathcal{A}^2$ where the output of $\mathcal{A}^1$ is the input of $\mathcal{A}^2$. Note that the need for a generalized Büchi condition comes from such a composition of testers for unbounded operators as we need to identify accepting runs of $\mathcal{A}^2$ triggered by outputs of accepting runs of $\mathcal{A}^1$.



$/x_1 := y_1 := \ldots := x_m := y_m := \perp$
$/x_1 := 0$
$/y_1 := 0$
$/0$
$/1$
$/y_1 := 0$
$x_1 \geq b/x_1 := \perp$
$y_1 \geq b/y_1 := \perp$
$y_1 \geq b-a/$
$x_2 := 0$
$/y_2 := 0$
$y_2 \geq b-a/$
$x_3 := 0$
$/y_m := 0$
$x_m \geq b/x_m := \perp$
$y_m \geq b/y_m := \perp$
$y_m \geq b-a/$
$x_1 := 0$
$\bigwedge_{i:=1}^m x_i < b \wedge \bigwedge_{i:=1}^m y_i < b$

Fig. 6. The prediction generator. We use a StateChart-like notation when several states admit the same transition or staying condition.

Definition 3 (I/O Composition). *Let $\mathcal{A}^1 = (\Sigma^1, Q^1, \Gamma^1, \mathcal{C}^1, \lambda^1, \gamma^1, I^1, \Delta^1, q_0^1, F^1)$ and $\mathcal{A}^2 = (\Sigma^2, Q^2, \Gamma^2, \mathcal{C}^2, \lambda^2, \gamma^2, I^2, \Delta^2, q_0^2, F^2)$ be timed signal transducers such that $\Gamma^1 = \Sigma^2$. Their I/O composition is the transducer*

$$\mathcal{A} = \mathcal{A}^1 \triangleright \mathcal{A}^2 = (\Sigma^1, Q, \Gamma^2, \mathcal{C}, \lambda, \gamma, I, \Delta, q_0, F)$$

where

$$Q = \{(q^1, q^2) \in Q^1 \times Q^2 \text{ s.t. } \gamma^1(q^1) = \lambda^2(q^2)\},$$

$\mathcal{C} = \mathcal{C}^1 \cup \mathcal{C}^2$, $\lambda(q^1, q^2) = \lambda^1(q^1)$, $\gamma(q^1, q^2) = \gamma^2(q^2)$, $I_{(q^1,q^2)} = I^1_{q^1} \cap I^2_{q^2}$, $q_0 = (q_0^1, q_0^2)$ and $F = F^1 \cup F^2$. The transition relation Δ is the restriction to Q of the set of all transitions of either of the following forms

$$\begin{array}{l}((q^1, q^2), g^1 \cap g^2, \rho^1 || \rho^2, (q'^1, q'^2)) \\ ((q^1, q^2), g^1 \cap I_{q^2}, \rho^1, (q'^1, q^2)) \\ ((q^1, q^2), I_{q^1} \cap g^2, \rho^2, (q^1, q'^2))\end{array}$$

such that $(q^1, g^1, \rho^1, q'^1) \in \Delta^1$ and $(q^2, g^2, \rho^2, q'^2) \in \Delta^2$.

It is not hard to see that $A^1 \triangleright A^2$ realizes the sequential function obtained by composing the sequential functions realized by $\mathcal{A}^1$ and $\mathcal{A}^2$.

Corollary 1 (Main Result). *MITL formulae can be transformed into timed automata using a simple procedure.*

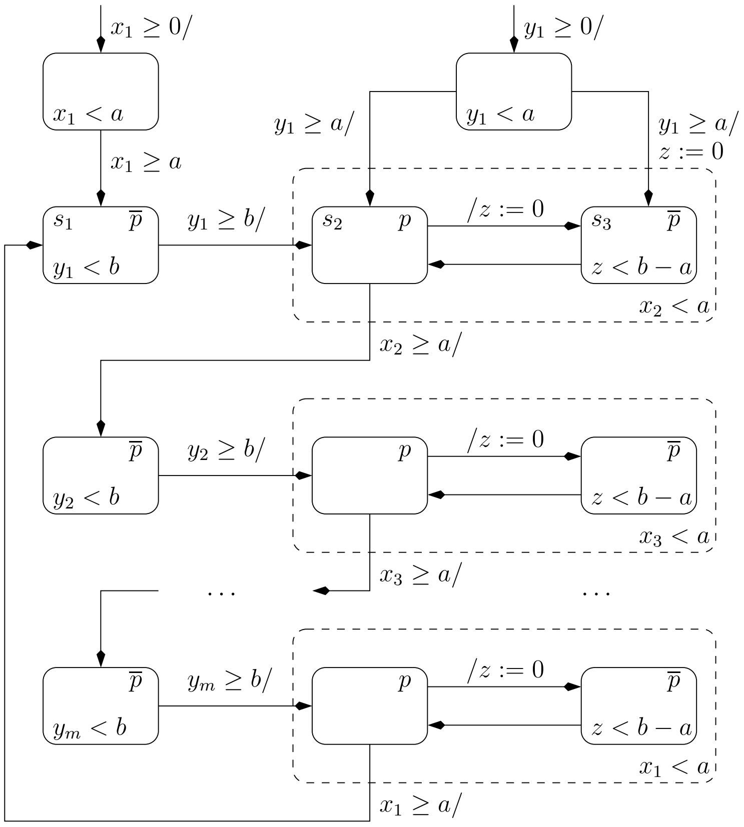


Fig. 7. The automaton for checking predictions

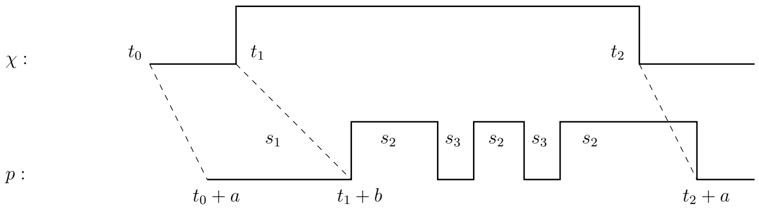


Fig. 8. A behavior of the checker on a prediction-input pair

This construction provides a decision procedure for MITL by emptiness checking. Following the results in [TYB05, Tri06], timed Büchi emptiness checking can be reduced to the search of accepting cycles in the zone-closed simulation graph, which is generated in practice by timed verification tools such as KRONOS [Y97], UPPAAL [LPY97] or IF [BGM02]. The test for emptiness can be conducted efficiently on-the-fly using standard maximal SCC or double DFS algorithms [CVWY92]. Due to the generalized Büchi conditions, our implementation is based on the nested DFS algorithm of [Tau06].

Given a real-time system S modeled by a timed automaton $\mathcal{A}_S$ and an MITL formula φ, the model checking problem, that is, the language inclusion $L(\mathcal{A}_S) \subseteq L(\varphi)$ between the possible behaviors of $\mathcal{A}_S$ and the behaviors satisfying φ, reduces to the timed Büchi emptiness check on the product automaton $\mathcal{A}_S \times \mathcal{A}_{\neg\varphi}$, where $\mathcal{A}_{\neg\varphi}$ is obtained from the negated MITL formula φ using the modular construction described in this paper.

5 Discussion

Our definitions deviate from those of [AFH96] in the following respects:

1. We disallow signals that admit punctuality;
2. We restrict the temporal modalities to closed intervals;
3. We modify the semantics of $p\,\mathcal{U}\,q$ to require a “handshake” moment where both p and q hold.

The restriction to non-punctual signals is very reasonable from a semantic point of view and constitutes the natural choice for signals. The two other modifications are consequences of this choice as we want the output of the testers to be valid signals as well. The main limitation of this logic is the inability to specify *events* (or the *rising* and *falling* of a signal) which prevents, for example, expressing properties such as *bounded variability*. The extension of the logic to express subsets of the more general signal-event monoid [ACM02] is a topic for future research.

Let us also remark that giving preference to results proved with respect to the most general existing definitions is a mathematicians attitude that should not be adopted without a critical examination. Such an attitude can be counter-productive in young domains where “classical” results and definitions are only decade old, and the most appropriate formalization has not yet stabilized.

Acknowledgment. We thank anonymous referees of several conferences for their remarks and suggestions.

References

[Alu99] R. Alur, Timed Automata, *CAV’99*, LNCS 1633, 8–22, Springer, 1999.

[AD94] R. Alur and D.L. Dill, A Theory of Timed Automata, *Theoretical Computer Science* **126**, 183–235, 1994.

[AFH96] R. Alur, T. Feder, and T.A. Henzinger, The Benefits of Relaxing Punctuality, *Journal of the ACM* **43**, 116–146, 1996 (first published in *PODC’91*).

[AH92a] R. Alur and T.A. Henzinger, Logics and Models of Real-Time: A Survey, *REX Workshop, Real-time: Theory in Practice*, 74–106. LNCS 600, 1992.

[AH92b] R. Alur and T.A. Henzinger, Back to the Future: Towards a Theory of Timed Regular Languages, *FOCS’92*, 177-186, 1992.

[A04] E. Asarin, Challenges in Timed Languages, *Bulletin of EATCS* 83, 2004.

[ACM02] E. Asarin, P. Caspi and O. Maler, Timed Regular Expressions, *The Journal of the ACM* **49**, 172–206, 2002.

[BGM02] M. Bozga, S. Graf and L. Mounier, IF-2.0: A Validation Environment for Component-Based Real-Time Systems, *CAV'02*, 343–348, LNCS 2404, 2002.

[BCM$^+$92] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill and L.J. Hwang, Symbolic Model Checking: 10^{20} States and Beyond, *Information and Computation* **98**, 140–170, 1992.

[CGH94] E.M. Clarke, O. Grumberg and K. Hamaguchi, Another look at LTL Model Checking, *CAV'94*, 415–427, LNCS 818, 1994.

[CGP99] E.M. Clarke, O. Grumberg, and D.A. Peled, *Model Checking*. The MIT Press, 1999.

[CVWY92] C. Courcoubetis, M.Y. Vardi, P. Wolper and M. Yannakakis, Memory-Efficient Algorithms for the Verification of Temporal Properties, *Formal Methods in System Design* **1**, 275–288, 1992.

[DY96] C. Daws and S. Yovine, Reducing the Number of Clock Variables of Timed Automata, *RTSS'96*, 73–81, 1996.

[DT04] D. D'Souza and N. Tabareau, On Timed Automata with Input-determined Guards, *FORMATS/FTRTFT'04*, 68–83, LNCS 3253, 2004.

[GO01] P. Gastin and D. Oddoux, Fast LTL to Büchi Automata Translation, *CAV'01*, 53–65, LNCS 2102, 2001.

[GPVW95] R. Gerth, D.A. Peled, M.Y. Vardi and P. Wolper, Simple On-the-fly Automatic Verification of Linear Temporal Logic, *PSTV*, 3–18, 1995.

[Hen98] T.A. Henzinger, It's about Time: Real-time Logics Reviewed, *CONCUR'98*, 439–454, LNCS 1466, 1998.

[HR02] K. Havelund and G. Rosu, Synthesizing Monitors for Safety Properties, *TACAS'02*, 342–356, LNCS 2280, 2002.

[HFE04] J. Havlicek, D. Fisman and C. Eisner, Basic results on the semantics of Accellera PSL 1.1 foundation language, *Technical Report 2004.02*, Accelera, 2004.

[HNSY94] T. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine, Symbolic Model-checking for Real-time Systems, *Information and Computation* **111**, 193–244, 1994.

[HRS98] T.A. Henzinger, J.-F. Raskin, and P.-Y. Schobbens, The Regular Real-time Languages, *ICALP'98*, 580–591, LNCS 1343, 1998.

[HR04] Y. Hirshfeld and A. Rabinovich, Logics for Real Time: Decidability and Complexity, *Fundamenta Informaticae* **62**, 1–28, 2004.

[KP05] Y. Kesten and A. Pnueli, A Compositional Approach to CTL* Verification, *Theoretical Computer Science* **331**, 397–428, 2005.

[Koy90] R. Koymans, Specifying Real-time Properties with Metric Temporal Logic, *Real-time Systems* **2**, 255–299, 1990.

[KCV04] A. Kumari B. Cohen and S. Venkataramanan, *Using PSL/Sugar for Formal and Dynamic Verification*, VhdlCohen Publishing, 2004.

[LPY97] K.G. Larsen, P. Pettersson and W. Yi, Uppaal in a Nutshell, *International Journal of Software Tools for Technology Transfer* **1** 134–152, 1997.

[MMP92] O. Maler, Z. Manna, and A. Pnueli, From Timed to Hybrid Systems, *Real-Time: Theory in Practice*, 447–484, LNCS 600, 1992.

[MN04] O. Maler and D. Nickovic, Monitoring Temporal Properties of Continuous Signals, *FORMATS/FTRTFT'04*, 152–166, LNCS 3253, 2004.

[MNP05] O. Maler, D. Nickovic and A. Pnueli, Real Time Temporal Logic: Past, Present, Future, *FORMATS'05*, 2–16, LNCS 3829, 2005.

[MP91] Z. Manna and A. Pnueli, *Temporal Verification of Reactive Systems: Specification*, Springer, 1991.

[MP95] Z. Manna and A. Pnueli, *Temporal Verification of Reactive Systems: Safety*, Springer, 1995.

[OW05] J. Ouaknine and J. Worrell, On the Decidability of Metric Temporal Logic, *LICS'05*, 188–197, 2005.

[RSH98] J.-F. Raskin, P.Y. Schobbens and T.A. Henzinger, Axioms for Real-Time Logics, *Concur'98*, 219–236, 1998.

[SB00] F. Somenzi and R. Bloem, Efficient Büchi automata from LTL formulae, *CAV'00*, 248–263, LNCS 1855, 2000. 1855.

[Tau06] H. Tauriainen, Nested Emptiness Search for Generalized Bchi automata *Fundamenta Informaticae*, **70**, 127–154, 2006.

[Tri06] S. Tripakis, Checking Timed Büchi Automata Emptiness on Simulation Graphs, *Technical Report 2006-1*, Verimag, 2006.

[TYB05] S. Tripakis, S. Yovine and A. Bouajjani, Checking Timed Büchi Automata Emptiness Efficiently, *Formal Methods in System Design* **26**, 267-292, 200.

[Var96] M.Y. Vardi, Alternating Automata and Program Verification, *Current Trends in Computer Science*, 267–278, LNCS 1000, 1996.

[VW86] M.Y. Vardi and P. Wolper, An Automata-theoretic Approach to Automatic Program Verification, *LICS'86*, 322–331, 1986.

[Y97] S. Yovine, Kronos: A Verification Tool for Real-time Systems, *International Journal of Software Tools for Technology Transfer* **1**, 123–133, 1997.

Adding Invariants to Event Zone Automata

Peter Niebert and Hongyang Qu

Laboratoire d'Informatique Fondamentale de Marseille, Université de Provence
39, rue Joliot-Curie F-13453 Marseille Cedex 13
{niebert, hongyang}@cmi.univ-mrs.fr

Abstract. Recently, a new approach to the symbolic model checking of timed automata based on a partial order semantics was introduced, which relies on *event zones* that use vectors of event occurrences instead of *clock zones* that use vectors of clock values grouped in polyhedral clock constraints. Symbolic state exploration with event zones rather than clock zones can result in significant reductions in the number of symbolic states explored. In this work, we show how to extend the event zone approach to networks of automata with local state invariants, an important feature for modeling complex timed systems. To avoid formalizing local states, we attach to each transition an urgency constraint, that allows to code local state invariants. We have integrated the extension into a prototype tool with event zones and reported very promising experimental results.

1 Introduction

Timed automata [1] are a powerful tool for the modeling and the analysis of timed systems. They extend classical automata by *clocks*, continuous variables "measuring" the flow of time. A state of a timed automaton is a combination of its discrete control location and the *clock values* taken from the real domain. While the resulting state space is infinite, *clock constraints* have been introduced to reduce the state spaces to a finite set of equivalence classes, thus yielding a finite (although often huge) symbolic state graph on which reachability and some other verification problems can be resolved.

While the theory, algorithms [9,10] and tools [13,3] for timed automata represent a considerable achievement (and indeed impressing industrial applications have been treated), the combinatorial explosion particular to this kind of modeling and analysis – sometimes referred to as "clock explosion" (at the same time similar to and different from classical "state explosion") – remains a challenge for research and practice. Despite the theoretical limits (for a PSPACE complete problem), great effort has been invested into the optimization of the symbolic approach (see e.g. [8,4,7,2]).

Event zone automata [11] are a partial order based approach to reduce one source of clock explosion, interleaving semantics. Partial order methods basically try to avoid redundant research by exploiting knowledge about the structure of the reachability graph, in particular *independence* of pairs of transitions of loosely related parts of a complex system. Such pairs a and b commute, i.e. a state s allowing a sequence ab of transitions to state s' also allows ba and this sequence also leads to a state s'' that has the same control location as s'. However, this kind of commutation is easily lost in classical symbolic analysis algorithms for timed automata, which represent sets of possible clock values by symbolic states: Consider two "independent" actions a resetting clock $x := 0$, and b resetting clock $y := 0$. Executing a first and then b means that afterwards (time may have elapsed) $x \geq y$ whereas executing b first and then a implies that afterwards $x \leq y$. The result of this is that in the algorithms used in tools like UppAal [3] and Kronos [13], ab and ba lead to incomparable symbolic states.

The *event zone* approach successfully avoids such zone splitting, while preserving most algorithmic possibilities offered by clock zone automata. The underlying notion of independence is based on reading and writing of shared variables: If for some clock x, transition a resets x and transition b has a condition on x or if both a and b reset x, then they must be dependent.

However, being based on Mazurkiewicz trace theory and thus more action than state oriented, [11] did not address the question of state invariants (urgency constraints), an important modeling feature used in tools like UppAal. Unlike transition guards that give conditions on the interval in which a transition can be executed, state invariants are conditions attached to states (locations) that limit the allowed stay in the state, somewhat like a residence permit : before a violation of the invariant, a transition must be taken. Typically, allowed invariants are conjunctions of upper bounds on clock values.

The contribution of this work is *completing the approach of [11]* by an integration of local state invariants. With this addition, the full class of timed automata as present in tools like UppAal as networks of timed automata can be analyzed. This generality also distinguishes our work from many other publications in the same area, which exclude global clocks (e.g. [6] or recently [12]).

A technical difficulty to overcome for this is the formalization of invariants: The event zone approach assumes only a single timed automaton with structural properties (diamonds) but does not as such expose "local states". It was not obvious, how state based invariants could be added to that framework. Our solution attaches urgency constraints to transitions, and then requires a consistency between these constraints and *global* state invariants. The urgency constraints allow a way of coding of local state invariants: One attach to a transition the local state invariant of its original automaton. Urgency constraints on transitions have been introduced before, our contribution is to use them (rather than state based urgency) for independence analysis.

Then we show how to extend the framework of [11] to the timed automata with these invariants. A key auxiliary tool, a "separator action" \$, in [11] is transformed into a "snapshot action": A snapshot is an artificial action that separates past and future and at which the global state invariant must be satisfied. The snapshot action establishes a link between partial order semantics and interleaving semantics. In terms of classical algorithms for timed automata, adding the snapshot action at the end of a sequence is like a transformation of a zone by passage of time (limited by invariants). The key to avoid zone splitting is that the snapshot is not introduced into the zones we explore, but is used to stop exploration.

For all actions other than the snapshot, the invariants to be satisfied are local: We only require satisfaction of its invariant whenever a clock is reset! It turns out that this approach allows to preserve completely the notion of independence from [11].

In this extended abstract, we concentrate on the formalization of the local invariants and the snapshot action and show the relevant properties. Moreover, we have extended the algorithm in [11] and added it to a new prototype tool POEM (Partial Order Environment of Marseille) based on the code of ELSE [14] to give an interesting (albeit not exhaustive) experimental comparison with a recent build of UppAal (v3.6 beta 1).

The paper is structured as follows: In Section 2, we informally explain the use and problems related to local state invariants. In Section 3, we formalize timed automata with invariants attached to transitions and we introduce the notion of a run and the language of a timed automaton. In Section 4, we develop the notion of independence in the context of the automata introduced in Section 3 and show the main fundamental results of this paper, the consistency of the relaxed semantics with the standard semantics. In Section 5, we give hints on how event zone based exploration tools for timed automata with invariants actually work. In Section 6, we give some experimental results.

2 State Invariants and Reachability in Timed Automata

Many verification problems, notably *safety properties*, of timed automata can be coded as (non-)reachability of a location (discrete state), and we concentrate on such specifications in this section. This does not exclude the extension of our ideas to more sophisticated properties like emptiness of Büchi automata.

A state invariant is a downward closed condition on the clocks, a conjunction of constraints like "$x < 5$" or "$y \leq 7$" or even "$z \leq 0$", stating, how long at most the "stay" in a state is "allowed", i.e. the stay in the state must not extend to a moment where the invariant is violated. In a run, a transition must thus occur before the expiry of the state invariant.

For reachability in timed automata, state invariants do not fundamentally add expressive power! We can eliminate invariants without touching the set of reachable states by two transformation steps:

- Strengthen the conditions of each outgoing transition by the invariant without changing the possible behaviour of the automaton; likewise strengthen the conditions so to guarantee that the invariants of the target states are respected at entry.
- Remove the invariants from the states after having strengthened the transitions. While this change may allow partial runs where the automaton stays beyond invariant in a state, it cannot add to the reachable states.

However, the situation changes when we consider the use of invariants in modelling *networks of timed automata*. Consider the following timed system in Figure 1: A multi lane highway with cars on each lane and a rabbit who wants to cross. The rabbit has some freedom of going slower or faster and so do the cars. Can - with the help of the car drivers - the rabbit reach the other side of the highway alive? To model this by a network of timed automata, we choose to model the highway as a checker board of lanes and positions on lanes as indicated in the picture, cars move in the horizontal direction and the rabbit in the vertical direction. Each car and the rabbit is realised by an individual automaton. The freedom of going slower or faster is modeled by a time interval in which the rabbit can advance by one lane and an interval in which the car can advance for one unit length on a discretized highway. If a car and the rabbit are in the same field of the checker board at the same time, an accident occurs. The UppAal model is indicated by showing the automaton for the rabbit and an instance of the automata for cars.

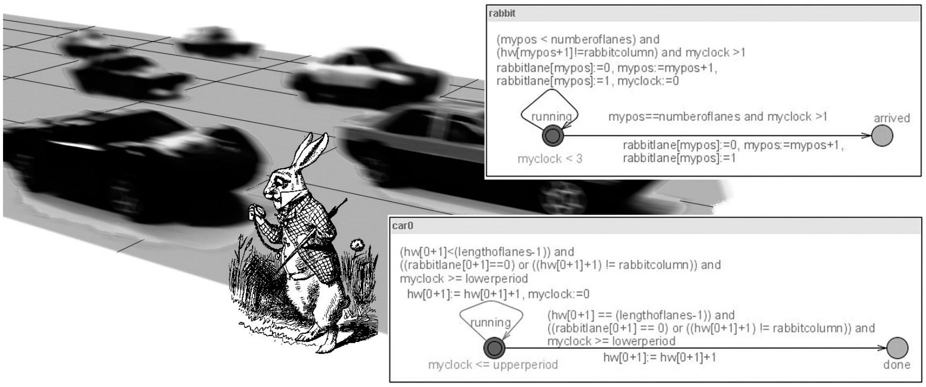

rabbit
(mypos < numberoflanes) and
(hw[mypos+1]!=rabbitcolumn) and myclock >1
rabbitlane[mypos]:=0, mypos:=mypos+1,
rabbitlane[mypos]:=1, myclock:=0
running
mypos==numberoflanes and myclock >1
arrived
rabbitlane[mypos]:=0, mypos:=mypos+1,
rabbitlane[mypos]:=1
myclock < 3
car0
(hw[0+1]<(lengthoflanes-1)) and
((rabbitlane[0+1]==0) or ((hw[0+1]+1) != rabbitcolumn)) and
myclock >= lowerperiod
hw[0+1]:= hw[0+1]+1, myclock:=0
running
(hw[0+1] == (lengthoflanes-1)) and
((rabbitlane[0+1] == 0) or ((hw[0+1]+1) != rabbitcolumn)) and
myclock >= lowerperiod
hw[0+1]:= hw[0+1]+1
myclock <= upperperiod
done

Fig. 1. A real life race condition and its UppAal model with invariants

Without invariants, the cars or the rabbit could just stop (not take a advancing transition), in which case it is obvious that the rabbit will reach its target safely. Hence, the invariants are essential for correct modeling in that the invariants enforce a more or less synchronous progress of the cars and the rabbit, so that everyone has to choose their speed to allow the rabbit to pass unharmed.

A naïve approach to eliminate the invariants is to apply the two step transformation to automata (on-the-fly or offline) and after that have an Alur-Dill automaton without invariants to which we could apply the algorithm shown in [11]. However, the global invariant limits the time progress for the local clock of each car, thus the rewriting will essentially render all transitions dependent! This is precisely a source of combinatorial explosion in this example.

But this additional dependency is not unavoidable, the "independence" of the cars in the example is quite obvious: They don't interfere with each other, they just interfere with the rabbit. In the rest of the article we show that there is a better way of dealing with local invariants, preserving full independence.

3 Timed Automata with Transition Invariants

In this section, we introduce basic notions of timed words, timed languages, as well as their finite representation by timed automata [1].

For an alphabet Σ of actions denoted by $a, b, c \ldots$, let $\$ \notin \Sigma$ be a special symbol, the *snapshot action*[1], and let $\Sigma_\$:= \Sigma \uplus \{\$\}$ denote the extension of Σ by \$. Σ^* (or $\Sigma_\$^*$) is the set of finite sequences $a_1 \ldots a_n$ called words, with ϵ the empty word. The length n of a word $a_1 \ldots a_n$ is denoted by $|a_1 \ldots a_n|$. A *timed word* is a sequence $(a_1, \tau_1) \ldots (a_n, \tau_n)$ of elements in $(\Sigma_\$ \times \mathbb{R}^+)^*$, with $\mathbb{R}^+$ the set of non-negative reals, the τ_i's are *time stamps.* For convenience, we set $\tau_0 = 0$ to be an additional time stamp for the beginning. A timed word is *normal* if $\tau_i \leq \tau_j$ for $i \leq j$ as in $(a, 3.2)(c, 4.5)(b, 6.3)$ whereas $(a, 3.2)(c, 2.5)(b, 6.3)$ is not normal. Normal timed words represent temporally ordered sequences of events and serve as standard semantics of timed automata in the literature.

In timed systems, events can occur only if certain time constraints are satisfied. In timed automata, a finite set of real valued[2] variables X, called *clocks*, are used to express the time constraints between an event that resets a clock and another event that refers to the clock value at the time of its occurrence. The clock constraints permitted here are conjunctions of *atomic clock constraints*, comparisons between a clock and a numerical constant. To preserve decidability, constants are assumed to be positive rationals and for simplicity in $\mathbb{N}$, the set of natural numbers. For a set of clocks X, the set $\Phi(X)$ of clock constraints ϕ is formally defined by the grammar $\phi := \text{true} | x \bowtie c \; | \phi_1 \wedge \phi_2$, where x is a clock in X, $\bowtie \in \{<, \leq, >, \geq\}$ and c is a constant in $\mathbb{N}$ (true is for transitions without conditions), moreover excluding the trivially false combination "< 0". Another way of looking at clock constraints is sets of atomic constraints that must all be satisfied. A subset of clock constraints result from the restriction to upper bounds $\lhd \in \{<, \leq\}$, so let $\Phi_{upper}(X) \subseteq \Phi(X)$ be defined according to the grammar $\phi := \text{true} | x \lhd c \; | \phi_1 \wedge \phi_2$.

A *clock valuation* $v : X \to \mathbb{R}$ is a function that assigns a real number to each clock. We denote by $v + \tau$ the clock valuation that translates all clock $x \in X$ synchronously by τ such that $(v + \tau)(x) = v(x) + \tau$. For a subset C of clocks, $v[C \leftarrow 0]$ denotes the clock valuation with $v[C \leftarrow 0](x) = 0$ if $x \in C$ and $v[C \leftarrow 0](x) = v(x)$ if $x \notin C$, i.e. the valuation where the clocks in C are reset to 0. The satisfaction of the clock constraint ϕ by the clock valuation v, i.e. the fact that all atomic constraints are satisfied when substituting $v(x)$ for x, is denoted by $v \models \phi$.

In this paper, we use classical timed automata semantics defined in [1]. However, in order to be able to apply partial order reduction to verification, we extend the definition of timed automata as follows:

[1] The rôle of which will become clear later.

[2] For normal timed words *positive real values* would suffice, see Remark 2.

1. We assume at the origin of our timed automaton the interleaving semantics of a network of timed automata, where states have local invariants.
2. To make the local state invariants accessible in a global automaton (a product of the local automata), we add local invariants to the guards of transitions, as *transition invariants.*
3. We add a snapshot action to each global state as a self loop. The guard of a snapshot action is *true.* In addition, we use global state invariants as transition invariants of snapshot actions.
4. While we assume that the global invariant of a state is the conjunction of local invariants, we assume in our formalization global invariants for each state that have to meet certain consistency properties concerning the transition invariants.

It is easy to see that the above extension does not change the semantics of timed automata. Now we give a formal definiton for the extended timed automata.

Definition 1. *Given an alphabet $\Sigma_\$$ and a set of clocks X, a* timed automaton with transition invariants *is a quintuple $\mathcal{A} = (\Sigma_\$, S, s_0, \rightarrow, Inv, F)$ where S is a finite set of locations, $s_0 \in S$ is the initial location, $F \subseteq S$ is the set of final locations and $\rightarrow \subseteq S \times [\Sigma_\$ \times \Phi(X) \times \Phi_{upper}(X) \times 2^X] \times S$ is a set of transitions, $Inv : S \longrightarrow \Phi_{upper}(X)$ is an assignment of* clock invariants *to locations. For a transition $(s, a, \phi, \psi, C, s') \in\rightarrow$, we write $s \xrightarrow{(a,\phi,\psi,C)} s'$, and call a the* label *of the transition, and ψ the invariant of the transition or the transition invariant.*

For a conjunction of constraints ϕ, let $\phi(x)$ denote the restriction of ϕ to constraints concerning the clock x. The following rules must be satisfied for all transitions $(s, a, \phi, \psi, C, s')$:

1. *$Inv(s) \Rightarrow \psi$, i.e. the global state invariant implies the transition invariant;*
2. *For all $x \notin C$ we have $Inv(s)(x) \wedge \phi(x) \Longrightarrow Inv(s')(x)$;*
3. *For all $x \notin C$ we have $Inv(s')(x) \wedge \psi(x) \Longrightarrow Inv(s)(x)$.*

A *state* (s, v) consists of a location s and a clock valuation v. A snapshot action is of the form $(s, \$, true, Inv(s), \emptyset, s)$. Here s is a global location and $Inv(s)$ is the global state invariant, i.e. we require that the snapshot action verifies the global state invariant.

The intuition of the invariant constraint ψ of a transition is to result from the local state invariant of the predecessor state of one automaton in a network. Hence, the first condition says that the global state invariant is (at least as strong as) the conjunction of the invariants of the outgoing transitions. Indeed, it is the case according to the definition of the global state invariant.

Remark 1. A timed automaton is *action deterministic* if for two transitions $s \xrightarrow{(a,\phi_1,\psi_1,C_1)} s_1$ and $s \xrightarrow{(a,\phi_2,\psi_2,C_2)} s_2$, we have that $\phi_1 = \phi_2$, $\psi_1 = \psi_2$, $C_1 = C_2$ and $s_1 = s_2$. Similarly, we call the timed automaton *constraint consistent* if actions determine uniquely clock constraints and resets, i.e. for each pair of transitions $(s_1, a, \phi, \psi, C, s_2)$ and $(s'_1, a, \phi', \psi, C', s'_2)$ with the same action, we have $\phi = \phi'$, $\psi = \psi'$ and $C = C'$. In that case, given an action a, the unique clock constraint, invariant and reset are denoted by ϕ_a, ψ_a and C_a respectively. In this paper, we will *only consider timed automata that are action deterministic and constraint consistent.* This is no restriction to applications, as this can be easily achieved by renaming (a more detailed discussion is in [11]).

Figure 2 (produced by UppAal) shows a system consisting of two automata. Initial locations are s0 and w0, and final locations are s0, s2, s3 and w0. Clocks are x, y, and z. States invariants are labeled in boldface, e.g., "y<=9" is the variant for local state s1 and "y<=9 ∧ z<=4" is the global variant for (s1,w1).

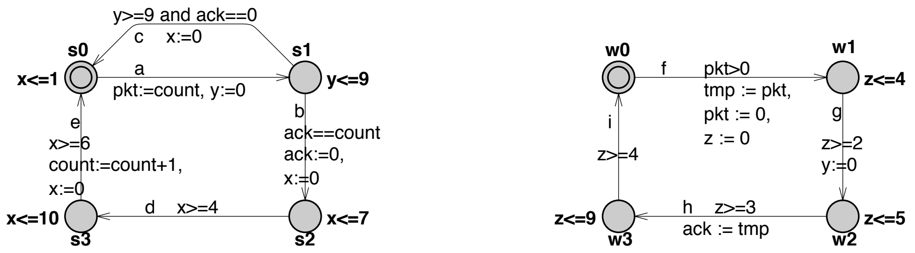


Fig. 2. The example

For our formal development, we introduce three distinct notions of sequences of execution: *paths* (ignoring time constraints), *runs* (paths with time stamps respecting the time constraints), *normal runs* (furthermore the time stamps respect the progress of time):

Definition 2. *A* path *in $\mathcal{A}$ is a finite sequence $s_0 \xrightarrow{(a_1,\phi_1,\psi_1,C_1)} s_1 \ldots \xrightarrow{(a_n,\phi_n,\psi_n,C_n)}$ s_n of consecutive transitions $s_{i-1} \xrightarrow{(a_i,\phi_i,\psi_i,C_i)} s_i$. The word $a_1 \ldots a_n \in \Sigma_\$^*$ of transition labels is called the* path labeling. *If $a_n = \$$ (final snapshot) and $s_n \in F$, the path is said to be* accepted. *The set of labelings of accepted paths is called the* untimed language *of $\mathcal{A}$ and denoted $L(\mathcal{A})$.*

Definition 3. *A* run *of a timed automaton is a path extended by time stamps for the transition occurrences satisfying clock constraints and resets:* $(s_0, v_0) \xrightarrow{(a_1,\phi_1,\psi_1,C_1),\tau_1} (s_1, v_1) \ldots \xrightarrow{(a_n,\phi_n,\psi_n,C_n),\tau_n} (s_n, v_n)$ *where $(a_1, \tau_1)\ldots(a_n, \tau_n)$ is a timed word and $(v_i)_{0\leq i\leq n}$ are clock valuations defined by:*

1. $v_0(x) = 0$ *for all* $x \in X$, $\tau_0 = 0$
2. $v_{i-1} + (\tau_i - \tau_{i-1}) \models \phi_i \wedge \psi_i$
3. $v_{i-1} + (\tau_i - \tau_{i-1}) \models Inv(s_{i-1})(x)$ *for* $x \in C_i$
4. $v_i = (v_{i-1} + (\tau_i - \tau_{i-1}))[C_i \leftarrow 0]$.

Note that the above conditions imply for $a_i = \$$ that $v_i \models Inv(s_i)$, since $s_i = s_{i-1}$. The timed word $(a_1, \tau_1) \ldots (a_n, \tau_n)$ is the timed labeling *of the run. The run is* accepted by $\mathcal{A}$ *if $a_n = \$$, $s_n \in F$.*

Definition 4. *A* normal run *is a run such that its timed labeling* $(a_1,\tau_1)\dots(a_n,\tau_n)$ *is a normal timed word i.e.* $\tau_1 \leq \tau_2 \leq \dots \leq \tau_n$.

Remark 2. It is straightforward to see that for normal runs the valuations always produce positive values: Clocks are either reset to 0 or the translations $v + (\tau_i - \tau_{i-1})$ increase the values since $\tau_i \geq \tau_{i-1}$. In non-normal runs, this need not be the case.

Proposition 1. *In a normal accepted run it holds for all intermediate states* s_i *that* $v_i + (\tau_{i+1} - \tau_i) \vDash Inv(s_i)$*, i.e. global invariants are never violated.*

Proof. The proof is by induction on $n-k$, i.e. on the distance from the last state in the sequence.

The basis is to prove that the proposition holds for $k = 1$. Since $a_n = \$$, $s_n = s_{n-1}$ and $\psi_n = Inv(s_n) = Inv(s_{n-1})$. The second condition implies in turn that $v_{n-1} + (\tau_n - \tau_{n-1}) \vDash \psi_n$.

For the induction step, assume that for $k \geq 1$, the proposition is true, i.e., $v_{n-k} + (\tau_{n-k+1} - \tau_{n-k}) \vDash Inv(s_{n-k})$. For the case $k = k+1$, $v_{n-k-1} + (\tau_{n-k} - \tau_{n-k-1}) \vDash \phi_{n-k} \wedge \psi_{n-k}$ and $v_{n-k-1} + (\tau_{n-k} - \tau_{n-k-1}) \vDash Inv(s_{n-k-1})(x)$ if $x \in C_{n-k-1}$. For $y \notin C_{n-k-1}$, $v_{n-k-1} + (\tau_{n-k} - \tau_{n-k-1}) \vDash Inv(s_{n-k})(y) \wedge \psi_{n-k}$ implies $v_{n-k-1} + (\tau_{n-k} - \tau_{n-k-1}) \vDash Inv(s_{n-k-1})(y)$. Therefore, $v_{n-k-1} + (\tau_{n-k} - \tau_{n-k-1}) \vDash Inv(s_{n-k-1})$, which means the proposition holds for $k+1$. □

The *timed language* $L_T(\mathcal{A})$ of $\mathcal{A}$ is the set of *normal* timed words $(a_1,\tau_1)\dots(a_n,\tau_n)$, such that accepted by $\mathcal{A}$. The path labeling $a_1 \dots a_n$ is said to be *realizable* if for some time stamps τ_i the normal timed word $(a_1,\tau_1)\dots(a_n,\tau_n)(\$,\tau)$ (then called the normal realization of $a_1 \dots a_n$) is the labeling of a normal run. The language of realizable words that are the labeling of an *accepted run* is denoted by $L_N(\mathcal{A})$.

For instance, in Figure 2 (a,0.5)(f,2.5)(g,4.6)(h,7)(i,9.3)(b,9.4)$\in L_T(\mathcal{A})$ is a normal realization of the path labeling afghib, hence afghib$\in L_N(\mathcal{A})$.

4 Independence for Timed Automata

To model concurrency, we use an independence relation between actions such that actions are independent when the order of their occurrence is irrelevant. Formally, an *independence relation* I for an (action deterministic and constraint consistent) timed automaton $\mathcal{A} = (\Sigma_\$, S, s_0, \rightarrow, Inv, F)$ is a symmetric and irreflexive relation $I \subseteq \Sigma \times \Sigma$ such that the following two properties hold for any two $a, b \in \Sigma$ with $a\ I\ b$:

(i) $s \xrightarrow{(a,\phi_a,\psi_a,C_a)} s_1 \xrightarrow{(b,\phi_b,\psi_b,C_b)} s_2$ implies $s \xrightarrow{(b,\phi_b,\psi_b,C_b)} s_1' \xrightarrow{(a,\phi_a,\psi_a,C_a)} s_2$ for some location s_1'

(ii) $C_a \cap C_b = \emptyset$ and no clock x in C_b belongs to an atomic clock constraint $x \bowtie c$ of $\phi_a \wedge \psi_a$ and conversely no clock x in C_a belongs to an atomic clock constraint $x \bowtie c$ of $\phi_b \wedge \psi_b$.

We also use the dependence relation $D = \Sigma \times \Sigma - I$, which is reflexive and symmetric.

We extend the independence relation to $\Sigma_\$$ by setting $\$Db$ for all $b \in \Sigma_\$$, i.e. we *define* the snapshot to be dependent of every other action. This obviously meets conditions (i) and (ii).

Intuitively, condition (ii) arises from the view of clocks as shared variables in concurrent programming: An action resetting a clock is writing it whereas an action with a clock constraint on this clock is reading it. The restriction states that two actions are dependent if both are writing the same variable, or one is writing a variable and the other one is reading it. In pratice, we extend timed automata to allow testing program variables in guards of transitions, and assigning values to program variables by transitions.

Since $I = \emptyset$ trivially meets (i) and (ii) such a relation always exists. Computing a good (the larger, the better) I meeting (i) and (ii) is a matter of static analysis and is typically done on the level of a network *before* constructing the product timed automaton: Sufficient criteria for (i) may require that two transitions originate from distinct components and do not have conflicts around shared variables and do not synchronize on the same channels. For instance, (b,f) is in the independence relation for the timed automata of Figure 2, while (b,g) in the dependence relation because after we put the state invariant y<=9 to be the transition invariant of b, b reads the clock y and g writes it.

The *Mazurkiewicz trace equivalence* associated to the independence relation I is the least congruence $\simeq$ over Σ^* such that $ab \simeq ba$ for any pair of independent actions $a\ I\ b$. A *trace* $[u]$ is the congruence class of a word $u \in \Sigma^*$. By definition, two words are equivalent with respect to $\simeq$ if they can be obtained from each other by a finite number of exchanges of adjacent independent actions. E.g., acf $\simeq$ afc for Figure 2, but acf $\not\simeq$ fac (a and f are dependent). In other words, this permutation of actions between two equivalent words lets the relative order of occurrences of dependent actions unchanged, formally:

Lemma 1. *Let I be an independence relation, $\simeq$ the induced Mazurkiewicz trace equivalence and $a_1 \ldots a_n \simeq b_1 \ldots b_n$ be two equivalent words. There exists a uniquely determined permutation $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$ such that $a_i = b_{\pi(i)}$ and for $a_i\ D\ a_j$ we have $i < j$ iff $\pi(i) < \pi(j)$.*

Conversely, let $a_1 \ldots a_n$ be a word and $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$ be a permutation of indices such that for each pair i, j $a_i\ D\ a_j$ we have $i < j$ iff $\pi(i) < \pi(j)$. Then $a_{\pi(1)} \ldots a_{\pi(n)} \simeq a_1 \ldots a_n$.

Proof. By induction on the number of exchanges. □

For convenience in applications to timed words, we assume π to be extended to 0 with $\pi(0) = 0$.

The untimed language $L(\mathcal{A})$ of a timed automaton $\mathcal{A}$ is closed under the equivalence $\simeq$ and this is the theoretical foundation of many partial order reduction approaches. For instance, reductions that preserve at least one representative for each equivalence class do preserve non-emptiness of the untimed languages. Moreover the equivalence relation extends to runs when disregarding normality constraints:

Lemma 2. *Let* $(a_1,\tau_1)\dots(a_n,\tau_n)$ *be the timed labeling of a run,* $\pi : \{1,\dots,n\} \to \{1,\dots,n\}$ *be a permutation with* $a_1\dots a_n \simeq a_{\pi(1)}\dots a_{\pi(n)}$*. Then* $(a_{\pi(1)},\tau_{\pi(1)})\dots(a_{\pi(n)},\tau_{\pi(n)})$ *is also a timed labeling of a run.*

Proof. The proof is by induction of the number of exchanges in π, it is sufficient to consider the case of a single exchange.

Let $(a_1,\tau_1)\dots(a_k,\tau_k)(a,\tau_{k+1})(b,\tau_{k+2})(a_{k+3},\tau_{k+3})\dots(a_n,\tau_n)$ be the time labeling where $a\ I\ b$ and let $r = (s_0,v_0)\dots(s_n,v_n)$ be the corresponding run. Assume that $s_k \xrightarrow{(a,\phi_a,\psi_a,C_a)} s_{k+1} \xrightarrow{(b,\phi_b,\psi_b,C_b)} s_{k+2}$.

We prove the existence of a unique run $r' = (s'_0,v'_0)\dots(s'_n,v'_n)$ with timed labeling $(a_1,\tau_1)\dots(a_k,\tau_k)(b,\tau_{k+2})(a,\tau_{k+1})(a_{k+3},\tau_{k+3})\dots(a_n,\tau_n)$ such that $s'_i = s_i$ for $i \neq k+1$ and $v'_i = v_i$ for $i \notin \{k+1,k+2\}$.

By property (i) of I, $s_k \xrightarrow{(b,\phi_b,\psi_b,C_b)} s'_{k+1} \xrightarrow{(a,\phi_a,\psi_a,C_a)} s_{k+2}$ and all other transitions are unchanged hence $s'_i = s_i$ for $i \neq k+1$.

The sequence r' is a run if the time valuations v'_i satisfy the constraints. We consider two cases:

(1) $i \leq k$ or $i > k+3$. The result holds since r is a run.
(2) $i = k+1$,$i = k+2$ and $i = k+3$.

First observe that since r is a run, we have $v_{k+1} + (\tau_{k+2} - \tau_{k+1}) \vDash \phi_b \wedge \psi_b$: By condition (ii) of independence, no clock mentioned in $\phi_b \wedge \psi_b$ is reset in C_a hence $(v_{k+1} + (\tau_{k+2} - \tau_{k+1}))(x) = v_{k+1}(x) + (\tau_{k+2} - \tau_{k+1}) = (v_k + (\tau_{k+1} - \tau_k))(x) + (\tau_{k+2} - \tau_{k+1}) = (v_k + (\tau_{k+2} - \tau_k))(x)$ for any clock x mentioned in ϕ_b.

Therefore $v_{k+1} + (\tau_{k+2} - \tau_{k+1}) \vDash \phi_b \wedge \psi_b$ iff $v_k + (\tau_{k+2} - \tau_k) \vDash \phi_b \wedge \psi_b$.

Second, for a clock $x \in C_b$ we have $v_{k+1} + (\tau_{k+2} - \tau_{k+1}) \vDash \mathit{Inv}(s_{k+1})(x)$. Then, since due to independence we know that $x \notin C_a$, again $(v_{k+1} + (\tau_{k+2} - \tau_{k+1}))(x) = (v_k + (\tau_{k+2} - \tau_k))(x)$. Therefore the transition $s_k \xrightarrow{(b,\phi_b,\psi_b,C_b)} s'_{k+1}$ is enabled at τ_{k+2} yielding $(s'_{k+1}, v'_{k+1}) \vDash \mathit{Inv}(s_{k+1})(x)$. But $\mathit{Inv}(s_{k+1})(x) \wedge \psi_a(x) \Longrightarrow \mathit{Inv}(s_k)(x)$ and ψ_a does not restrict x (due to independence), hence $\mathit{Inv}(s_{k+1})(x) \Longrightarrow \mathit{Inv}(s_k)(x)$ and we obtain $v_k + (\tau_{k+2} - \tau_k) \vDash \mathit{Inv}(s_k)(x)$. We therefore obtain that r' is a run up to (s'_{k+1}, v'_{k+1}).

Similarly the transition $s'_{k+1} \xrightarrow{(a,\phi_a,\psi_a,C_a)} s_{k+2}$ satisfies the conditions on $v'_{k+1} + (\tau_{k+1} - \tau_{k+2})$: For $x \notin C_b$ we have $(v'_{k+1} + (\tau_{k+1} - \tau_{k+2}))(x) = (v_k + (\tau_{k+1} - \tau_k))(x)$, hence $v'_{k+1} + (\tau_{k+1} - \tau_{k+2}) \vDash \phi_a \wedge \psi_a$. If, on the other hand, $x \in C_a$, then $x \notin C_b$ and we see that $v'_{k+1} + (\tau_{k+1} - \tau_{k+2}) \vDash \mathit{Inv}(s_k)(x)$. But $\mathit{Inv}(s_k)(x) \wedge \phi_b \wedge \psi_b \Longrightarrow \mathit{Inv}(s'_{k+1})(x)$ and since $\phi_b \wedge \psi_b$ do not constrain x due to independence, we obtain that $\mathit{Inv}(s_k)(x) \Longrightarrow \mathit{Inv}(s'_{k+1})(x)$ and hence $v'_{k+1} + (\tau_{k+1} - \tau_{k+2}) \vDash \mathit{Inv}(s'_{k+1})(x)$.

Finally, since $C_a \cap C_b = \emptyset$ we get $v'_{k+2} = v_{k+2} + (\tau_{k+1} - \tau_{k+2})$ which implies $v'_{k+2} + (\tau_{k+3} - \tau_{k+1}) = v_{k+2} + (\tau_{k+3} - \tau_{k+2})$. This guarantees that the transition corresponding to a_{k+3} is still possible at τ_{k+3} and that $v'_{k+3} = v_{k+3}$. □

However, Lemma 2 only claims commutability of runs without taking time progress into account. For the timed language $L_T(\mathcal{A})$ and consequently for $L_N(\mathcal{A})$, the normality condition may exclude some representatives in a trace:

Let afcgahib and acfgahib be two equivalent paths of Figure 2. It is easy to know that the former one is in $L_N(\mathcal{A})$, while the latter not because the constraint $(\tau_5 - \tau_2 \leq 1) \wedge (2 \leq \tau_4 - \tau_3 \leq 4) \wedge (\tau_2 \leq \tau_3) \wedge (\tau_4 \leq \tau_5)$ cannot be satisfied by a normal word (a,τ_1)(c,τ_2)(f,τ_3)(g,τ_4)(a,τ_5)... Therefore we introduce a weaker notion of normality:

A timed word $(a_1, \tau_1) \dots (a_n, \tau_n)$ is *I-normal* iff for any two letters a_i, a_j with $i \leq j$ *and additionally* $a_i \; D \; a_j$ we have $\tau_i \leq \tau_j$. In Figure 2, the timed word (a,0.5)(c,9.5)(f,5.7)(g,9.6)(a,10.1) is I-normal. The intuition behind this relaxation of constraints is that in practice, actions are dependent if they are executed by the same component in a network of timed automata. This non-decreasing condition on action occurrences models the sequential behavior of each component. In [6], this is modeled by considering a local time for each component. The interaction between components leads to the propagation of time progress to other components (formally due to dependency).

In analogy to realisable words, we say that $a_1 \dots a_n$ is *I-realisable* iff it is the labelling of a run $(s_0, v_0) \xrightarrow{(a_1,\phi_{a_1},C_{a_1}),\tau_1} (s_1, v_1) \dots \xrightarrow{(a_n,\phi_{a_n},C_{a_n}),\tau_n} (s_n, v_n)$ in $\mathcal{A}$ such that $(a_1, \tau_1) \dots (a_n, \tau_n)$ is I-normal. As for L_N, let $L_I(\mathcal{A})$ denote the set of I-realisable words $a_1 \dots a_n$ such that $a_1 \dots a_n\$$ is the labelling of an *accepted run* (i.e. $s_n \in F$ and in particular $v_n \models Inv(s_n)$). For instance, afghb is I-realisable in Figure 2 as time stamps $0.5, 2.5, 4.6, 7, 9.4, 9.3$ satisfy clock constraints of transitions from (s0,w0) to (s2,w0) and (a,0.5)(f,2.5)(g,4.6)(h,7)(b,9.4)(i,9.3) is I-normal. Moreover, afghbi is also in $L_I(\mathcal{A})$ since (s2,w0) is final.

Obviously $L_N(\mathcal{A}) \subseteq L_I(\mathcal{A})$.

By definition $L_T(\mathcal{A}) = \emptyset$ if and only if $L_N(\mathcal{A}) = \emptyset$. Moreover, the following proposition implies that $L_N(\mathcal{A}) = \emptyset$ iff $L_I(\mathcal{A}) = \emptyset$, so that we can check this emptiness problem equivalently for either language.

Proposition 2. *For every I-normal labelling $(a_1, \tau_1) \dots (a_n, \tau_n)$ of a run in an action deterministic, constraint consistent timed automaton $\mathcal{A}$, there exists $(a_{\pi(1)}, \tau_{\pi(1)}) \dots (a_{\pi(n)}, \tau_{\pi(n)})$ an equivalent normal labelling of an (equivalent) run in $\mathcal{A}$, where π is a permutation as defined in Lemma 1.*

Proof. Consider the following ordering on $\{1, \dots, n\}$: $i \sqsubset j$ iff $\tau_i < \tau_j$ or $\tau_i = \tau_j$ and $i < j$. There is a unique permutation such that $i \sqsubset j$ iff $\pi(i) < \pi(j)$. Moreover, for $a_i D a_j$ and $i < j$, I-normality implies that $\tau_i \leq \tau_j$ and finally $\pi(i) < \pi(j)$, i.e. π yields an equivalent path. By Lemma 2, $(a_{\pi(1)}, \tau_{\pi(1)}) \dots (a_{\pi(n)}, \tau_{\pi(n)})$ is thus a timed labelling of some run and by the construction of $\sqsubset$ it is a normal timed word. □

A sorting algorithm provides an efficient way of computing a normal timed labelling of a run from an I-normal labelling.

A key main feature of $L_I(\mathcal{A})$ is the closure under equivalence that is stated in Theorem 1. In principle this allows to limit exploration of realisable clocked words to representatives of equivalence class:

Theorem 1. *(1) Let $u \simeq v$ and $u \in L_I(\mathcal{A})$ then $v \in L_I(\mathcal{A})$.*
(2) $L_I(\mathcal{A}) = \{u \mid \exists v \simeq u : \; v \in L_N(\mathcal{A})\}$.

Proof. (1) Let $u = a_1 \dots a_n$, $v = b_1 \dots b_n$ and π be the permutation linking $a_1 \dots a_n$ and $b_1 \dots b_n$ according to Lemma 1. Let $(a_1, \tau_1) \dots (a_n, \tau_n)$ an I-normal labelling of some accepting run of $\mathcal{A}$. Then $(b_1, \tau_{\pi(1)}) \dots (b_n, \tau_{\pi(n)})$ is a timed labelling of some accepting run according to Lemma 2 and it inherits I-normality since π preserves the order of occurrences of dependent actions.

(2) "$\supseteq$" follows from $L_N(\mathcal{A}) \subseteq L_I(\mathcal{A})$ (normality implies I-normality) and reflexivity of $\simeq$. "$\subseteq$" is an easy consequence of Proposition 2. □

5 Symbolic Analysis for L_I

The goal of this section is to show how the results of the previous section can be used for reachability analysis. A full description would require a lot of space and we refer the reader to [11] for an exhaustive treatment without invariants. Instead, here we concentrate on one aspect, the constraints required to check whether a word belongs to $L_I(A)$.

Let $\mathbb{T} = \{t_0, t_1, \dots\}$ be a set of time stamp variables. An atomic time constraint is a time constraint of the form $t_i - t_j \prec c$, a general constraint a conjunction of atomic constraints, where t_i, t_j are time stamp variables in $\mathbb{T}$, $\prec \in \{<, \leq\}$ and c is a constant in $\mathbb{Z}$. An *interpretation* of a time stamp constraint φ is a function $v : \mathbb{T} \rightarrow \mathbb{R}^+$ assigning a non-negative real number τ_i to each time stamp variable t_i. The satisfaction of φ by v is denoted $v \models \varphi$ and in that case v is a *model* for φ. We call φ *consistent* iff it has a model otherwise *inconsistent.* As is well known, consistency and a model can be determined with the Bellman-Ford shortest path algorithm.

To express I-realizability in terms of time stamp constraints we need to define special positions in a path. Given a path labeling $a_1 \dots a_n$ we define $last_a(a_1 \dots a_n)$, the last occurrence of a, to be the maximal k such that $a_k = a$, if such a k exists, otherwise $last_a(a_1 \dots a_n) = 0$. Similarly, we define $last_x(a_1 \dots a_n)$ to be the maximal position k at which x is reset, that is $x \in C_{a_k}$, if such a position exists, otherwise $last_x(a_1 \dots a_n) = 0$ (every clock is reset at the beginning).

With these positions we express that in a word dependent actions are ordered according to their order of occurrence (condition (1) in the following) and that clock constraints are satisfied (conditions (2) (3) and (4)) allowing to check I-realizability on the level of consistency:

For a timed automaton $\mathcal{A}$ and a path labelling $a_1 \dots a_n$ let $\varphi_{a_1 \dots a_n}$ be the *associated time stamp constraint* which is the conjunction of the time stamp constraints satisfying one of four cases :

1. $t_i - t_j \leq 0$ with $i < j$ and $a_i \; D \; a_j$ and $i = last_{a_i}(a_1 \dots a_{j-1})$;
2. $t_j - t_i \prec c$ with $x \prec c$ in ϕ_j, ψ_j, and $i = last_x(a_1 \dots a_{j-1})$
3. $t_i - t_j \prec -c$ with $x \succ c$ in ϕ_j, and $i = last_x(a_1 \dots a_{j-1})$
4. $t_j - t_i \prec c$ with $x \in C_j$ and $x \prec c = Inv(\sigma(a_1 \dots a_{j-1}))(x)$ and $i = last_x(a_1 \dots a_{j-1})$

Proposition 3. *Let $\mathcal{A} = (\Sigma, S, s_0, \rightarrow, F)$ be a deterministic timed automaton and I an independence relation. Moreover, let $a_1 \dots a_n$ be a path labeling of $\mathcal{A}$.*

The word $a_1 \dots a_n$ is I-realizable iff its associated time stamp constraint $\varphi_{a_1 \dots a_n}$ is consistent.

In [11], *event zones* were introduced as an incremental way of computing consistency of time stamp constraints: An event zone is a triple $Z = (T, \varphi, Last)$ where T is a set of time stamp variables, φ is a time stamp constraint and $Last : X \cup \Sigma \rightarrow T$ is the *last occurrence function* that assigns to a clock or an action a the time stamps that represents respectively its last reset and the last occurrence of the action. Formally, the event zone $Z_u = (T_u, \varphi_u, Last_u)$ of the path labeling $u = a_1 \dots a_n$ is given by $T_u = \{t_0, \dots, t_n\}$, where φ_u is the time stamp constraint associated to u and $Last_u(a) = t_i$ with $i = last_a(a_1 \dots a_n)$ for all action a, $Last_u(x) = t_i$ with $i = last_x(a_1 \dots a_n)$ for all clock x.

To obtain a symbolic automaton, we consider pairs (s, Z) with s a location from the original timed automaton and Z an event zone, *symbolic states.* Transitions on symbolic states are obtained by *extension* $(s_1, Z_1) \circ a := (s_2, Z_2) = (s_2, (T_2, \varphi_2, Last_2))$ of a symbolic state $(s_1, Z_1) = (s_1, (T_1, \varphi_1, Last_1)$ by an action a is defined if there exists a transition $s_1 \xrightarrow{a, \phi_a, \psi_a, C_a} s_2$, such that $T_2 = T_1 \uplus \{t\}$ with t a fresh time stamp variable not in T_1, and φ_2 is the *consistent* conjunction of φ_1 and

– $t_i - t \leq 0$ for all $t_i = Last(b)$ for b such that $a\ D\ b$,
– $t - t_i \prec c$ with $x \prec c$ in ϕ_a, ψ_a, and $t_i = Last(x)$,
– $t_i - t \prec -c$ with $x \succ c$ in ϕ_a and $t_i = Last(x)$,
– $t - t_i \prec c$ with $x \prec c = Inv(s_1)(x)$ iff $x \in C_a$,

and finally $Last_2$ is such that $Last_2(\alpha) = t$ for α a clock in C_a or $\alpha = a$ otherwise $Last_2(\alpha) = Last_1(\alpha)$.

Following the lines of [11], it is then possible to define an equivalence relation $\simeq_{EZ}$ compatible with the extension, such that if $(s_1, Z_1) \simeq_{EZ} (s_2, Z_2)$ then $(s_1, Z_1) \circ a \simeq_{EZ} (s_2, Z_2) \circ a$. This equivalence is essentially "same constraint up to pointer renaming". It turns out that the independence relation is compatible with $\simeq_{EZ}$, i.e. for $a\ I\ b$ we have $(s_1, Z_1) \circ a \circ b \simeq_{EZ} (s_1, Z_1) \circ b \circ a$, the fundamental reason why event zones reduce the number of symbolic states explored. The equivalence classes are the symbolic states of the event zone automaton, which thus itself respects the independence relation.

The interesting aspect of event zones is that they allow to abstract from time stamps that are not referenced by *Last*. More precisely, the constraint of the event zone is *closed* using the Floyd-Warshall algorithm and then the time stamps not referenced by pointers can be projected away. This allows to limit the dimensions of event zones to the number of pointers (here: clocks and the size of the alphabet). In practice, we use an optimized set of pointers to further reduce the dimensions, which moreover results in event zones for \$ terminated paths that never have more dimensions than the number of clocks plus one. This corresponds to the dimension of classical clock zones.

The snapshot action \$ obtains a special rôle in the state exploration with event zones: When we reach a final state (desired property) with a symbolic state (s, Z), we evaluate the consistency of $(s, Z) \circ \$$ according to Definition 3. We also eliminate intermediate states using the snapshot action, due to the following observation:

Lemma 3. *Let* $(a_1, \tau_1) \dots (a_m, \tau_m)(a_{m+1}, \tau_{m+1}) \dots (a_n, \tau_n)(\$, \tau_n)$ *be a normal run accepted by* $\mathcal{A}$*, then so is* $(a_1,\tau_1)\dots(a_m,\tau_m)(\$,\tau_m)(a_{m+1},\tau_{m+1})\dots(a_n,\tau_n)(\$,\tau_n)$*. In particular,* $a_1 \dots a_m\$$ *is* I*-realisable.*

Proof. Check definitions and use Proposition 1. □

This observation allows us to restrict our search to paths u such that $u\$$ is I-realisable (we do *not* want to check *realisable*). This reduction ensures that every symbolic state we explore is also reachable by some equivalent interleaving in classical zone automata: We never explore paths (up to commutation) that would not be explored with standard semantics.

Algorithm 1. Generic exploration algorithm

```
Waiting ← $\{((s_\epsilon, Z_\epsilon), \epsilon)\}$, Past ← $\emptyset$
while Waiting $\neq \emptyset$ do
  Choose $((s, Z), w) \in$ Waiting, Waiting ← Waiting $\setminus \{((s, Z), w)\}$
  for all $w' = wa$ with $(s', Z') = (s, Z) \circ a$ consistent do
    if $(s', Z'') := (s', Z') \circ \$$ consistent then
      if $s' \in F$ then return "$witness(w')$" end if
      if there exits no $(s', Z''') \in$ Past with $(s', Z'') \lesssim_C (s', Z''')$ then
        Waiting ← Waiting $\cup \{((s', Z'), w')\}$, Past ← Past $\cup \{(s', Z'')\}$
      end if
    end if
  end for
end while
return "$empty$"
```

Algorithm 1 shows how we search for a state in F. "Waiting" is a set of paths and symbolic states to be explored, whereas "Past" is a set of symbolic states terminated with \$ that have been explored. $\lesssim_C$ is similar to a *zone inclusion test* and assures that we explore only a finite number of symbolic states, for details see [11]. It is essential, that we do not put the same symbolic states in Past and Waiting and that \$ is never used in a symbolic state in Waiting.

6 Experiments

We recently finished a prototype implementation based on a previous implementation of event zones without invariants. We made two experiments to demonstrate the performance of the event zone approach with state invariants. The experiments were carried out in a machine with two 2.8GHz Xeon CPUs, 2GB memory and Fedora core 4 Linux.

The first experiment is a timed version of dining philosophers with urgency representing constraints to avoid literal starvation (due to lack of space, we cannot give the detailed model here). Figure 3 shows the results generated by our prototype and UppAal (v3.6 beta 1). The data under the title "No partial order" were obtained by our implementation setting all transitions dependent. They are results in principle in the same basic algorithm used by UppAal, but our implementation is lacking abstraction techniques like [8], which explains the largely superior results obtained by UppAal. The potential of the reduction can thus be seen by comparing the figures with and without "partial order" reduction. However, with reduction - despite the lack of abstractions - even the current implementation greatly outperforms UppAal.

Number of philosophers	No partial order		With partial order		UppAal	
	time	memory	time	memory	time	memory
2	0.02s	16m	0.03s	16m	0.03s	4m
3	0.11s	17m	0.05s	16m	0.04s	5m
4	21.88s	44m	0.53s	17m	0.29s	6m
5	—	—	9.79s	22m	12.86s	36m
6	—	—	175.10s	72m	1523.22s	730m
7	—	—	2909.32s	540m	—	—

Fig. 3. Results of the philosophers example

The second experiment was performed on the highway example of Section 2. The results[3] are listed in Figure 4. The advantage of event zone with state invariants against UppAal in this experiment was more explicit than the first one.

Number of lanes	No partial order		With partial order		UppAal	
	time	memory	time	memory	time	memory
1	0.02s	16m	0.03s	16m	0.02s	6m
2	0.03s	16m	0.03s	16m	0.02s	6m
3	0.06s	16m	0.04s	16m	0.03s	6m
4	1.98s	22m	0.30s	17m	0.23s	7m
5	548.57s	279m	1.29s	19m	20.35s	29m
6	34681.91s(*)	2301m	10.80s	36m	2946.67s	438m
7	—	—	87.35s	119m	—	—
8	—	—	554.35s	466m	—	—

Fig. 4. Results of the highway example

7 Conclusions

In this paper, we have shown how to add local state invariants to the event zone approach for timed automata reachability. We thus lift the application domain to the full class of reachability analysis that can be done with tools like UppAal or Kronos. Moreover, we have implemented the algorithm and shown that it can compete with state of the art timed automata tools. However, we continue to improve certain aspects of the implementation, that is meant to be published soon.

[3] The number labeled "(*)" was not accurate since the swap memory was used automatically by the operating system.

Recently, an alternative approach to the exploitation of independence has been introduced in [12], which is based on unions of clock zones for different interleavings of the same trace. Note that a single event zone represents these unions, regardless of the interleaving that was used to compute it. On the other hand, the current setting for invariants can be applied in the dependency analysis of an extension of the latter work, where clocks can be shared variables between different automata/processes of a network.

References

1. R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
2. G. Behrmann, P. Bouyer, E. Fleury, and K.G. Larsen. Static guard analysis in timed automata verification. In *In TACAS*, volume 2619 of *Lecture Notes in Computer Science*, pages 254–277. Springer Verlag, 2003.
3. G. Behrmann, A. David, K. G. Larsen, O. Moeller, P. Pettersson, and W. Yi. UPPAAL - present and future. In *Proc. of 40*th *IEEE Conference on Decision and Control.* IEEE Computer Society Press, 2001.
4. G. Behrmann, K. Larsen, J. Pearson, C. Weise, W. Yi, and J. Lind-Nielsen. Efficient timed reachability analysis using clock difference diagrams. In *International Conference on Computer Aided Verification*, volume 1633 of *Lecture Notes in Computer Science*, pages 341–353, 1999.
5. W. Belluomini and C. Myers. Verification of timed systems using POSETs. In *International Conference on Computer Aided Verification*, pages 403–415, 1998.
6. J. Bengtsson, B. Jonsson, J. Lilius, and W. Yi. Partial order reductions for timed systems. In *International Conference on Concurrency Theory (CONCUR)*, volume 1466 of *Lecture Notes in Computer Science*, pages 485–500. Springer Verlag, 1998.
7. C. Daws and S. Tripakis. Model checking of real-time reachability properties using abstractions. *Lecture Notes in Computer Science*, 1384:pp. 313, 1998.
8. C. Daws and S. Yovine. Reducing the number of clock variables of timed automata. In *IEE Real-Time Systems Symposium*, pages 73–81, December 1996.
9. T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111:193–244, 1994.
10. K. Larsen, P. Pettersson, and W. Yi. Model-checking for real-time systems. In *Fundamentals of Computation Theory*, Lecture Notes in Computer Science, pages 62–88. Springer Verlag, 1995.
11. D. Lugiez, P. Niebert, and S. Zennou. A partial order semantics approach to the clock explosion problem of timed automata. *Theoretical Computer Science*, 345(1):27–59, 2005.
12. R. Ben Salah, M. Bozga, and O. Maler. On interleaving in timed automata. In *International Conference on Concurrency Theory, CONCUR 2006*, volume 4137 of *LNCS*, 2006.
13. S. Yovine. Kronos: A verification tool for real-time systems. *Software Tools for Technology Transfer*, 1(1):123–133, 1997.
14. S. Zennou, M. Yguel, and P. Niebert. Else: A new symbolic state generator for timed automata. In *Proceedings of the 1st International Conference FORMATS*, volume 2791 of *Lecture Notes in Computer Science*, pages 273–280. Springer, 2003.

Static Analysis for State-Space Reduction of Polygonal Hybrid Systems

Gordon Pace[1] and Gerardo Schneider[2]

[1] Dept. of Computer Science and AI, University of Malta, Msida, Malta
[2] Dept. of Informatics, University of Oslo, Oslo, Norway
gordon.pace@um.edu.mt, gerardo@ifi.uio.no

Abstract. Polygonal hybrid systems (SPDI) are a subclass of planar hybrid automata which can be represented by piecewise constant differential inclusions. The reachability problem as well as the computation of certain objects of the phase portrait, namely the viability, controllability and invariance kernels, for such systems is decidable. In this paper we show how to compute another object of an SPDI phase portrait, namely semi-separatrix curves and show how the phase portrait can be used for reducing the state-space for optimizing the reachability analysis.

1 Introduction

Hybrid systems combining discrete and continuous dynamics arise as mathematical models of various artificial and natural systems, and as approximations to complex continuous systems. They have been used in various domains, including avionics, robotics and bioinformatics. Reachability analysis has been the principal research question in the verification of hybrid systems, even if it is a well-known result that for most subclasses of hybrid systems most verification questions are undecidable. Various decidable subclasses have, subsequently, been identified, including timed [AD94] and rectangular automata [HKPV95], hybrid automata with linear vector fields [LPY01], piecewise constant derivative systems (PCDs) [MP93] and polygonal hybrid systems (SPDIs) [ASY01].

Compared to reachability verification, qualitative analysis of hybrid systems is a relatively neglected area [ALQ$^+$01b, DV95, MS00, SP02, SJSL00]. Typical qualitative questions include: 'Are there 'sink' regions where a trajectory can never leave once it enters the region?' and 'Are there regions in which every point in the region is reachable from every other?'. The collection of objects in a system satisfying these properties is called the *phase portrait* of the system.

Defining and constructing phase portraits of hybrid systems has been directly addressed for PCDs in [MS00], and for SPDIs in [ASY02]. Given a cycle on a SPDI, the *viability* kernel is the largest set of points in the cycle which may loop forever within the cycle. The *controllability* kernel is the largest set of strongly connected points in the cycle (such that any point in the set may be reached from any other). An *invariant set* is a set of points such that each point must keep rotating within the set forever, and the *invariance kernel* is the largest such set. Algorithms for computing these kernels have been presented in [ASY02, Sch04] and implemented in the tool set SPeeDI$^+$[PS].

E. Asarin and P. Bouyer (Eds.): FORMATS 2006, LNCS 4202, pp. 306–321, 2006.

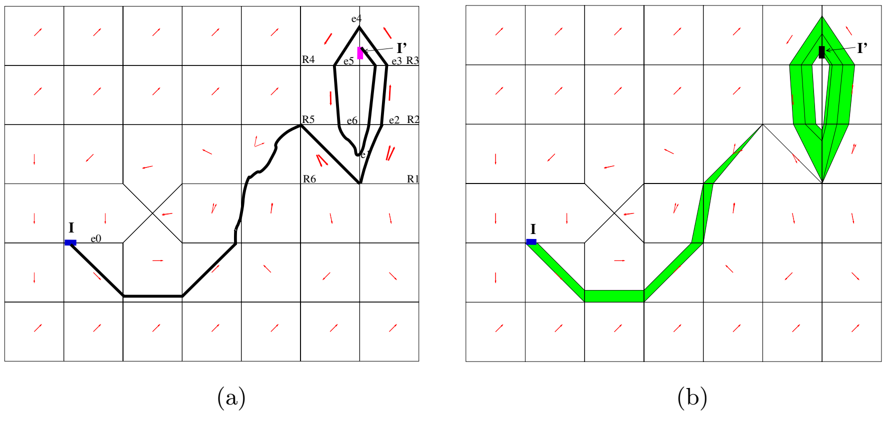


Fig. 1. (a) An SPDI and its trajectory segment; (b) Reachability analysis

The contribution of this paper is threefold. We start by introducing a new element of the phase portrait of SPDIs, *semi-separatrix curves*, and give an algorithm to compute them. Separatrices are convex polygons dissecting the plane into two mutually non-reachable subsets. We then show how the kernels can be used to answer reachability questions directly. We also show how semi-separatrices can be used to optimize the reachability algorithm for SPDIs by reducing the number of states of the SPDI graph. The optimization is based on topological properties of the plane (and in particular, those of SPDIs).

2 Theoretical Background

We summarize here the main definitions and results about SPDIs; for a more detailed description refer to [Sch02]. A (positive) *affine* function $f : \mathbb{R} \to \mathbb{R}$ is such that $f(x) = ax + b$ with $a > 0$. An *affine multivalued* function $F : \mathbb{R} \to 2^{\mathbb{R}}$, denoted $F = \langle f_l, f_u \rangle$, is defined by $F(x) = \langle f_l(x), f_u(x) \rangle$ where f_l and f_u are affine and $\langle \cdot, \cdot \rangle$ denotes an interval. For notational convenience, we do not make explicit whether intervals are open, closed, left-open or right-open, unless required for comprehension. For an interval $I = \langle l, u \rangle$ we have that $F(\langle l, u \rangle) = \langle f_l(l), f_u(u) \rangle$. The *inverse* of F is defined by $F^{-1}(x) = \{y \mid x \in F(y)\}$. The *universal inverse* of F is defined by $\tilde{F}^{-1}(I) = I'$ where I' is the greatest non-empty interval satisfying $\forall x \in I' \cdot F(x) \subseteq I$.

Clearly, $F^{-1} = \langle f_u^{-1}, f_l^{-1} \rangle$ and $\tilde{F}^{-1} = \langle f_l^{-1}, f_u^{-1} \rangle$, provided that $\langle f_l^{-1}, f_u^{-1} \rangle \neq \emptyset$.

A *truncated affine multivalued* function (TAMF) $\mathcal{F} : \mathbb{R} \to 2^{\mathbb{R}}$ is defined by an affine multivalued function F and intervals $S \subseteq \mathbb{R}^+$ and $J \subseteq \mathbb{R}^+$ as follows: $\mathcal{F}(x) = F(x) \cap J$ if $x \in S$, otherwise $\mathcal{F}(x) = \emptyset$. For convenience we write $\mathcal{F}(x) = F(\{x\} \cap S) \cap J$. For an interval I, $\mathcal{F}(I) = F(I \cap S) \cap J$ and $\mathcal{F}^{-1}(I) = F^{-1}(I \cap J) \cap S$. The *universal inverse* of $\mathcal{F}$ is defined by $\tilde{\mathcal{F}}^{-1}(I) = I'$ if and only if I' is the greatest non-empty interval such that for all $x \in I'$, $F(x) \subseteq I$ and

$F(x) = \mathcal{F}(x)$. We say that $\mathcal{F}$ is *normalized* if $S = \mathsf{Dom}(\mathcal{F}) = \{x \mid F(x) \cap J \neq \emptyset\}$ (thus, $S \subseteq F^{-1}(J)$) and $J = \mathsf{Im}(\mathcal{F}) = \mathcal{F}(S)$.

It can be proved [ASY01], that TAMFs are closed under composition.

Theorem 1. The composition of two TAMFs $\mathcal{F}_1(I) = F_1(I \cap S_1) \cap J_1$ and $\mathcal{F}_2(I) = F_2(I \cap S_2) \cap J_2$, is the TAMF $(\mathcal{F}_2 \circ \mathcal{F}_1)(I) = \mathcal{F}(I) = F(I \cap S) \cap J$, where $F = F_2 \circ F_1$, $S = S_1 \cap F_1^{-1}(J_1 \cap S_2)$ and $J = J_2 \cap F_2(J_1 \cap S_2)$. □

2.1 SPDIs

An *angle* $\angle_{\mathbf{a}}^{\mathbf{b}}$ on the plane, defined by two non-zero vectors $\mathbf{a}, \mathbf{b}$, is the set of all positive linear combinations $\mathbf{x} = \alpha\ \mathbf{a} + \beta\ \mathbf{b}$, with $\alpha, \beta \geq 0$, and $\alpha + \beta > 0$. We will assume that $\mathbf{b}$ is situated in the counter-clockwise direction from $\mathbf{a}$.

A *polygonal hybrid system*[1] (SPDI) is a finite partition $\mathbb{P}$ of the plane into convex polygonal sets, such that for each $P \in \mathbb{P}$ we have two vectors $\mathbf{a}_P$ and $\mathbf{b}_P$. Let $\phi(P) = \angle_{\mathbf{a}_P}^{\mathbf{b}_P}$. The SPDI is determined by $\dot{\mathbf{x}} \in \phi(P)$ for $\mathbf{x} \in P$.

Let $E(P)$ be the set of edges of P. We say that e is an *entry* of P if for all $\mathbf{x} \in e$ and for all $\mathbf{c} \in \phi(P)$, $\mathbf{x} + \mathbf{c}\epsilon \in P$ for some $\epsilon > 0$. We say that e is an *exit* of P if the same condition holds for some $\epsilon < 0$. We denote by $in(P) \subseteq E(P)$ the set of all entries of P and by $out(P) \subseteq E(P)$ the set of all exits of P.

Assumption 1. *All the edges in $E(P)$ are either entries or exits, that is, $E(P) = in(P) \cup out(P)$.*

Reachability for SPDIs is decidable provided the above assumption holds [ASY01]; without such assumption it is not know whether reachability is decidable.

A *trajectory segment* of an SPDI is a continuous function $\xi : [0, T] \to \mathbb{R}^2$ which is smooth everywhere except in a discrete set of points, and such that for all $t \in [0, T]$, if $\xi(t) \in P$ and $\dot{\xi}(t)$ is defined then $\dot{\xi}(t) \in \phi(P)$. The *signature*, denoted $\mathsf{Sig}(\xi)$, is the ordered sequence of edges traversed by the trajectory segment, that is, $e_1, e_2, \ldots$, where $\xi(t_i) \in e_i$ and $t_i < t_{i+1}$. If $T = \infty$, a trajectory segment is called a *trajectory*.

Example 1. Consider the SPDI illustrated in Fig. 1-(a). For sake of simplicity we will only show the dynamics associated to regions R_1 to R_6 in the picture. For each region R_i, $1 \leq i \leq 6$, there is a pair of vectors $(\mathbf{a}_i, \mathbf{b}_i)$, where: $\mathbf{a}_1 = (45, 100), \mathbf{b}_1 = (1, 4)$, $\mathbf{a}_2 = \mathbf{b}_2 = (1, 10)$, $\mathbf{a}_3 = \mathbf{b}_3 = (-2, 3)$, $\mathbf{a}_4 = \mathbf{b}_4 = (-2, -3)$, $\mathbf{a}_5 = \mathbf{b}_5 = (1, -15)$, $\mathbf{a}_6 = (1, -2), \mathbf{b}_6 = (1, -1)$. A trajectory segment starting on interval $I \subset e_0$ and finishing in interval $I' \subseteq e_4$ is depicted. ■

We say that a signature σ is *feasible* if and only if there exists a trajectory segment ξ with signature σ, i.e., $\mathsf{Sig}(\xi) = \sigma$. From this definition, it immediately follows that extending an unfeasible signature can never make it feasible:

[1] In the literature the names *polygonal differential inclusion* and *simple planar differential inclusion* have been used to describe the same systems.

Proposition 1. *If a signature σ is not feasible, then neither is any extension of the signature — for any signatures σ' and σ'', the signature $\sigma'\sigma\sigma''$ is not feasible.* □

Given an SPDI $\mathcal{S}$, let $\mathcal{E}$ be the set of edges of $\mathcal{S}$, then we can define a graph $\mathcal{G}_\mathcal{S}$ where nodes correspond to edges of $\mathcal{S}$ and such that there exists an arc from one node to another if there exists a trajectory segment from the first edge to the second one without traversing any other edge. More formally: Given an SPDI $\mathcal{S}$, the *underlying graph of* $\mathcal{S}$ (or simply the *graph of* $\mathcal{S}$), is a graph $\mathcal{G}_\mathcal{S} = (N_\mathcal{G}, A_\mathcal{G})$, with $N_\mathcal{G} = \mathcal{E}$ and $A_\mathcal{G} = \{(e,e') \mid \exists \xi, t \,.\, \xi(0) \in e \wedge \xi(t) \in e' \wedge \mathsf{Sig}(\xi) = ee'\}$. We say that a sequence $e_0 e_1 \ldots e_k$ of nodes in $\mathcal{G}_\mathcal{S}$ is a *path* whenever $(e_i, e_{i+1}) \in A_\mathcal{G}$ for $0 \leq i \leq k-1$.

The following lemma shows the relation between edge signatures in an SPDI and paths in its corresponding graph.

Lemma 1. *If ξ is a trajectory segment of $\mathcal{S}$ with edge signature $\mathsf{Sig}(\xi) = \sigma = e_0 \ldots e_p$, it follows that σ is a path in $\mathcal{G}_\mathcal{S}$.* □

Note that the converse of the above lemma is not true in general. It is possible to find a counter-example where there exists a path from node e to e', but no trajectory from edge e to edge e' in the SPDI.

2.2 Successors and Predecessors

Given an SPDI, we fix a one-dimensional coordinate system on each edge to represent points laying on edges [ASY01]. For notational convenience, we indistinctly use letter e to denote the edge or its one-dimensional representation. Accordingly, we write $\mathbf{x} \in e$ or $x \in e$, to mean "point $\mathbf{x}$ in edge e with coordinate x in the one-dimensional coordinate system of e". The same convention is applied to sets of points of e represented as intervals (e.g., $\mathbf{x} \in I$ or $x \in I$, where $I \subseteq e$) and to trajectories (e.g., "ξ starting in x" or "ξ starting in $\mathbf{x}$").

Now, let $P \in \mathbb{P}$, $e \in \mathit{in}(P)$ and $e' \in \mathit{out}(P)$. For $I \subseteq e$, $\mathsf{Succ}_{e,e'}(I)$ is the set of all points in e' reachable from some point in I by a trajectory segment $\xi : [0,t] \to \mathbb{R}^2$ in P (i.e., $\xi(0) \in I \wedge \xi(t) \in e' \wedge \mathsf{Sig}(\xi) = ee'$). $\mathsf{Succ}_{e,e'}$ is a TAMF [ASY01].

Example 2. Let $e_1, \ldots, e_6$ be as in Fig. 1-(a), where all the edges have local coordinates over $[0, 10]$, and $I = [l, u]$. We assume a one-dimensional coordinate system. We show only the first and last edge-to-edge TAMF of the cycle:

$$\begin{array}{lll} F_{e_1 e_2}(I) = \left[\frac{l}{4}, \frac{9}{20}u\right], & S_1 = [0,10], & J_1 = \left[0, \frac{9}{2}\right] \\ F_{e_6 e_1}(I) = [l, 2u], & S_6 = [0,10], & J_6 = [0,10] \end{array}$$

with $\mathsf{Succ}_{e_i e_{i+1}}(I) = F_{e_i e_{i+1}}(I \cap S_i) \cap J_i$, for $1 \leq i \leq 6$; S_i and J_i are computed as shown in Theorem 1. ■

Given a sequence $w = e_1, e_2, \ldots, e_n$, since TAMFs are closed under composition, the successor of I along w, defined as $\mathsf{Succ}_w(I) = \mathsf{Succ}_{e_{n-1},e_n} \circ \ldots \circ \mathsf{Succ}_{e_1,e_2}(I)$, is a TAMF.

Example 3. Let $\sigma = e_1 \cdots e_6 e_1$. We have that $\mathsf{Succ}_\sigma(I) = F(I \cap S_\sigma) \cap J_\sigma$, where: $F(I) = [\frac{l}{4} + \frac{1}{3}, \frac{9}{10}u + \frac{2}{3}]$, with $S_\sigma = [0, 10]$ and $J_\sigma = [\frac{1}{3}, \frac{29}{3}]$. ■

For $I \subseteq e'$, $\mathsf{Pre}_{e,e'}(I)$ is the set of points in e that can reach a point in I by a trajectory segment in P. The $\forall$-*predecessor* $\widetilde{\mathsf{Pre}}(I)$ is defined in a similar way to $\mathsf{Pre}(I)$ using the universal inverse instead of just the inverse: For $I \subseteq e'$, $\widetilde{\mathsf{Pre}}_{ee'}(I)$ is the set of points in e such that *any* successor of such points are in I by a trajectory segment in P. Both definitions can be extended straightforwardly to signatures $\sigma = e_1 \cdots e_n$: $\mathsf{Pre}_\sigma(I)$ and $\widetilde{\mathsf{Pre}}_\sigma(I)$. The successor operator thus has two "inverse" operators.

2.3 Qualitative Analysis of Simple Edge-Cycles

Let $\sigma = e_1 \cdots e_k e_1$ be a simple edge-cycle, i.e., $e_i \neq e_j$ for all $1 \leq i \neq j \leq k$. Let $\mathsf{Succ}_\sigma(I) = F(I \cap S_\sigma) \cap J_\sigma$ with $F = \langle f_l, f_u \rangle$ (we suppose that this representation is normalized). We denote by $\mathcal{D}_\sigma$ the one-dimensional discrete-time dynamical system defined by Succ_σ, that is $x_{n+1} \in \mathsf{Succ}_\sigma(x_n)$.

Assumption 2. *None of the two functions f_l, f_u is the identity.*

Without the above assumption the results are still valid but need a special treatment making the presentation more complicated.

Let l^* and u^* be the fixpoints[2] of f_l and f_u, respectively, and $S_\sigma \cap J_\sigma = \langle L, U \rangle$. A simple cycle is of one of the following types [ASY01]: STAY, the cycle is not abandoned neither by the leftmost nor the rightmost trajectory, that is, $L \leq l^* \leq u^* \leq U$; DIE, the rightmost trajectory exits the cycle through the left (consequently the leftmost one also exits) or the leftmost trajectory exits the cycle through the right (consequently the rightmost one also exits), that is, $u^* < L \vee l^* > U$; EXIT-BOTH, the leftmost trajectory exits the cycle through the left and the rightmost one through the right, that is, $l^* < L \wedge u^* > U$; EXIT-LEFT, the leftmost trajectory exits the cycle (through the left) but the rightmost one stays inside, that is, $l^* < L \leq u^* \leq U$; EXIT-RIGHT, the rightmost trajectory exits the cycle (through the right) but the leftmost one stays inside, that is, $L \leq l^* \leq U < u^*$.

Example 4. Let $\sigma = e_1 \cdots e_6 e_1$. Then, $S_\sigma \cap J_\sigma = \langle L, U \rangle = [\frac{1}{3}, \frac{29}{3}]$. The fixpoints from Example 3 are $\frac{1}{3} < l^* = \frac{11}{25} < u^* = \frac{20}{3} < \frac{29}{3}$. Thus, σ is a STAY. ■

Any trajectory that enters a cycle of type DIE will eventually quit it after a finite number of turns. If the cycle is of type STAY, all trajectories that happen to enter it will keep turning inside it forever. In all other cases, some trajectories will turn for a while and then exit, and others will continue turning forever. This information is crucial for proving decidability of the reachability problem.

Example 5. Consider the SPDI of Fig. 1-(a). Fig. 1-(b) shows part of the reach set of the interval $[8, 10] \subset e_0$, answering positively to the reachability question:

[2] The fixpoint x^* is the solution of $f(x^*) = x^*$, where $f(\cdot)$ is positive affine.

Is $[1,2] \subset e_4$ reachable from $[8,10] \subset e_0$? Fig. 1-(b) has been automatically generated by the SPeeDI toolbox we have developed for reachability analysis of SPDIs [APSY02]. ■

2.4 Kernels

We present now how to compute the invariance, controllability and viability kernels of an SPDI. Proofs are omitted but for further details, refer to [ASY02] and [Sch04]. In the following, for σ a cyclic signature, we define $K_\sigma \subseteq \mathbb{R}^2$ as follows: $K_\sigma = \bigcup_{i=1}^{k}(\mathit{int}(P_i) \cup e_i)$ where P_i is such that $e_{i-1} \in \mathit{in}(P_i)$, $e_i \in \mathit{out}(P_i)$ and $\mathit{int}(P_i)$ is P_i's interior.

Viability Kernel. We now recall the definition of *viability kernel* [Aub01]. A trajectory ξ is *viable* in K if $\xi(t) \in K$ for all $t \geq 0$. K is a *viability domain* if for every $\mathbf{x} \in K$, there exists at least one trajectory ξ, with $\xi(0) = \mathbf{x}$, which is viable in K. The *viability kernel* of K, denoted $\mathsf{Viab}(K)$, is the largest viability domain contained in K.

For $I \subseteq e_1$ we define $\overline{\mathsf{Pre}}_\sigma(I)$ to be the set of all $\mathbf{x} \in \mathbb{R}^2$ for which there exists a trajectory segment ξ starting in $\mathbf{x}$, that reaches some point in I, such that $\mathsf{Sig}(\xi)$ is a suffix of $e_2 \ldots e_k e_1$. It is easy to see that $\overline{\mathsf{Pre}}_\sigma(I)$ is a polygonal subset of the plane which can be calculated using the following procedure. We start by defining $\overline{\mathsf{Pre}}_e(I) = \{\mathbf{x} \mid \exists \xi : [0,t] \to \mathbb{R}^2, t > 0 \,.\, \xi(0) = \mathbf{x} \wedge \xi(t) \in I \wedge \mathsf{Sig}(\xi) = e\}$ and apply this operation k times: $\overline{\mathsf{Pre}}_\sigma(I) = \bigcup_{i=1}^{k} \overline{\mathsf{Pre}}_{e_i}(I_i)$ with $I_1 = I$, $I_k = \mathsf{Pre}_{e_k,e_1}(I_1)$ and $I_i = \mathsf{Pre}_{e_i,e_{i+1}}(I_{i+1})$, for $2 \leq i \leq k-1$.

The following result provides a non-iterative algorithmic procedure for computing the viability kernel of K_σ on an SPDI:

Theorem 2. *If σ is DIE, $\mathsf{Viab}(K_\sigma) = \emptyset$, otherwise $\mathsf{Viab}(K_\sigma) = \overline{\mathsf{Pre}}_\sigma(S_\sigma)$.* □

Example 6. Fig. 2-(a) shows all the viability kernels of the SPDI given in Example 1. There are 4 cycles with viability kernels — in the picture two of the kernels are overlapping. ■

Controllability Kernel. We say K is *controllable* if for any two points $\mathbf{x}$ and $\mathbf{y}$ in K there exists a trajectory segment ξ starting in $\mathbf{x}$ that reaches an arbitrarily small neighborhood of $\mathbf{y}$ without leaving K. More formally: A set K is controllable if $\forall \mathbf{x}, \mathbf{y} \in K, \forall \delta > 0, \exists \xi : [0,t] \to \mathbb{R}^2, t > 0 \,.\, (\xi(0) = \mathbf{x} \wedge |\xi(t) - \mathbf{y}| < \delta \wedge \forall t' \in [0,t] \,.\, \xi(t') \in K)$. The *controllability kernel* of K, denoted $\mathsf{Cntr}(K)$, is the largest controllable subset of K.

For a given cyclic signature σ, we define $\mathcal{C}_\mathcal{D}(\sigma)$ as follows:

$$\mathcal{C}_\mathcal{D}(\sigma) = \begin{cases} \langle L, U\rangle & \text{if } \sigma \text{ is EXIT-BOTH} \\ \langle L, u^*\rangle & \text{if } \sigma \text{ is EXIT-LEFT} \\ \langle l^*, U\rangle & \text{if } \sigma \text{ is EXIT-RIGHT} \\ \langle l^*, u^*\rangle & \text{if } \sigma \text{ is STAY} \\ \emptyset & \text{if } \sigma \text{ is DIE} \end{cases} \tag{1}$$

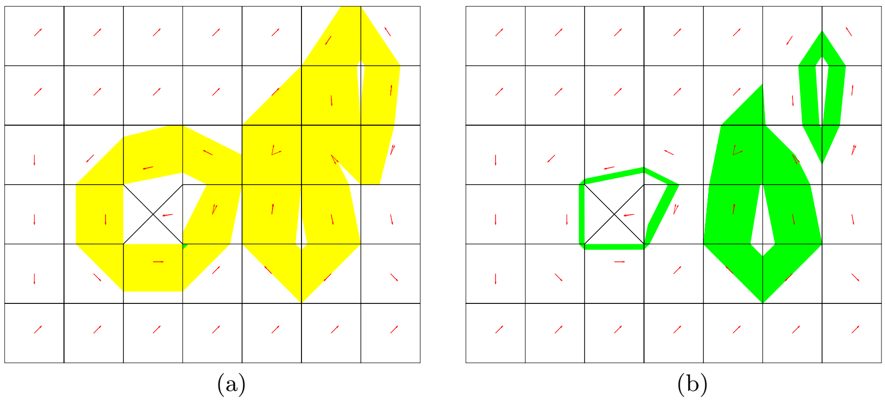


Fig. 2. (a) Viability kernels; (b) Controllability kernels

For $I \subseteq e_1$ let us define $\overline{\mathsf{Succ}}_\sigma(I)$ as the set of all points $\mathbf{y} \in \mathbb{R}^2$ for which there exists a trajectory segment ξ starting in some point $x \in I$, that reaches $\mathbf{y}$, such that $\mathsf{Sig}(\xi)$ is a prefix of $e_1 \dots e_k$. The successor $\overline{\mathsf{Succ}}_\sigma(I)$ is a polygonal subset of the plane which can be computed similarly to $\overline{\mathsf{Pre}}_\sigma(I)$. Define $\mathcal{C}(\sigma) = (\overline{\mathsf{Succ}}_\sigma \cap \overline{\mathsf{Pre}}_\sigma)(\mathcal{C}_\mathcal{D}(\sigma))$. We compute the controllability kernel of K_σ as follows:

Theorem 3. $\mathsf{Cntr}(K_\sigma) = \mathcal{C}(\sigma)$. □

Example 7. Fig. 2-(b) shows all the controllability kernels of the SPDI given in Example 1. There are 4 cycles with controllability kernels — in the picture two of the kernels are overlapping. ■

The following result which relates controllability and viability kernels, states that the viability kernel of a given cycle is the local basin of attraction of the corresponding controllability kernel.

Proposition 2. *Any viable trajectory in* K_σ *converges to* $\mathsf{Cntr}(K_\sigma)$. □

Let $\mathsf{Cntr}^l(K_\sigma)$ be the closed curve obtained by taking the leftmost trajectory and $\mathsf{Cntr}^u(K_\sigma)$ be the closed curve obtained by taking the rightmost trajectory which can remain inside the controllability kernel. In other words, $\mathsf{Cntr}^l(K_\sigma)$ and $\mathsf{Cntr}^u(K_\sigma)$ are the two polygons defining the controllability kernel.

A non-empty controllability kernel $\mathsf{Cntr}(K_\sigma)$ of a given cyclic signature σ partitions the plane into three disjoint subsets: (1) the controllability kernel itself, (2) the set of points limited by $\mathsf{Cntr}^l(K_\sigma)$ (and not including $\mathsf{Cntr}^l(K_\sigma)$) and (3) the set of points limited by $\mathsf{Cntr}^u(K_\sigma)$ (and not including $\mathsf{Cntr}^u(K_\sigma)$). We define the *inner* of $\mathsf{Cntr}(K_\sigma)$ (denoted by $\mathsf{Cntr}_{in}(K_\sigma)$) to be the subset defined by (2) above if the cycle is counter-clockwise or to be the subset defined by (3) if it is clockwise. The *outer* of $\mathsf{Cntr}(K_\sigma)$ (denoted by $\mathsf{Cntr}_{out}(K_\sigma)$) is defined to be the subset which is not the inner nor the controllability itself. Note that an edge in the SPDI may intersect a controllability kernel. In such cases, we can generate a different SPDI, with the same dynamics but with the edge split into parts, such that each part is completely inside, on or outside the kernel. Although the signatures will obviously change, it is easy to prove that the behaviour of the SPDI remains identical to the original. In the rest of the paper, we will assume that all edges are either completely inside, on or completely outside the kernels. We note that in practice splitting is not necessary since we can just consider parts of edges.

Proposition 3. *Given two edges e and e', one lying completely inside a controllability kernel, and the other outside or on the same controllability kernel, such that ee' is feasible, then there exists a point on the controllability kernel, which is reachable from e and from which e' is reachable.* □

Invariance Kernel. In general, an *invariant set* is a set of points such that for any point in the set, every trajectory starting in such point remains in the set forever and the *invariance kernel* is the largest of such sets. In particular, for an SPDI, given a cyclic signature, an *invariant set* is a set of points which keep rotating in the cycle forever and the *invariance kernel* is the largest of such sets. More formally: A set K is said to be *invariant* if for any $x \in K$ there exists at least one trajectory starting in it and every trajectory starting in x is viable in K. Given a set K, its largest invariant subset is called the *invariance kernel* of K and is denoted by $\mathsf{Inv}(K)$. We need some preliminary definitions before showing how to compute the kernel. The *extended* $\forall$*-predecessor* of an output edge e of a region R is the set of points in R such that every trajectory segment starting in such point reaches e without traversing any other edge. More formally, let R be a region and e be an edge in $out(R)$, then the e*-extended* $\forall$*-predecessor* of I, $\widetilde{\overline{\mathsf{Pre}}}_e(I)$ is defined as: $\widetilde{\overline{\mathsf{Pre}}}_e(I) = \{\mathbf{x} \mid \forall\xi\ .\ (\xi(0) = \mathbf{x} \Rightarrow \exists t \geq 0\ .\ (\xi(t) \in I \wedge \mathsf{Sig}(\xi[0,t]) = e))\}$. It is easy to see that $\widetilde{\overline{\mathsf{Pre}}}_\sigma(I)$ is a polygonal subset of the plane which can be calculated using a similar procedure as for $\overline{\mathsf{Pre}}_\sigma(I)$. We compute the invariance kernel of K_σ as follows:

Theorem 4. *If σ is STAY then $\mathsf{Inv}(K_\sigma) = \widetilde{\overline{\mathsf{Pre}}}_\sigma(\widetilde{\mathsf{Pre}}_\sigma(J_\sigma))$, otherwise it is $\emptyset$.* □

Example 8. Fig. 3-(a) shows the unique invariance kernel of the SPDI given in Example 1. ■

An interesting property of invariance kernels is that the limits are included in the invariance kernel, i.e. $[l^*, u^*] \subseteq \mathsf{Inv}(K_\sigma)$. In other words:

Proposition 4. *The set delimited by the polygons defined by the interval $[l^*, u^*]$ is an invariance set of STAY cycles.* □

The following result relates controllability and invariance kernels.

Proposition 5. *If σ is STAY then $\mathsf{Cntr}(K_\sigma) \subseteq \mathsf{Inv}(K_\sigma)$.* □

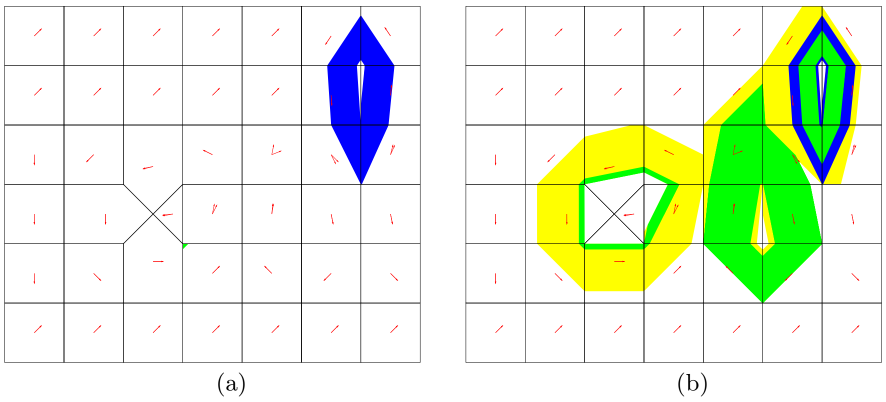


Fig. 3. (a) Invariance kernel; (b) All the kernels

Example 9. Fig. 3-(b) shows the viability, controllability and invariance kernels of the SPDI given in Example 1. For any point in the viability kernel of a cycle there exists a trajectory which will converge to its controllability kernel (proposition 2). It is possible to see in the picture that $\mathsf{Cntr}(\cdot) \subset \mathsf{Inv}(.)$ (proposition 5). All the above pictures has been obtained with the toolbox SPeeDI$^+$ [PS]. ■

In a similar way as for the controllability kernel, we define $\mathsf{Inv}^l(K_\sigma)$ and $\mathsf{Inv}^u(K_\sigma)$.

3 Semi-separatrix Curves

In this section we define the notion of *separatrix curves*, which are curves dissecting the plane into two mutually non-reachable subsets, and *semi-separatrix curves* which can only be crossed in one direction. All the proofs of this and forthcoming sections may be found in [PS06]. We start by defining these notions independently of SPDIs.

Definition 1. *Let $K \subseteq \mathbb{R}^2$. A* separatrix *in K is a closed curve γ partitioning K into three sets K_A, K_B and γ itself, such that K_A, K_B and γ are pairwise disjoint, $K = K_A \cup K_B \cup \gamma$ and the following conditions hold: (1) For any point $\mathbf{x}_0 \in K_A$ and trajectory ξ, with $\xi(0) = \mathbf{x}_0$, there is no t such that $\xi(t) \in K_B$; and (2) For any point $\mathbf{x}_0 \in K_B$ and trajectory ξ, with $\xi(0) = \mathbf{x}_0$, there is no t such that $\xi(t) \in K_A$. If only one of the above conditions holds then we say that the curve is a* semi-separatrix. *If only condition 1 holds, then we say that K_A is the* inner *of γ (written γ_{in}) and K_B is the* outer *of γ (written γ_{out}). If only condition 2 holds, K_B is the* inner *and K_A is the* outer *of γ.*

Notice that, as in the case of the controllability kernel, an edge of the SPDI may be split into two by a semi-separatrix — part inside, and part outside. As before, we can split the edge into parts, such that each part is completely inside, or completely outside the semi-separatrix.

The above notions are extended to SPDIs straightforwardly. The set of all the separatrices of an SPDI $\mathcal{S}$ is denoted by $\mathsf{Sep}(\mathcal{S})$, or simply Sep.

Now, let $\sigma = e_1 \dots e_n e_1$ be a simple cycle, $\angle_{\mathbf{a}_i}^{\mathbf{b}_i}$ $(1 \leq i \leq n)$ be the dynamics of the regions for which e_i is an entry edge and $I = [l, u]$ an interval on edge e_1. Remember that $\mathsf{Succ}_{e_1 e_2}(I) = F(I \cap S_1) \cap J_1$, where $F(x) = [a_1 x + b_1, a_2 x + b_2]$. Let $\mathbf{l}$ be the vector corresponding to the point on e_1 with local coordinates l and $\mathbf{l}'$ be the vector corresponding to the point on e_2 with local coordinates $F(l)$ (similarly, we define $\mathbf{u}$ and $\mathbf{u}'$ for $F(u)$). We define first $\overline{\mathsf{Succ}}_{e_1}^{\mathbf{b}_1}(I) = \{\mathbf{l} + \alpha(\mathbf{l}' - \mathbf{l}) \mid 0 < \alpha < 1\}$ and $\overline{\mathsf{Succ}}_{e_1}^{\mathbf{a}_1}(I) = \{\mathbf{u} + \alpha(\mathbf{u}' - \mathbf{u}) \mid 0 < \alpha < 1\}$. We extend these definitions in a straight way to any (cyclic) signature $\sigma = e_1 \dots e_n e_1$, denoting them by $\overline{\mathsf{Succ}}_{\sigma}^{\mathbf{b}}(I)$ and $\overline{\mathsf{Succ}}_{\sigma}^{\mathbf{a}}(I)$, respectively; we can compute them similarly as for $\overline{\mathsf{Pre}}$. Whenever applied to the fixpoint $I^* = [l^*, u^*]$, we denote $\overline{\mathsf{Succ}}_{\sigma}^{\mathbf{b}}(I^*)$ and $\overline{\mathsf{Succ}}_{\sigma}^{\mathbf{a}}(I^*)$ by ξ_{σ}^{l} and ξ_{σ}^{u} respectively. Intuitively, ξ_{σ}^{l} (ξ_{σ}^{u}) denotes the piece-wise affine closed curve defined by the leftmost (rightmost) fixpoint l^* (u^*).

We show now how to identify semi-separatrices for simple cycles.

Theorem 5. *Given an SPDI, let σ be a simple cycle, then the following hold:*

1. *If σ is EXIT-RIGHT then ξ_{σ}^{l} is a semi-separatrix curve (filtering trajectories from "left" to "right");*
2. *If σ is EXIT-LEFT then ξ_{σ}^{u} is a semi-separatrix curve (filtering trajectories from "right" to "left");*
3. *If σ is STAY, then the two polygons defining the invariance kernel ($\mathsf{Inv}^{l}(K_{\sigma})$ and $\mathsf{Inv}^{u}(K_{\sigma})$), are semi-separatrices.* □

In the case of STAY cycles, ξ_{σ}^{l} and ξ_{σ}^{u} are also semi-separatrices. Notice that in the above result, computing a semi-separatrix depends only on one simple cycle, and the corresponding algorithm is then reduced to find simple cycles in the SPDI and checking whether it is STAY, EXIT-RIGHT or EXIT-LEFT. DIE cycles induce an infinite number of semi-separatrices and are not treated in this setting.

Example 10. Fig. 4 shows all the semi-separatrices of the SPDI given in Example 1, obtained as shown in Theorem 5. The small arrows traversing the semi-separatrices show the inner and outer of each semi-separatrix: a trajectory may traverse the semi-separatrix following the direction of the arrow, but not vice-versa. ■

The following two results relate feasible signatures and semi-separatrices.

Proposition 6. *If, for some semi-separatrix γ, $e \in \gamma_{in}$ and $e' \in \gamma_{out}$, then the signature ee' is not feasible.* □

Proposition 7. *If, for some semi-separatrix γ, and signature σ (of at least length 2), then, if head$(\sigma) \in \gamma_{in}$ and last$(\sigma) \in \gamma_{out}$, σ is not feasible.* □

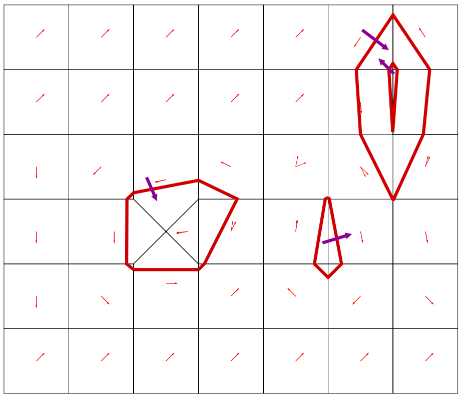


Fig. 4. Semi-separatrices

4 State-Space Reduction Using Semi-separatrices

Semi-separatrices partition the state space into two parts[3] – once one crosses such a border, all states outside the region can be ignored. We present a technique, which, given an SPDI and a reachability question, enables us to discard portions of the state space based on this information. The approach is based on identifying *inert* states (edges in the SPDI) not playing a role in the reachability analysis.

Definition 2. *Given an SPDI $\mathcal{S}$, a semi-separatrix $\gamma \in \mathsf{Sep}$, a source edge $e0$ and a destination edge $e1$, an edge e is said to be* inert *if it lies outside the semi-separatrix while $e0$ lies inside, or it lies inside, while $e1$ lies outside:*

$$inert^{\gamma}_{e0 \to e1} = \{e : \mathcal{E} \mid e0 \in \gamma_{in} \wedge e \in \gamma_{out}\} \cup \{e : \mathcal{E} \mid e1 \in \gamma_{out} \wedge e \in \gamma_{in}\}.$$

We can prove that these inert edges can never appear in a feasible signature:

Lemma 2. *Given an SPDI $\mathcal{S}$, a semi-separatrix γ, a source edge $e0$ and a destination edge $e1$, and a feasible signature $e0\sigma e1$ in $\mathcal{S}$. No inert edge from $inert^{\gamma}_{e0\to e1}$ may appear in $e0\sigma e1$.* □

Given an SPDI, we can reduce the state space by discarding inert edges.

Definition 3. *Given an SPDI $\mathcal{S}$, a semi-separatrix γ, a source edge $e0$ and a destination edge $e1$, we define the reduced SPDI $\mathcal{S}^{\gamma}_{e0\to e1}$ to be the same as $\mathcal{S}$ but without the inert edges.*

Clearly, the resulting SPDI is not bigger than the original one. Finally, we prove that checking reachability on the reduced SPDI is equivalent to checking reachability on the original SPDI:

[3] Here, we do not consider the semi-separatrix itself.

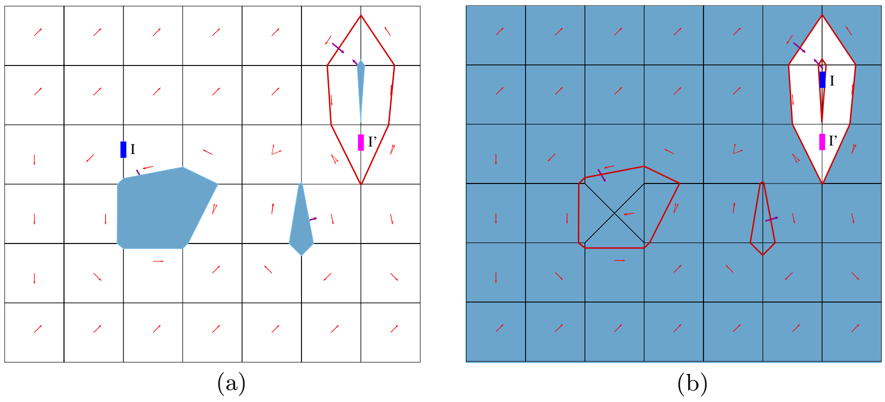


Fig. 5. Reduction using semi-separatrices

Theorem 6. *Given an SPDI $\mathcal{S}$, a semi-separatrix γ, and edges $e0$ and $e1$, then, $e1$ is reachable from $e0$ in $\mathcal{S}$ if and only if $e1$ is reachable from $e0$ in $\mathcal{S}^{\gamma}_{e0\to e1}$.* □

We have shown, that once semi-separatrices are identified, given a reachability question, we can reduce the size of the SPDI to be verified by removing inert edges of all the known semi-separatrices.

Example 11. The shaded areas of Fig. 5 (a) and (b) are examples of subsets of the SPDI edges of the reachability graph, eliminated by the reduction presented in this section applied to all semi-separatrices, when answering reachability questions (in this case to the question: Is I' reachable from I?). ■

This result enables us to verify SPDIs much more efficiently. It is important to note that model-checking an SPDI requires identification of simple loops, which means that the calculation of the semi-separatrices is not more expensive than the initial pass of the model-checking algorithm. Furthermore, we can perform this analysis only once for an SPDI and store the information to be used in any reachability analysis on that SPDI. Reduction, however, can only be applied once we know the source and destination states.

5 State-Space Reduction Using Kernels

5.1 State-Space Reduction Using Kernels

We have already shown that any invariant set is essentially a pair of semi-separatices, and since the invariance kernel is an invariant set, we can use the results from section 4 to abstract an SPDI using invariance kernels. We now turn our attention to state space reduction using controllability kernels:

Definition 4. *Given an SPDI $\mathcal{S}$, a loop σ, a source edge $e0$ and a destination edge $e1$, an edge e is said to be* redundant *if it lies on the opposite side of a controllability kernel as both $e0$ and $e1$:*

$$\begin{aligned} \mathit{redundant}^{\sigma}_{e0\to e1} &= \{e : \mathcal{E} \mid \{e0,\ e1\} \subseteq \mathsf{Cntr}_{in}(\sigma) \cup \mathsf{Cntr}(\sigma) \wedge e \in \mathsf{Cntr}_{out}(\sigma)\} \\ &\cup \{e : \mathcal{E} \mid \{e0,\ e1\} \subseteq \mathsf{Cntr}_{out}(\sigma) \cup \mathsf{Cntr}(\sigma) \wedge e \in \mathsf{Cntr}_{in}(\sigma)\}. \end{aligned}$$

We can prove that we can do without these edges to check feasibility:

Lemma 3. *Given an SPDI $\mathcal{S}$, a loop σ, a source edge $e0$, a destination edge $e1$, and a feasible signature $e0\sigma e1$ then there exists a feasible signature $e0\sigma' e1$ such that σ' contains no redundant edge from $\mathit{redundant}^{\sigma}_{e0\to e1}$.* □

Given an SPDI, we can reduce the state space by discarding redundant edges.

Definition 5. *Given an SPDI $\mathcal{S}$, a loop σ, a source edge $e0$ and a destination edge $e1$, we define the reduced SPDI $\mathcal{S}^{\sigma}_{e0\to e1}$ to be the same as $\mathcal{S}$ but without redundant edges.*

Clearly, the resulting SPDI is smaller than the original one. Finally, based on proposition 3, we prove that reachability on the reduced SPDI is equivalent to reachability on the original one:

Theorem 7. *Given an SPDI $\mathcal{S}$, a loop σ, a source edge $e0$ and a destination edge $e1$, then, $e1$ is reachable from $e0$ in $\mathcal{S}$ if and only if $e1$ is reachable from $e0$ in $\mathcal{S}^{\sigma}_{e0\to e1}$.* □

Given a loop which has a controllability kernel, we can thus reduce the state space to explore. In practice, we apply this state space reduction for each controllability kernel in the SPDI. Once a loop in the SPDI is identified, it is straightforward to apply the reduction algorithm.

5.2 Immediate Answers to Reachability Questions

By definition of the controllability kernel, any two points inside it are mutually reachable. This can be used to answer reachability questions in which both the source and destination edge lie (possibly partially) within the same controllability kernel. Using proposition 2, we know that any point in the viability kernel of a loop can eventually reach the controllability kernel of the same loop, which allows us to relax the condition about the source edge to just check whether it (partially) lies within the viability kernel. Finally, we note that the union of non-disjoint controllability sets is itself a controllability set which allows us to extend the result to work for a collection of loops whose controllability kernels form a strongly connected set.

Definition 6. *We extend viability and controllability kernels for a set of loops Σ by taking the union of the kernels of the individual loops, with $\mathsf{Viab}(K_\Sigma)$ being the union of all viability kernels of loops in Σ, and similarly $\mathsf{Cntr}(K_\Sigma)$.*

Definition 7. *Two loops σ and σ' are said to be compatible ($\sigma \leftrightsquigarrow \sigma'$) if their controllability kernels overlap: $\mathsf{Cntr}(K_\sigma) \cap \mathsf{Cntr}(K_{\sigma'}) \neq \emptyset$.*

We extend the notion of compatibility to a set of loops Σ to mean that all loops in the set are transitively compatible: $\forall \sigma,\ \sigma' \in \Sigma \cdot \sigma \leftrightsquigarrow^{*} \sigma'$.

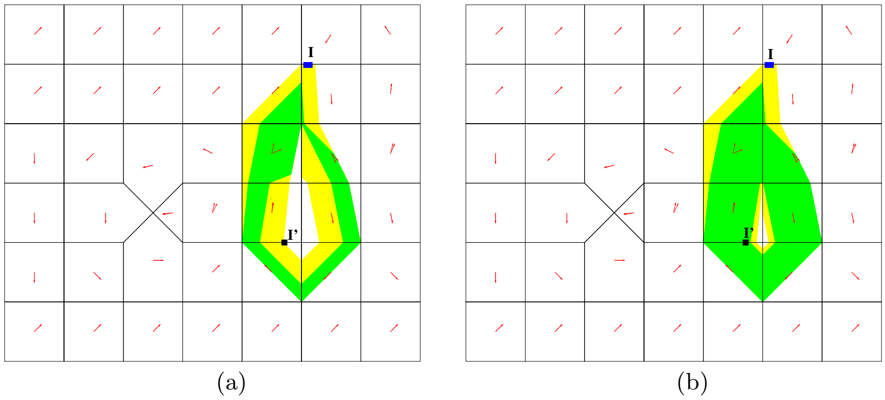


Fig. 6. Answering reachability using kernels

Based on proposition 2, we can prove the following:

Theorem 8. *Given a source edge e_{src} and a destination edge e_{dst}, if for some compatible set of loops Σ, $e_{src} \cap \mathsf{Viab}(K_\Sigma) \neq \emptyset$ and $e_{dst} \cap \mathsf{Cntr}(K_\Sigma) \neq \emptyset$, then e_{dst} is reachable from e_{src}.* □

Example 12. Fig. 6-(a) shows a viability and a controllability kernel of a cycle and two intervals I and I'. Whether I' is reachable from I cannot be answered immediately in this case, but Fig. 6-(b) shows the overlapping of the viability and controllability kernels depicted in Fig. 6-(a) with the kernels of an inner cycle. I' thus lies in a compatible controllability kernel, and we can immediately conclude (by theorem 8) that I' is reachable from I. ■

In practice, we propose to use these theorems to enable answering certain reachability questions without having to explore the complete state space. It can also be used to reduce reachability questions to (possibly) simpler ones by trying to reach a viability kernel rather than a particular edge. As in the case of semi-separatrices, a preliminary analysis of an SPDI's kernels be used in all subsequent reachability queries. SPeeDI [APSY02] starts by calculating and caching all loops in the given SPDI, and can thus easily identify maximal compatible sets of loops. Combining this technique with the semi-separatrix reduction technique we envisage substantial gains.

6 Concluding Remarks

We have introduced the concept of semi-separatrices for polygonal hybrid systems, and presented non-iterative algorithms to calculate them.

Using semi-separatrices and kernels, we presented techniques to improve reachability analysis on SPDIs. In all cases, the techniques require the identification and analysis of loops in the SPDI. When multiple reachability questions are to be asked about the same SPDI, this information can be gathered once to avoid repeated analysis. We note that most of this information is still required in reachability analysis, and thus no extra work is required to perform the optimization presented in this paper. The results presented all depend on checking whether an edge lies within a given polygon which can be efficiently checked using standard geometrical techniques frequently used in computer graphics such as using the odd-parity test [FvDFH96]. Sometimes, using kernel information, we can answer reachability questions without any further analysis. In other cases, we use semi-separatrices and controllability kernels to reduce the size of the SPDI.

Our work is obviously restricted to planar systems, which enables us to compute these kernels exactly. In higher dimensions and hybrid systems with higher complexity, calculation of kernels is not computable. Other work is thus based on calculations of approximations of these kernels (e.g., [ALQ$^+$01b, ALQ$^+$01a, SP02]). We are not aware of any work using kernels and semi-separatrices to reduce the state-space of the reachability graph as presented in this paper.

We have built a toolset SPeeDI [APSY02] for the analysis of SPDIs. We have recently extended this toolset to SPeeDI$^+$ [PS] which calculates kernels of SPDIs. We are currently exploring the implementation of the optimizations presented in this paper to improve the efficiency of SPeeDI$^+$. We are also investigating other applications of these kernels in the model-checking of SPDIs.

References

[AD94] R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[ALQ$^+$01a] J.-P. Aubin, J. Lygeros, M. Quincampoix, S. Sastry, and N. Seube. Towards a viability theory for hybrid systems. In *European Control Conference*, 2001.

[ALQ$^+$01b] J.-P. Aubin, J. Lygeros, M. Quincampoix, S. Sastry, and N. Seube. Viability and invariance kernels of impulse differential inclusions. In *Conference on Decision and Control*, volume 40 of *IEEE*, December 2001.

[APSY02] E. Asarin, G. Pace, G. Schneider, and S. Yovine. SPeeDI: a verification tool for polygonal hybrid systems. In *CAV'2002*, volume LNCS 2404, 2002.

[ASY01] E. Asarin, G. Schneider, and S. Yovine. On the decidability of reachability for planar differential inclusions. In *HSCC'2001*, volume LNCS 2034, 2001.

[ASY02] E. Asarin, G. Schneider, and S. Yovine. Towards computing phase portraits of polygonal differential inclusions. In *HSCC'02*. LNCS 2289, 2002.

[Aub01] J.-P. Aubin. The substratum of impulse and hybrid control systems. In *HSCC'01*, volume 2034 of *LNCS*, pages 105–118. Springer, 2001.

[DV95] A. Deshpande and P. Varaiya. Viable control of hybrid systems. In *Hybrid Systems II*, number 999 in LNCS, pages 128–147, 1995.

[FvDFH96] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer graphics (2nd ed. in C): principles and practice.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1996.

[HKPV95] T.A. Henzinger, P.W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? In *STOC'95*, pages 373–382. ACM Press, 1995.

[LPY01] G. Lafferriere, G. Pappas, and S. Yovine. Symbolic reachability computation of families of linear vector fields. *Journal of Symbolic Computation*, 32(3):231–253, September 2001.

[MP93] O. Maler and A. Pnueli. Reachability analysis of planar multi-linear systems. In *CAV'93*, pages 194–209. LNCS 697, Springer Verlag, July 1993.

[MS00] A. Matveev and A. Savkin. *Qualitative theory of hybrid dynamical systems*. Birkhäuser Boston, 2000.

[PS] G. Pace and G. Schneider. SPeeDI$^+$. `http:\\www.cs.um.edu.mt\speedi`.

[PS06] G. Pace and G. Schneider. Static analysis of SPDIs for state-space reduction. Technical Report 336, Department of Informatics, University of Oslo, April 2006.

[Sch02] G. Schneider. *Algorithmic Analysis of Polygonal Hybrid Systems*. PhD thesis, VERIMAG – UJF, Grenoble, France, July 2002.

[Sch04] G. Schneider. Computing invariance kernels of polygonal hybrid systems. *Nordic Journal of Computing*, 11(2):194–210, 2004.

[SJSL00] S. Simić, K. Johansson, S. Sastry, and J. Lygeros. Towards a geometric theory of hybrid systems. In *HSCC'00*, number 1790 in LNCS, 2000.

[SP02] P. Saint-Pierre. Hybrid kernels and capture basins for impulse constrained systems. In *HSCC'02*, volume 2289 of *LNCS*. Springer-Verlag, 2002.

On the Expressiveness of MTL with Past Operators

Pavithra Prabhakar and Deepak D'Souza

Department of Computer Science and Automation,
Indian Institute of Science, Bangalore 560012, India
{pavithra, deepakd}@csa.iisc.ernet.in

Abstract. We compare the expressiveness of variants of Metric Temporal Logic (MTL) obtained by adding the past operators 'S' and 'S_I'. We consider these variants under the "pointwise" and "continuous" interpretations over both finite and infinite models. Among other results, we show that for each of these variants the continuous version is strictly more expressive than the pointwise version. We also prove a counter-freeness result for MTL which helps to carry over some results from [3] for the case of infinite models to the case of finite models.

1 Introduction

The timed temporal logic *Metric Temporal Logic* (MTL) [6] has received much attention in the literature on the verification of real-time systems. It is interpreted over (finite or infinite) timed behaviours and extends the until operator of classical temporal logic with an interval which specifies the time distance within which the formula must be satisfied. Over dense time the logic has traditionally been interpreted in either of the two ways which have come to be known as the "pointwise" and the "continuous" semantics. In the pointwise version temporal assertions are interpreted only at time points where an "action" or "event" happens in the observed timed behaviour of a system, whereas in the continuous version one is allowed to assert formulas at arbitrary time points between events as well. For instance consider a timed word comprising two events: an a which happens at time 1 and a b which occurs at time 3. Then the MTL formula $\Diamond_{[1,1]}b$ ("a b occurs at a distance of 1 time unit") is not true at any point in this model in the pointwise semantics, since there is no action point from which the action b happens at a distance of 1 time unit. However in the continuous semantics the formula is true at the time instant 2 in the model since at this point the event b occurs at a time distance of 1.

There are many results in the literature regarding the decidability of these logics and the reader is referred to [2,1,8,9] for more details. In this paper we are more interested in the expressiveness of the variants of MTL obtained by adding the past operators S ("since") and S_I (interval constrained "since"), under the pointwise and continuous interpretations, for both finite and infinite models. We will refer to these logics as MTL_S and MTL_{S_I} respectively, and add the superscripts *pw* and *c* to denote the pointwise and continuous versions of the logics respectively.

It is easy to see that for each of these variants the continuous version is at least as expressive as the pointwise version, as one can characterize the action points in the continuous semantics, and hence mimic the pointwise interpretation. There have also been some strict containment results. In [3], it is shown that the language L_{2b}, which consists of timed words in which there are two occurrences of b's in the interval $(0, 1)$, is not expressible by MTL in the pointwise semantics but is expressible by MTL in the continuous semantics, and also by MTL_S in the pointwise semantics. It is also shown that the language L_{last_a}, which consists of timed words in which there is an action at time 1 which is preceded by an a, is not expressible by MTL in the continuous semantics but is expressible by MTL_S in the continuous semantics. However these results hold for the case of infinite words and do not extend readily to the case of finite words. The proofs exploit the fact that the models are infinite by using the property that the futures of two distinct points in the constructed models are the same (which is never true for any finite model).

In [4], it is shown that MTL in the continuous semantics is strictly more expressive than MTL in the pointwise semantics for the case of finite words. This is done by showing that the language L_{ni} (for "no insertions") over the alphabet $\{a, b\}$, consisting of timed words in which for every two consecutive a's the time period between them translated by one time unit does not contain any events, is expressible in the continuous semantics, but its expressibility in the pointwise semantics would render the logic undecidable, contradicting the decidability result in [8].

The diagram below shows the known relative expressiveness results. The solid arrows denote "strict containment", the dashed arrows represent "containment", the dashed line says that "relative expressiveness in not known", and the absence of an arrow, transitive arrow, or line, denotes "incomparable".

$$\begin{tikzcd}
 & \mathrm{MTL}^{c}_{S_I} \\
\mathrm{MTL}^{pw}_{S_I} \arrow[ur, dashed] \arrow[dr, dash, dashed] \arrow[dddr, dash, dashed] & \\
 & \mathrm{MTL}^{c}_{S} \arrow[uu, dashed] \\
\mathrm{MTL}^{pw}_{S} \arrow[uu, dashed] \arrow[ur, dashed] \arrow[dr, dash, dashed] & \\
 & \mathrm{MTL}^{c} \arrow[uu, dashed] \\
\mathrm{MTL}^{pw} \arrow[uu, dashed] \arrow[ur] &
\end{tikzcd}$$

Finite words

$$\begin{tikzcd}
 & \mathrm{MTL}^{c}_{S_I} \\
\mathrm{MTL}^{pw}_{S_I} \arrow[ur, dashed] \arrow[dr, dash, dashed] \arrow[dddr, dash, dashed] & \\
 & \mathrm{MTL}^{c}_{S} \arrow[uu, dashed] \\
\mathrm{MTL}^{pw}_{S} \arrow[uu, dashed] \arrow[ur, dashed] \arrow[dr, dash, dashed] & \\
 & \mathrm{MTL}^{c} \arrow[uu] \\
\mathrm{MTL}^{pw} \arrow[uu] \arrow[ur] &
\end{tikzcd}$$

Infinite words

In this paper we first show a way of carrying over the results of [3] to the case of finite words by proving a kind of "counter-freeness" property of MTL. We show that for a given MTL formula φ, there cannot exist finite timed words μ, τ and ν, such that for infinitely many i's, $\mu\tau^i\nu$ is a model of φ, and for infinitely many i's, $\mu\tau^i\nu$ is not a model of φ. This is true for the pointwise semantics and we show a similar result for the continuous semantics which takes into account the "granularity" of φ. These results help us in extending the results of [3] to finite models.

Next we show that each of the continuous versions of the logic is strictly more expressive than its pointwise counterpart. We do so by showing that the language L_{2ins}, which consists of timed words which contain two consecutive a's such that the time period between them when translated by one time unit contains two a's, is not expressible by MTL_{S_I} (and hence by MTL_S and MTL) in the pointwise semantics, but is expressible by MTL (and hence by MTL_S and MTL_{S_I}) in the continuous semantics.

Finally we show that the language L_{em} (for "exact match"), which consists of timed words such that for every a in the interval $(0, 1)$ there is an a in the interval $(1, 2)$ at distance 1 from it, and vice versa, is expressible by MTL_{S_I} in the pointwise semantics but not by MTL_S in the pointwise semantics. This result holds for both finite and infinite words.

The picture below summarizes the relative expressiveness of the various version of MTL after the work in this paper.

$$\begin{tikzcd}
 & \mathrm{MTL}^c_{S_I} \\
\mathrm{MTL}^{pw}_{S_I} \arrow[ur] \arrow[dr, dashed, no head] & \\
 & \mathrm{MTL}^c_S \arrow[uu, dashed] \\
\mathrm{MTL}^{pw}_S \arrow[uu] \arrow[ur] & \\
 & \mathrm{MTL}^c \arrow[uu] \\
\mathrm{MTL}^{pw} \arrow[uu] \arrow[ur] &
\end{tikzcd}$$

Finite words

$$\begin{tikzcd}
 & \mathrm{MTL}^c_{S_I} \\
\mathrm{MTL}^{pw}_{S_I} \arrow[ur] \arrow[dr, dashed, no head] & \\
 & \mathrm{MTL}^c_S \arrow[uu, dashed] \\
\mathrm{MTL}^{pw}_S \arrow[uu] \arrow[ur] & \\
 & \mathrm{MTL}^c \arrow[uu] \\
\mathrm{MTL}^{pw} \arrow[uu] \arrow[ur] &
\end{tikzcd}$$

Infinite words

We note that it is still open whether $\mathrm{MTL}^c_{S_I}$ is strictly more expressive than MTL^c_S and whether $\mathrm{MTL}^{pw}_{S_I}$ is contained in or incomparable with MTL^c_S.

At some places in the sequel, we have only sketched the proofs due to lack of space. However, the details can be found in [10].

2 Preliminaries

We begin with some preliminary definitions. As usual, A^* and A^ω will denote the set of finite words and the set of infinite words over an alphabet A, respectively. For a finite word $w = a_1 \cdots a_n$ we use $|w|$ to denote the length of w (in this case n). Given finite words u and v, we denote the concatenation of u followed by v as $u \cdot v$, or just uv. We use u^i to denote the concatenation of u with itself i times, and u^ω to denote the infinite word obtained by repeated concatenation of u. We extend these notations to subsets of A^* in the standard way.

The set of non-negative and positive real numbers will be denoted by $\mathbb{R}_{\geq 0}$ and $\mathbb{R}_{>0}$ respectively, the set of positive rational numbers by $\mathbb{Q}_{>0}$, and the set of non-negative integers by $\mathbb{N}$.

We now define finite and infinite timed words which are sequences of action and time pairs. An *infinite timed word* α over an alphabet Σ is an element of $(\Sigma \times \mathbb{R}_{>0})^\omega$ of the form $(a_1, t_1)(a_2, t_2) \cdots$ satisfying:

– (Strict-monotonicity) $t_1 < t_2 < \cdots$.
– (Progressiveness) For every $t \in \mathbb{R}_{\geq 0}$, there exists $i \in \mathbb{N}$ such that $t_i > t$.

Wherever convenient we will also denote the timed word α above as a sequence of delay and action pairs $(d_1, a_1)(d_2, a_2) \cdots$, where for each i, $d_i = t_i - t_{i-1}$. Here and elsewhere we use the convention that t_0 denotes the time point 0.

A *finite timed word* over Σ is an element of $(\Sigma \times \mathbb{R}_{>0})^*$ which satisfies the strict monotonicity condition above. Given $\sigma = (a_1, t_1)(a_2, t_2) \cdots (a_n, t_n)$, we use $time(\sigma)$ to denote the time of the last action, namely t_n. The delay representation for the above finite timed word σ is $(d_1, a_1) \cdots (d_n, a_n)$ where for each i, $d_i = t_i - t_{i-1}$. Given finite timed words σ and ρ, the delay representation for the concatenation of σ followed by ρ is the concatenation of the delay representations of σ and ρ. We will use $T\Sigma^*$ for the set of all finite timed words over Σ, and $T\Sigma^\omega$ for the set of all infinite timed words over Σ.

We now give the syntax and semantics of the two versions of the logic MTL_{S_I}. Let us fix a finite alphabet Σ for the rest of this section. The formulas of MTL_{S_I} over the alphabet Σ are built up from symbols in Σ by boolean connectives and time-constrained versions of the temporal logic operators U ("until") and S ("since"). The formulas of MTL_{S_I} over an alphabet Σ are inductively defined as follows:

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

where $a \in \Sigma$ and I is an interval with end-points which are rational or ∞.

The models for both the pointwise and continuous interpretations will be timed words over Σ. With the aim of having a common syntax for the pointwise and continuous versions, we use "until" and "since" operators which are "strict" in their first argument.

We first define the *pointwise* semantics for MTL_{S_I} for finite words. Given an MTL_{S_I} formula φ, a finite timed word $\sigma = (a_1, t_1)(a_2, t_2) \cdots (a_n, t_n)$ and a position $i \in \{0, \ldots, n\}$ denoting the leftmost time point 0 or one of the action points $t_1, t_2, \cdots, t_n$, the satisfaction relation $\sigma, i \models_{pw} \varphi$ (read "σ at position i satisfies φ in the pointwise semantics") is inductively defined as:

$$\begin{array}{lll}
\sigma, i \models_{pw} a & \text{iff} & a_i = a. \\
\sigma, i \models_{pw} \neg\varphi & \text{iff} & \sigma, i \not\models_{pw} \varphi. \\
\sigma, i \models_{pw} \varphi_1 \vee \varphi_2 & \text{iff} & \sigma, i \models_{pw} \varphi_1 \text{ or } \sigma, i \models_{pw} \varphi_2. \\
\sigma, i \models_{pw} \varphi_1 U_I \varphi_2 & \text{iff} & \exists j : i \leq j \leq |\sigma| \text{ such that } t_j - t_i \in I,\ \sigma, j \models_{pw} \varphi_2, \\
 & & \text{and } \forall k \text{ such that } i < k < j,\ \sigma, k \models_{pw} \varphi_1. \\
\sigma, i \models_{pw} \varphi_1 S_I \varphi_2 & \text{iff} & \exists j : 0 \leq j \leq i \text{ such that } t_i - t_j \in I,\ \sigma, j \models_{pw} \varphi_2, \\
 & & \text{and } \forall k \text{ such that } j < k < i,\ \sigma, k \models_{pw} \varphi_1.
\end{array}$$

The timed language defined by an MTL_{S_I} formula φ in the pointwise semantics over finite timed words is given by $L^{pw}(\varphi) = \{\sigma \in T\Sigma^* \mid \sigma, 0 \models_{pw} \varphi\}$. We will use $\text{MTL}^{pw}_{S_I}$ to denote the pointwise interpretation of this logic.

We now turn to the continuous semantics. Given an MTL_{S_I} formula φ, a finite timed word $\sigma = (a_1, t_1)(a_2, t_2) \cdots (a_n, t_n)$ and a time $t \in \mathbb{R}_{\geq 0}$, such that $0 \leq t \leq \mathit{time}(\sigma)$, the satisfaction relation $\sigma, t \models_c \varphi$ (read "σ at time t satisfies φ in the continuous semantics") is inductively defined as follows:

$$\begin{array}{lll}
\sigma, t \models_c a & \text{iff} & \exists i \,\text{such that}\, t_i = t \,\text{and}\, a_i = a. \\
\sigma, t \models_c \neg\varphi & \text{iff} & \sigma, t \not\models_c \varphi. \\
\sigma, t \models_c \varphi_1 \vee \varphi_2 & \text{iff} & \sigma, t \models_c \varphi_1 \,\text{or}\, \sigma, t \models_c \varphi_2. \\
\sigma, t \models_c \varphi_1 U_I \varphi_2 & \text{iff} & \exists t' \,\text{such that}\, t \leq t' \leq \mathit{time}(\sigma),\ t' - t \in I,\ \sigma, t' \models_c \varphi_2 \\
 & & \text{and } \forall t'' \text{ such that } t < t'' < t',\ \sigma, t'' \models_c \varphi_1. \\
\sigma, t \models_c \varphi_1 S_I \varphi_2 & \text{iff} & \exists t' \,\text{such that}\, 0 \leq t' \leq t,\ t - t' \in I,\ \sigma, t' \models_c \varphi_2 \\
 & & \text{and } \forall t'' \text{ such that } t' < t'' < t,\ \sigma, t'' \models_c \varphi_1.
\end{array}$$

The timed language defined by an MTL_{S_I} formula φ in the continuous semantics over finite timed words is defined as $L^c(\varphi) = \{\sigma \in T\Sigma^* \mid \sigma, 0 \models_c \varphi\}$. We will use $\mathrm{MTL}^c_{S_I}$ to denote this continuous interpretation of the MTL_{S_I} formulas.

We can similarly define the semantics for infinite timed words. The only change would be to replace $\mathit{time}(\sigma)$ and $|\sigma|$ by ∞.

We define the following derived operators which we will make use of in the sequel. Syntactically, $\Diamond_I \varphi$ is $\top U_I \varphi$, $\Box_I \varphi$ is $\neg \Diamond_I \neg \varphi$, $\varphi_1 U \varphi_2$ is $\varphi_1 U_{(0,\infty)} \varphi_2$, $\Diamond \varphi$ is $\Diamond_{(0,\infty)} \varphi$, $\Box \varphi$ is $\neg \Diamond \neg \varphi$, $\diamondminus_I \varphi$ is $\top S_I \varphi$, $\boxminus_I \varphi$ is $\neg \diamondminus_I \neg \varphi$, $\varphi_1 S \varphi_2$ is $\varphi_1 S_{(0,\infty)} \varphi_2$, $\diamondminus \varphi$ is $\diamondminus_{(0,\infty)} \varphi$ and $\boxminus \varphi$ is $\neg \diamondminus \neg \varphi$.

The fragment of MTL_{S_I} without the S_I operator will be called MTL. The fragment of MTL_{S_I} obtained by replacing S_I with the derived operator S will be called MTL_S. We denote their pointwise and continuous interpretations by MTL^{pw} and MTL^c, and MTL^{pw}_S and MTL^c_S respectively. The continuous versions of the above logics can be seen to be at least as expressive as their pointwise versions. This is because one can characterize the occurrence of an action point in a timed word in the continuous semantics using the formula $\varphi_{act} = \bigvee_{a \in \Sigma} a$. We can then force assertions to be interpreted only at these action points.

3 Ultimate Satisfiability of MTL^{pw}

In this section we show that an MTL formula in the pointwise semantics is either ultimately satisfied or ultimately not satisfied over a periodic sequence of timed words, leading to a "counter-freeness" property of MTL.

We first define the notion of when a formula is *ultimately satisfied* or *ultimately not satisfied* over a sequence of finite timed words. Let $\langle \sigma_i \rangle$ be a sequence of finite timed words $\sigma_0, \sigma_1, \cdots$. Given a $j \in \mathbb{N}$ and $\varphi \in \mathrm{MTL}$, we say that $\langle \sigma_i \rangle$ at j *ultimately satisfies* φ, denoted $\langle \sigma_i \rangle, j \models_{us} \varphi$, iff $\exists k \in \mathbb{N} : \forall k' \geq k, \sigma_{k'}, j \models_{pw} \varphi$. We say that $\langle \sigma_i \rangle$ at j *ultimately does not satisfy* φ, denoted $\langle \sigma_i \rangle, j \models_{un} \varphi$, iff $\exists k \in \mathbb{N} : \forall k' \geq k, \sigma_{k'}, j \models_{pw} \neg\varphi$. We refer to the least such k in either case above as the *stability point* of φ at j in $\langle \sigma_i \rangle$.

We now define a *periodic sequence* of timed words. A sequence $\langle \sigma_i \rangle$ of finite timed words is said to be periodic if there exist finite timed words μ, τ and ν, where $|\tau| > 0$, such that $\sigma_i = \mu \tau^i \nu$ for all $i \in \mathbb{N}$.

The following theorem says that a periodic sequence of timed words at a position j either ultimately satisfies a given MTL formula or ultimately does not satisfy it. This is not true in general for a non-periodic sequence. For example, consider the sequence $\langle\sigma_i\rangle$ given by $\sigma_0 = (1,a)$, $\sigma_1 = (1,a)(1,b)$, $\sigma_2 = (1,a)(1,b)(1,a)$, etc. Then the formula $\Diamond(a \wedge \neg\Diamond\top)$, which says that the last action of the timed word is an a, is neither ultimately satisfied nor ultimately not satisfied in $\langle\sigma_i\rangle$ at 0.

Theorem 1. *Let $\langle\sigma_i\rangle$ be a periodic sequence of finite timed words. Let φ be an* MTL *formula and let $j \in \mathbb{N}$. Then either $\langle\sigma_i\rangle, j \models_{us} \varphi$ or $\langle\sigma_i\rangle, j \models_{un} \varphi$.*

Proof. Since $\langle\sigma_i\rangle$ is periodic, there exist timed words $\mu = (d_1,a_1)\cdots(d_l,a_l)$, $\tau = (e_1,b_1)\cdots(e_m,b_m)$ and $\nu = (f_1,c_1)\cdots(f_n,c_n)$, such that $\sigma_i = \mu\tau^i\nu$. Let $\mu\tau^\omega = (a'_1,t'_1)(a'_2,t'_2)\cdots$. We use induction on the structure of φ.

Case $\varphi = a$: If $a'_j = a$, then clearly $\langle\sigma_i\rangle, j \models_{us} \varphi$, otherwise $\langle\sigma_i\rangle, j \models_{un} \varphi$.

Case $\varphi = \neg\psi$: If $\langle\sigma_i\rangle, j \models_{us} \psi$, then $\langle\sigma_i\rangle, j \models_{un} \varphi$. Otherwise, by induction hypothesis, $\langle\sigma_i\rangle, j \models_{un} \psi$ and hence $\langle\sigma_i\rangle, j \models_{us} \varphi$.

Case $\varphi = \eta\vee\psi$: Suppose $\langle\sigma_i\rangle, j \models_{us} \eta$ or $\langle\sigma_i\rangle, j \models_{us} \psi$. Let k be the maximum of the stability points of η and ψ at j in $\langle\sigma_i\rangle$. For all $k' \geq k, \sigma_{k'}, j \models_{pw} \eta$ or for all $k' \geq k, \sigma_{k'}, j \models_{pw} \psi$, and hence for all $k' \geq k, \sigma_{k'}, j \models_{pw} \eta \vee \psi$. Therefore, $\langle\sigma_i\rangle, j \models_{us} \eta\vee\psi$. Otherwise, it is not the case that $\langle\sigma_i\rangle, j \models_{us} \eta$ and it is not the case that $\langle\sigma_i\rangle, j \models_{us} \psi$. By induction hypothesis, $\langle\sigma_i\rangle, j \models_{un} \eta$ and $\langle\sigma_i\rangle, j \models_{un} \psi$. So, $\langle\sigma_i\rangle, j \models_{us} \neg\eta$ and $\langle\sigma_i\rangle, j \models_{us} \neg\psi$. Let k be the maximum of the stability points of $\neg\eta$ and $\neg\psi$ above, at j. Then for all $k' \geq k, \sigma_{k'}, j \models_{pw} \neg\eta$ and for all $k' \geq k, \sigma_{k'}, j \models_{pw} \neg\psi$, and hence for all $k' \geq k, \sigma_{k'}, j \models_{pw} \neg\eta \wedge \neg\psi$. Therefore, $\langle\sigma_i\rangle, j \models_{us} \neg\eta \wedge \neg\psi$ and hence $\langle\sigma_i\rangle, j \models_{us} \neg(\eta \vee \psi)$. So, $\langle\sigma_i\rangle, j \models_{un} \eta \vee \psi$.

Case $\varphi = \eta U_I \psi$: We consider two cases, one in which there exists $j' \geq j$ such that $t_{j'} - t_j \in I$ and $\langle\sigma_i\rangle, j' \models_{us} \psi$ and the other in which the above condition does not hold.

Suppose there exists $j' \geq j$ such that $t_{j'} - t_j \in I$ and $\langle\sigma_i\rangle, j' \models_{us} \psi$. Let j_s be the smallest such j'.

Now suppose for all k such that $j < k < j_s$, $\langle\sigma_i\rangle, k \models_{us} \eta$. Let n_k be the stability point of η at k for each k above and n_{j_s} that of ψ at j_s. Let n' be the maximum of all n_k's and n_{j_s}. So, for all $n'' \geq n'$, $\sigma_{n''}, j \models_{pw} \eta U_I \psi$. Hence $\langle\sigma_i\rangle, j \models_{us} \varphi$.

Otherwise there exists k such that $j < k < j_s$ and $\langle\sigma_i\rangle, k \models_{un} \eta$. Let m_k be the stability point of η at k. For each $j < k' < j_s$ such that $t_{k'} - t_j \in I$, $\langle\sigma_i\rangle, k' \models_{un} \psi$ (because we chose j_s to be the smallest). Let $n_{k'}$ be the stability point of each k' above. Take n' to be the maximum of m_k and $n_{k'}$'s. For all $n'' \geq n'$, $\sigma_{n''}, j \not\models_{pw} \eta U_I \psi$. Hence $\langle\sigma_i\rangle, j \models_{un} \varphi$.

Now turning to the second case, suppose that for all $j' \geq j$ such that $t_{j'} - t_j \in I$, it is not the case that $\langle\sigma_i\rangle, j' \models_{us} \psi$. Then by induction hypothesis, $\langle\sigma_i\rangle, j' \models_{un} \psi$.

Suppose I is bounded. If there is no j' such that $t_{j'} - t_j \in I$, then it is easy to see that $\langle\sigma_i\rangle, j \models_{un} \eta U_I \psi$. Otherwise there exist finite number of j''s which satisfy $t_{j'} - t_j \in I$ and $\langle\sigma_i\rangle, j' \models_{un} \psi$ since I is bounded. Let $n_{j'}$ be the stability point of ψ at each of these $n_{j'}$'s. Take n' to be the maximum of all $n_{j'}$'s. Then for all $n'' \geq n'$, $\sigma_{n''}, j \not\models_{pw} \eta U_I \psi$. Hence $\langle \sigma_i \rangle, j \models_{un} \varphi$.

Suppose I is unbounded. Let $S = \{s_1, s_2, \cdots, s_m\}$ be the suffixes of τ in the order of decreasing length. Thus $s_i = (e_i, b_i) \cdots (e_m, b_m)$. Let $W = \{w_1, w_2, \cdots, w_n\}$ be the suffixes of ν in the order of decreasing length. Let $X = W \cup (S \cdot \tau^* \cdot \nu)$. (We note that we can arrange the timed words in X in the increasing order of length such that the difference in lengths of the adjacent words in this sequence is one and that the succeeding string in the sequence is a prefix of the present. The sequence is $w_n, w_{n-1}, \cdots, w_1, s_n\nu, s_{n-1}\nu, \cdots, s_1\nu$, $s_n\tau\nu, \cdots, s_1\tau\nu, s_n\tau^2\nu, \cdots, s_1\tau^2\nu$, and so on.)

We now claim that ψ is satisfied at 1 for only finitely many timed words from X. Otherwise ψ is satisfied at 1 for infinitely many timed words from $W \cup S \cdot \tau^* \cdot \nu$ and hence for infinitely many from $s_i \cdot \tau^* \cdot \nu$ for some i. By induction hypothesis $\langle \sigma_i \rangle, l + i \models_{us} \psi$ (l is the length of μ) and therefore $\langle \sigma_i \rangle, l + i + cm \models_{us} \psi$, m is the length of τ and $c \in \mathbb{N}$. Since I is unbounded there exists $j' \geq j$ such that $t_{j'} - t_j \in I$ and $\langle \sigma_i \rangle, j' \models_{us} \psi$. This is a contradiction.

Every $j'' > j$ such that $t_{j''} - t_i \in I$ and $j'' < |\mu|$, $\langle \sigma_i \rangle, j'' \models_{un} \psi$ (by the assumption of the present case). Let $n_{j''}$ be the stability point of the j'''s (which are finite in number).

Suppose there is no timed word in X which satisfies ψ at 1. Let n' be the maximum of $n_{j''}$'s. For all $n'' \geq n'$, $\sigma_{n''}, j \not\models_{pw} \eta U_I \psi$. Hence $\langle \sigma_i \rangle, j \models_{un} \eta U_I \psi$.

Suppose there exists a timed word in X which satisfies ψ at 1. Since we proved that they are finite in number, let l' be the length of the largest such timed word.

Suppose that there exists a timed word in X whose length is greater than l' and which does not satisfy η at 1. Let the length of one such timed word be l''. Let n' be a number which is greater than or equal to the maximum of $n_{j''}$'s and which satisfies $|\sigma_{n'}| > max(j, |\mu|) + l''$. Now for all $n'' \geq n'$, $\sigma_{n''}, j \not\models_{pw} \eta U_I \psi$ since the smallest $j' \geq j$ where ψ is satisfied is $|\sigma_{n''}| - l'$ but before that there is the point $|\sigma_{n''}| - l''$ where η is not satisfied. Hence $\langle \sigma_i \rangle, j \models_{un} \varphi$.

Suppose that all timed words in X whose length is greater than l' satisfy η at 1. Now if there exists $j < k \leq |\mu|$ such that $\langle \sigma_i \rangle, k \models_{un} \eta$, then let n' be such that it is larger than the $n_{j''}$'s and the stability point of η at k and $|\sigma_{n'}| > |\mu| + l'$. For all $n'' \geq n'$, $\sigma_{n''}, j \not\models_{pw} \eta U_I \psi$. Hence, $\langle \sigma_i \rangle, j \models_{un} \varphi$. Otherwise for every $j < k \leq |\mu|$, $\langle \sigma_i \rangle, k \models_{us} \eta$. Take n' to be greater than the maximum of the stability point of η at k's and such that $|\sigma_{n'}| > j + n_I + l'$, where n_I is such that $t_{j+n_I} - t_j \in I$. For all $n'' \geq n'$, $\sigma_{n''}, j \models_{pw} \eta U_I \psi$ and hence $\langle \sigma_i \rangle, j \models_{us} \varphi$. $\square$

It is well known that linear-time temporal logic (LTL) and counter-free languages [5,7] are expressively equivalent. We recall that a *counter* in a deterministic finite automaton is a finite sequence of states $q_0 q_1 \cdots q_n$ such that $n > 1$, $q_0 = q_n$ and there exists a non-empty finite word v such that every q_i on reading v reaches q_{i+1} for $i = 0, \cdots, n-1$. A counter-free language is a regular language whose minimal DFA does not contain any counters. It is not difficult to see that the following is an equivalent characterization of counter-free languages. A regular language L is a counter-free language iff there do not exist finite words u, v and

w, where $|v| > 0$, such that $uv^iw \in L$ for infinitely many i's and $uv^iw \notin L$ for infinitely many i's.

We show a similar necessary property for timed languages defined by MTL^{pw} formulas. Let us call a timed language L *counter-free* if there do not exist finite timed words μ, τ and ν, where $|\tau| > 0$, such that $\mu\tau^i\nu \in L$ for infinitely many i's and $\mu\tau^i\nu \notin L$ for infinitely many i's. The following theorem follows from the ultimate satisfiability result for MTL^{pw}.

Theorem 2. *Every timed language of finite words definable in* MTL^{pw} *is counter-free.*

Proof. Suppose that a timed language L is definable in MTL^{pw} by a formula φ, but is not counter-free. Then there exist finite timed words μ, τ and ν, where $|\tau| > 0$, such that $\mu\tau^i\nu \in L$ for infinitely many i's and $\mu\tau^i\nu \notin L$ for infinitely many i's. The periodic sequence $\langle\sigma_i\rangle$ where $\sigma_i = \mu\tau^i\nu$, is neither ultimately satisfied by φ nor ultimately not satisfied by it which is a contradiction to Theorem 1. $\square$

The above theorem, for example, implies that the language L_{even_b}, which consists of timed words in which the number of b's is even, is not expressible in MTL^{pw}. Taking $\mu = \nu = \epsilon$ and $\tau = (1, b)$ implies that L_{even_b} is not counter-free. By Theorem 2, L_{even_b} is not definable in MTL^{pw}.

4 Ultimate Satisfiability of MTL^c

In this section we show an ultimate satisfiability result for the continuous semantics analogous to the one in the previous section for pointwise semantics. We show that an MTL^c formula with "granularity" p is either ultimately satisfied or ultimately not satisfied by a "p-periodic" sequence of finite timed words.

We say that an MTL formula φ has *granularity* p, where $p \in \mathbb{Q}_{>0}$, if all the end-points of the intervals in it are either integral multiples of p, or ∞. We denote by MTL(p) the set of MTL formulas with granularity p. A periodic sequence of timed words $\langle\sigma_i\rangle$ has *period* p if there exist μ, τ and ν such that $time(\tau) = p$ and for each i, $\sigma_i = \mu\tau^i\nu$. Note that every periodic sequence has a unique period.

We now proceed to define the notion of ultimate satisfiability for the continuous semantics. Given a sequence $\langle\sigma_i\rangle$ of finite timed words, $t \in \mathbb{R}_{\geq 0}$ and $\varphi \in$ MTL, we say that $\langle\sigma_i\rangle$ at t *ultimately satisfies* φ in the continuous semantics, denoted $\langle\sigma_i\rangle, t \models^c_{us} \varphi$, iff $\exists j : \forall k \geq j, \sigma_k, t \models_c \varphi$. And we say that $\langle\sigma_i\rangle$ at t *ultimately does not satisfy* φ in the continuous semantics, denoted $\langle\sigma_i\rangle, t \models^c_{un} \varphi$, iff $\exists j : \forall k \geq j, \sigma_k, t \models_c \neg\varphi$.

In the proof of the ultimate satisfiability for the pointwise case in the previous section, we extensively use the argument that if a formula is ultimately satisfied at all points in a bounded interval then there is a point in the periodic sequence after which all timed words in the sequence satisfy the formula at all points in the interval. However the argument fails in the continuous semantics since there are infinitely many time points even in a bounded interval. Towards tackling this problem, we define a *canonical* set of time points in a timed word such that the satisfiability of a formula is invariant between two consecutive points in the set. So given a finite timed word $\sigma = (a_1, t_1) \cdots (a_n, t_n)$ and a $p \in \mathbb{Q}_{>0}$, we define the set of *canonical points* in σ with respect to p to be the set containing 0 and $\{t \mid \exists i, c \in \mathbb{N} : t = t_i - cp\}$. Since this is finite, we can arrange the time points in it in increasing order to get the sequence $r_0 r_1 \cdots r_m$ which we call the *canonical sequence* of σ with respect to p. We mention below some of the immediate properties of a canonical sequence which we will use later.

1. For each $i \in \{0, \cdots, m-1\}$ σ does not contain any action in the interval (r_i, r_{i+1}).
2. Let $t, t' \in [0, \mathit{time}(\sigma)]$ with $t < t'$, such that (t, t') does not contain any r_i. Then for every $c \in \mathbb{N}$, the interval $(t + cp, t' + cp)$ also does not contain any r_i.

Lemma 1. *Let σ be a finite timed word and $p \in \mathbb{Q}_{>0}$. Let $r_0 r_1 \cdots r_m$ be the canonical sequence of σ with respect to p. Let $\varphi \in \mathrm{MTL}(p)$. Then for each $i \in \{0, \cdots, m-1\}$ and for all $t, t' \in (r_i, r_{i+1})$, $\sigma, t \models_c \varphi$ iff $\sigma, t' \models_c \varphi$.* □

With each finite word σ we associate a sequence of delays which specifies the delays between the consecutive canonical points in the canonical sequence. So given a canonical sequence $r_0 r_1 \cdots r_m$ of σ with respect to p, we call the sequence of delays $D = e_1 e_2 \cdots e_m$ an *invariant delay sequence* of σ with respect to p if each $e_i = r_i - r_{i-1}$. Given any subword of σ, $(d_i, a_i) \cdots (d_j, a_j)$, we can associate a delay sequence with it in a natural way which is given by $e_{i'} \cdots e_{j'}$, where i' and j' are such that $\sum_{k=1}^{i-1} d_k = \sum_{k=1}^{i'-1} e_k$ and $\sum_{k=i}^{j} d_k = \sum_{k=i'}^{j'} e_k$.

Proposition 1. *Let $\sigma = \mu\tau\nu$ be a finite timed word such that $\mathit{time}(\tau) = p$ and $p \in \mathbb{Q}_{>0}$. Let $D = D_1 D_2 D_3$ be the invariant delay sequence of σ with respect to p where D_1, D_2 and D_3 are the delay sequences corresponding to the subwords μ, τ and ν. Then for any j, the invariant delay sequence of $\mu\tau^j\nu$ with respect to p is $D_1 (D_2)^j D_3$.* □

To mimic the proof of the pointwise case we define intervals in a timed word in which the satisfaction of formulas is invariant. Moreover we require this breaking up of the timed word into intervals to be consistent in some sense over the timed words in a periodic sequence. Hence we introduce the following definitions.

Given a canonical sequence $r_0 r_1 \cdots r_m$ of σ with respect to p, we define the *invariant interval sequence* of σ with respect to p to be $J = J_0 J_1 \cdots J_{2m}$ where $J_{2i} = [r_i, r_i]$ and $J_{2i+1} = (r_i, r_{i+1})$. It follows from Lemma 1 that the satisfiability of an $\mathrm{MTL}(p)$ formula is invariant in σ over each interval J_i.

Given a delay sequence $D = d_1 \cdots d_m$, we can associate an interval sequence $J = J_0 J_1 \cdots J_{2m}$ with it such that $J_0 = [0, 0]$, $J_{2i} = [t, t]$, where $t = \sum_{j=1}^{i} d_j$ and $J_{2i+1} = (t_1, t_2)$, where $t_1 = \sum_{j=1}^{i} d_j$ and $t_2 = \sum_{j=1}^{i+1} d_j$. Note that the interval sequence associated with an invariant delay sequence is the invariant interval sequence.

Lemma 2. *Let $\sigma = \mu\tau\nu$ be a finite timed word such that time$(\tau) = p$, where $p \in \mathbb{Q}_{>0}$. Let $D = D_1D_2D_3$ be the invariant delay sequence of σ with respect to p, where D_1, D_2 and D_3 are the delay sequences corresponding to the subwords μ, τ and ν. Let $\langle\sigma_i\rangle$ be the periodic sequence of finite timed words given by $\sigma_i = \mu\tau^i\nu$. Let $J = J_0J_1\cdots$ be the interval sequence corresponding to the delay sequence $D_1(D_2)^\omega$. Then for all $t \in J_j$ and $\varphi \in \mathrm{MTL}(p)$,*

1. *if $\langle\sigma_i\rangle, t \models^c_{us} \varphi$ then there exists n_j such that for all $n \geq n_j$ and $t' \in J_j$, $\sigma_n, t' \models_c \varphi$ and*
2. *if $\langle\sigma_i\rangle, t \models^c_{un} \varphi$ then there exists n_j such that for all $n \geq n_j$ and $t' \in J_j$, $\sigma_n, t' \not\models_c \varphi$.* □

Proposition 2. *Let $\langle\sigma_i\rangle$ be the periodic sequence given by $\sigma_i = \mu\tau^i\nu$ and time(τ) $= p$, where $p \in \mathbb{Q}_{>0}$. Let $t \in \mathbb{R}_{\geq 0}$ such that $t >$ time(μ). Let $c \in \mathbb{N}$ and let $\varphi \in \mathrm{MTL}$. Then $\langle\sigma_i\rangle, t \models^c_{us} \varphi$ iff $\langle\sigma_i\rangle, t + cp \models^c_{us} \varphi$.* □

Theorem 3. *Let $\langle\sigma_i\rangle$ be a periodic sequence with period p, where $p \in \mathbb{Q}_{>0}$. Let φ be an $\mathrm{MTL}(p)$ formula and let $t \in \mathbb{R}_{\geq 0}$. Then either $\langle\sigma_i\rangle, t \models^c_{us} \varphi$ or $\langle\sigma_i\rangle, t \models^c_{un} \varphi$.*

Proof. The proof follows that for the pointwise case. Since the ultimate satisfiability of a formula is invariant within the intervals of an invariant interval sequence, and there exists an n_j for each interval J_j as given by Lemma 2, we consider each of these intervals as one entity (comparable to a point in the pointwise case). The details of the proof can be found in [10]. □

The above theorem gives us a counter-freeness result for the continuous case. Given a $p \in \mathbb{R}_{>0}$, we call a timed language L, *p-counter-free*, if there do not exist timed words μ, τ and ν such that *time*$(\tau) = p$ and there exist infinitely many i's for which $\mu\tau^i\nu \in L$ and infinitely many of them for which $\mu\tau^i\nu \notin L$. Below is the result for the continuous semantics.

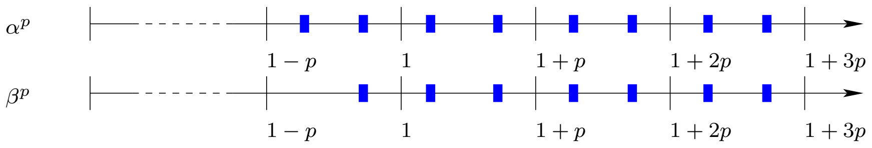
Theorem 4. *Let $p \in \mathbb{Q}_{>0}$. Then every timed language of finite words definable by an $\mathrm{MTL}(p)$ formula in the continuous semantics is p-counter-free.* □

5 Strict Containment of MTL^{pw} in MTL^c

In this section we show the strict containment of MTL^{pw} in MTL^c for finite words. We show that the language L_{2b} described below is not expressible by any MTL^{pw} formula. We will first sketch a proof of the same for infinite words. It is a simplified version of the proof in [3] and the details can be found in [10].

L_{2b} is the timed language over the alphabet $\Sigma = \{b\}$ which consists of timed words in which there are at least two b's in the interval $(0, 1)$. Formally, $L_{2b} = \{(b, t_1)(b, t_2) \cdots \in T\Sigma^\omega \mid \exists t_i, t_j : 0 < t_i < t_j < 1\}$.

Let $p \in \mathbb{Q}_{>0}$, where $p = 1/k$ and $k \in \mathbb{N}$. We give two models, α^p and β^p such that $\alpha^p \in L_{2b}$ but $\beta^p \notin L_{2b}$, and no $\mathrm{MTL}(p)$ formula φ can distinguish between the two models in the pointwise semantics in the sense that $\alpha^p, 0 \models_{pw} \varphi$ iff $\beta^p, 0 \models_{pw} \varphi$. α^p is given by the timed word $(1-3p/4, b)(p/2, b)(p/2, b)(p/2, b)\cdots$ and β^p is given by $(1-p/4, b)(p/2, b)(p/2, b)\cdots$. They are depicted below.



Proposition 3. *Let $i, j \in \mathbb{N}$ and $i, j > 0$. Let $\varphi \in$ MTL. Then $\alpha^p, i \models_{pw} \varphi$ iff $\alpha^p, j \models_{pw} \varphi$. Similarly, $\beta^p, i \models_{pw} \varphi$ iff $\beta^p, j \models_{pw} \varphi$ and $\alpha^p, i \models_{pw} \varphi$ iff $\beta^p, j \models_{pw} \varphi$.*

Proof. All proper suffixes of the two timed words are the same. □

Theorem 5. *For any $\varphi \in$ MTL(p), $\alpha^p, 0 \models_{pw} \varphi$ iff $\beta^p, 0 \models_{pw} \varphi$.* □

Now suppose that there exists an MTL formula φ which in the pointwise semantics defines the language L_{2b}. It belongs to MTL(p) where $p = 1/k$ and k is the least common multiple of the denominators of the interval end-points in φ (recall that the end points are rational). But φ cannot distinguish between α^p and β^p. It is either satisfied by both of them or is not satisfied by any of them. In either case it does not define L_{2b}. Hence L_{2b} is not definable in MTL^{pw}.

But the disjunction of the formulas, $\Diamond_{(0,0.5]} b \wedge \Diamond_{(0.5,1)} b$, $\Diamond_{(0,0.5]}(b \wedge \Diamond_{(0,0.5)} b)$ and $\Diamond_{(0,0.5)}(\Diamond_{[0.5,0.5]} b \wedge \Diamond_{(0,0.5)} b)$, expresses L_{2b} in the continuous semantics. So, MTL^{pw} is strictly contained in MTL^c over infinite words.

Now we extend the above proof for the case of finite words using the notion of ultimate satisfiability. We replace every infinite word by an infinite sequence of finite timed words. Thus we replace α^p by $\langle \sigma_i^p \rangle$ and β^p by $\langle \rho_i^p \rangle$, respectively, which are defined as $\sigma_i^p = \mu_1 \tau^i$ and $\rho_i^p = \mu_2 \tau^i$, where $\mu_1 = (1-3p/4, b)(1-p/4, b)$, $\mu_2 = (1-p/4, b)$ and $\tau = (p/2, b)$. It can be seen that $\langle \sigma_i^p \rangle$ is completely contained in L_{2b} and $\langle \rho_i^p \rangle$ is completely outside L_{2b}. We can now argue that a formula φ in MTL(p) is ultimately satisfied at 0 in $\langle \sigma_i^p \rangle$ iff it is ultimately satisfied at 0 in $\langle \rho_i^p \rangle$. We see that the claims which were true for the infinite case continue to hold for the finite case with the notion of ultimate satisfiability.

Proposition 4. *Let $i, j \in \mathbb{N}$ and $i, j > 0$. Let $\varphi \in$ MTL. Then $\langle \sigma_i^p \rangle, i \models_{us} \varphi$ iff $\langle \sigma_i^p \rangle, j \models_{us} \varphi$. Similarly, $\langle \rho_i^p \rangle, i \models_{us} \varphi$ iff $\langle \rho_i^p \rangle, j \models_{us} \varphi$ and $\langle \sigma_i^p \rangle, i \models_{us} \varphi$ iff $\langle \rho_i^p \rangle, j \models_{us} \varphi$.* □

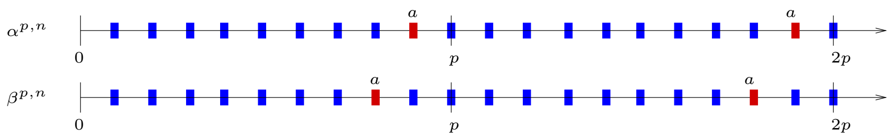
Theorem 6. *Given any $\varphi \in$ MTL(p), $\langle \sigma_i^p \rangle, 0 \models_{us} \varphi$ iff $\langle \rho_i^p \rangle, 0 \models_{us} \varphi$.* □

Suppose there exists a formula φ which defines L_{2b} in the pointwise semantics. Then $\varphi \in$ MTL(p) for some p. Since φ defines L_{2b} it is satisfied by all timed words in $\langle \sigma_i^p \rangle$. So φ is ultimately satisfied at 0 in $\langle \sigma_i^p \rangle$ and hence is ultimately satisfied at 0 in $\langle \rho_i^p \rangle$. This is a contradiction since none of the timed words in $\langle \rho_i^p \rangle$ are in L_{2b}. Therefore no MTL formula defines L_{2b} in the pointwise semantics.

6 Strict Containment of MTL^c in MTL$_S^c$

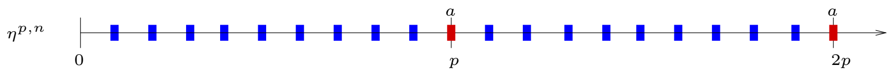
In this section we show that MTL^c is strictly contained in MTL$_S^c$ for finite timed words by showing that the language L_{last_a} is not expressible by any MTL^c formula but is expressible by an MTL$_S^c$ formula. L_{last_a} consists of timed words over $\{a,b\}$ such that the last symbol in the interval $(0,1)$ is an a and there is an action at time 1. We will sketch a proof of the above claim for the case of infinite words which essentially follows the one given in [3] and then show how it can be extended for finite words.

Let $p \in \mathbb{Q}_{>0}$, where $p = 1/q$ and $q \in \mathbb{N}$, and let $n \in \mathbb{N}$. We give two infinite timed words $\alpha^{p,n}$ and $\beta^{p,n}$ such that $\alpha^{p,n} \in L_{last\_a}$ and $\beta^{p,n} \notin L_{last\_a}$. Let $d = p/(n+4)$. Then,



$\alpha^{p,n} = (c_1, d)(c_2, 2d)\cdots$ where $c_k = a$ if $k \mod (n{+}4) = n{+}3$, $c_k = b$ otherwise.
$\beta^{p,n} = (c_1, d)(c_2, 2d)\cdots$ where $c_k = a$ if $k \mod (n{+}4) = n{+}2$, $c_k = b$ otherwise.

Let us consider the following infinite model $\eta^{p,n}$ given by the timed word $(c_1, d)(c_2, 2d)\cdots$, where $c_k = a$ if $k \mod (n+4) = 0$, $c_k = b$ otherwise.



We denote by MTL(p,k) the formulas in MTL(p) with an U nesting depth of k.

Lemma 3. *Let $k \in \mathbb{N}$ and $0 \le k \le n$. Let $\varphi \in$ MTL(p,k). Let $i,j \in \{1,\cdots,n+3-k\}$ and let $\alpha \ge 0$. Then $\eta^{p,n},(\alpha(n{+}4){+}i)d \models_c \varphi$ iff $\eta^{p,n},(\alpha(n{+}4){+}j)d \models_c \varphi$ and for all $t_1,t_2 \in (0,d)$, $\eta^{p,n},(\alpha(n+4)+i)d-t_1 \models_c \varphi$ iff $\eta^{p,n},(\alpha(n+4)+j)d-t_2 \models_c \varphi$.* □

Corollary 1. *Let $\varphi \in$ MTL(p,n) and let $\alpha \ge 0$. Then $\eta^{p,n},(\alpha(n{+}4){+}1)d \models_c \varphi$ iff $\eta^{p,n},(\alpha(n+4)+2)d \models_c \varphi$ iff $\eta^{p,n},(\alpha(n+4)+3)d \models_c \varphi$ and for all $t_1,t_2,t_3 \in (0,d)$, $\eta^{p,n},(\alpha(n+4)+1)d-t_1 \models_c \varphi$ iff $\eta^{p,n},(\alpha(n+4)+2)d-t_2 \models_c \varphi$ iff $\eta^{p,n},(\alpha(n+4)+3)d-t_3 \models_c \varphi$.* □

Theorem 7. *Let $\varphi \in$ MTL(p,n). Then $\alpha^{p,n},0 \models_c \varphi$ iff $\beta^{p,n},0 \models_c \varphi$.* □

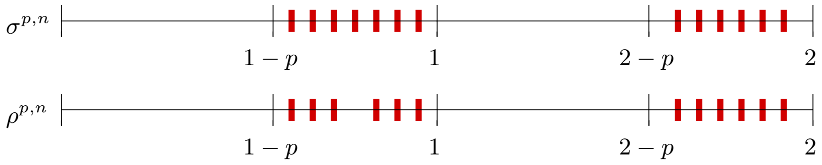
We now extend the results for the case of finite words. As before we replace the infinite words $\alpha^{p,n}$ and $\beta^{p,n}$ by the sequences $\langle\sigma_i^{p,n}\rangle$ and $\langle\rho_i^{p,n}\rangle$, respectively, which are given by $\sigma_i^{p,n} = \mu_1\tau^i$, and $\rho_i^{p,n} = \mu_2\tau^i$, where $\mu_1 = (b,d)(b,2d)\cdots(b,(n+2)d)(a,(n{+}3)d)$, $\mu_2 = (b,d)(b,2d)\cdots(b,(n{+}1)d)(a,(n{+}2)d)$ and $\tau = (b,d)(b,2d)\cdots(b,(n{+}3)d)(a,(n{+}4)d)$. Further, we replace $\eta^{p,n}$ by $\langle{\sigma'}_i^{p,n}\rangle$ where ${\sigma'}_i^{p,n} = \tau^i$. We can now mimic the proof sketched above by replacing satisfiability by ultimate satisfiability of the continuous semantics.

7 Continuous Semantics is Strictly More Expressive

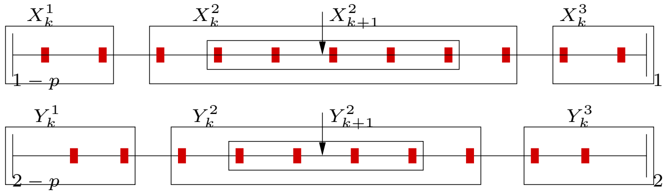
In this section we show that the language L_{2ins} is not expressible by $\mathrm{MTL}^{pw}_{S_I}$ but is expressible by MTL^c. This leads to the strict containment of the pointwise versions of the logics in their corresponding continuous versions, since L_{2ins} is not expressible by $\mathrm{MTL}^{pw}_{S_I}$, and hence is not expressible by MTL^{pw}_S and MTL^{pw}, but is expressible by MTL^c, and hence by MTL^c_S and $\mathrm{MTL}^c_{S_I}$.

We will first show the result for finite words and then show how it can be extended for infinite words. L_{2ins} is the timed language over $\Sigma = \{a, b\}$ such that every timed word in the language contains two consecutive a's such that there exist two time points between their times of occurrences, at distance one in the future from each of which there is an a. Formally, $L_{2ins} = \{\sigma \in T\Sigma^* \mid \sigma = (a_1, t_1) \cdots (a_n, t_n), \exists i \in \mathbb{N} : a_i = a_{i+1} = a, \exists t_j, t_k \in (t_i + 1, t_{i+1} + 1) : j \neq k, a_j = a_k = a\}$.

Let $p \in \mathbb{Q}_{>0}$, where $p = 1/k$ and $k \in \mathbb{N}$, and let $n \in \mathbb{N}$. Let $d = p/(2n + 3)$. We give the two models $\sigma^{p,n}$ and $\rho^{p,n}$ which are as defined follows. $\sigma^{p,n}$ is given by $(a, 1-p+d/2)(a, 1-p+3d/2) \cdots (a, 1-p/2-d)(a, 1-p/2)(a, 1-p/2+d) \cdots (a, 1-d/2)(a, 2-p+d)(a, 2-p+2d) \cdots (a, 2-d)$ and $\rho^{p,n}$ is given by $(a, 1-p+d/2)(a, 1-p+3d/2) \cdots (a, 1-p/2-d)(a, 1-p/2+d) \cdots (a, 1-d/2)(a, 2-p+d)(a, 2-p+2d) \cdots (a, 2-d)$. They are depicted below.



It is easy to see that $\sigma^{p,n} \notin L_{2ins}$ and $\rho^{p,n} \in L_{2ins}$. We use the following lemmas to show that no MTL_{S_I} formula can define L_{2ins} in the pointwise semantics. Let $X^2_k = \{k+1, \cdots, 2n+3-k\}$ and $Y^2_k = \{2n+3+(k+1), \cdots, 2n+3+(2n+2-k)\}$.



Lemma 4. *Let $k \in \mathbb{N}$ and $0 \leq k \leq n$. Let $\varphi \in \mathrm{MTL}_{S_I}(p, k)$. Then for all $i, j \in X^2_k$, $\sigma^{p,n}, i \models_{pw} \varphi$ iff $\sigma^{p,n}, j \models_{pw} \varphi$ and for all $i, j \in Y^2_k$, $\sigma^{p,n}, i \models_{pw} \varphi$ iff $\sigma^{p,n}, j \models_{pw} \varphi$.* □

Given an $n \in \mathbb{N}$, we define a partial function $h_n : \mathbb{N} \to \mathbb{N}$ which is defined for all $i \in \mathbb{N}$ except for $n+2$. $h_n(i) = i$ if $i < n+2$ and $h_n(i) = i-1$ if $i > n+2$. $h_n(i)$ is the position in $\rho^{p,n}$ corresponding to the position i in $\sigma^{p,n}$ in the sense that the time of the $h_n(i)$-th action in $\rho^{p,n}$ is the same as that of the i-th action in $\sigma^{p,n}$ (hence it is not defined for $n+2$).

Lemma 5. *Let $k \in \mathbb{N}$, $0 \le k \le n$ and let $\varphi \in \mathrm{MTL}_{S_I}(p,k)$. For all $i, j \in X_k^2 - \{n+2\}$, $\rho^{p,n}, h_n(i) \models_{pw} \varphi$ iff $\rho^{p,n}, h_n(j) \models_{pw} \varphi$ and for all $i, j \in Y_k^2$, $\rho^{p,n}, h_n(i) \models_{pw} \varphi$ iff $\rho^{p,n}, h_n(j) \models_{pw} \varphi$.* □

Corollary 2. *Let $\varphi \in \mathrm{MTL}_{S_I}(p,n)$. Then $\sigma^{p,n}, n+1 \models_{pw} \varphi$ iff $\sigma^{p,n}, n+2 \models_{pw} \varphi$ iff $\sigma^{p,n}, n+3 \models_{pw} \varphi$ and $\rho^{p,n}, h_n(n+1) \models_{pw} \varphi$ iff $\rho^{p,n}, h_n(n+3) \models_{pw} \varphi$.* □

Theorem 8. *For any $\varphi \in \mathrm{MTL}_{S_I}(p,n)$ and $i \in \mathbb{N}$, where $i \neq n+2$, $\sigma^{p,n}, i \models_{pw} \varphi$ iff $\rho^{p,n}, h_n(i) \models_{pw} \varphi$.* □

Corollary 3. *For any $\varphi \in \mathrm{MTL}_{S_I}(p,n)$, $\sigma^{p,n}, 0 \models_{pw} \varphi$ iff $\rho^{p,n}, 0 \models_{pw} \varphi$.* □

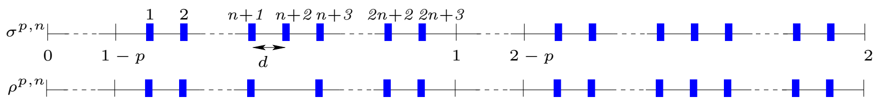
It can now be argued that no MTL_{S_I} formula in the pointwise semantics can define L_{2ins}. But the following MTL formula in the continuous semantics defines L_{2ins}. $\Diamond(a \wedge \neg\varphi_{act} U(\neg\varphi_{act} \wedge \Diamond_{[1,1]} a \wedge \neg\varphi_{act} U(\neg\varphi_{act} \wedge \Diamond_{[1,1]} a \wedge \neg\varphi_{act} U a)))$.

The above result can be extended for infinite timed words by replacing the finite models above, by infinite models which are similar to their counterparts in the interval $[0,2]$ but contain a b at every integer time greater than 2. Then the proof for infinite models is very similar to that of the finite models.

8 Strict Containment of MTL_S^{pw} in $\mathrm{MTL}_{S_I}^{pw}$

In this section we show the strict containment of MTL_S^{pw} in $\mathrm{MTL}_{S_I}^{pw}$ by showing that the language L_{ni} is not expressible by MTL_S^{pw}, but is expressible by $\mathrm{MTL}_{S_I}^{pw}$.

The timed language L_{em} (for "exact match") consists of timed words in which for every b in the interval $(0,1)$, there is a b in the future which is at time distance 1 from it, and for every b in the interval $(1,2)$, there is a b in the past which is at time distance 1 from it. The structure of the proof is very similar to that of the previous section. We give below a description of the models.



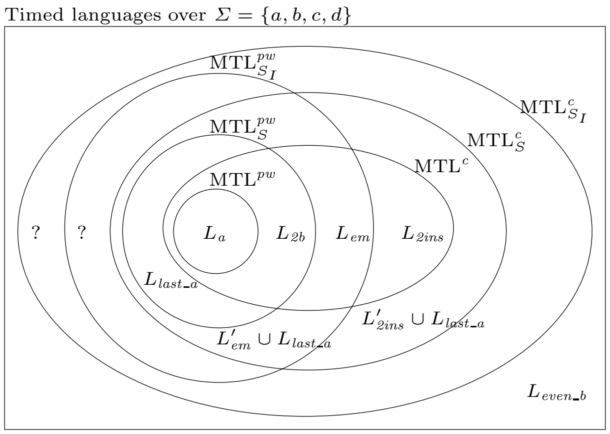
Let $x = 1-p$, $y = 2-p$ and $d = p/(2n+4)$. Then $\sigma^{p,n}$ is given by $(b, x+d)(b, x+2d)\cdots(b, x+(n+1)d)(b, x+(n+2)d)(b, x+(n+3)d)\cdots(b, x+(2n+2)d)(b, x+(2n+3)d)(b, y+d)(b, y+2d)\cdots(b, y+(n+1)d)(b, y+(n+2)d)(b, y+(n+3)d)\cdots(b, y+(2n+2)d)(b, y+(2n+3)d)$.

And $\rho^{p,n}$ is given by $(b, x+d)(b, x+2d)\cdots(b, x+(n+1)d)(b, x+(n+3)d)\cdots(b, x+(2n+2)d)(b, x+(2n+3)d)(b, y+d)(b, y+2d)\cdots(b, y+(n+1)d)(b, y+(n+2)d)(b, y+(n+3)d)\cdots(b, y+(2n+2)d)(b, y+(2n+3)d)$.

Taking $X_k^2 = \{k+1, \cdots, 2n+3-k\}$ and $Y_k^2 = \{(2n+3)+k+1, \cdots, (2n+3)+2n+3-k\}$, we can follow the proof of the inexpressibility of L_{2ins}. The proof can be similarly extended for the case of infinite words.

9 Expressiveness of MTL with Past Operators

In the diagram below, we summarize the relative expressiveness results. The timed language L_a consists of timed words containing an a. L'_{em} and L'_{2ins} are obtained from L_{em} and L_{2ins} respectively by replacing the occurrences of a's and b's in the timed words by c's and d's, respectively.



References

1. Rajeev Alur, Tomás Feder, and Thomas A. Henzinger. The Benefits of Relaxing Punctuality. *Journal of the ACM*, 43(1):116–146, 1996.
2. Rajeev Alur and Thomas A. Henzinger. Real-time Logics: Complexity and Expressiveness. In *LICS*, pages 390–401. IEEE Computer Society, 1990.
3. Patricia Bouyer, Fabrice Chevalier, and Nicolas Markey. On the Expressiveness of TPTL and MTL. In *FSTTCS*, volume 3821 of *Lecture Notes in Computer Science*, pages 432–443. Springer, 2005.
4. Deepak D'Souza and Pavithra Prabhakar. On the expressiveness of MTL in the pointwise and continuous semantics. *To appear in Formal Methods Letters, Software Tools for Technology Transfer.*
5. Johan Anthony Willem Kamp. *Tense Logic and the Theory of Linear Order.* PhD thesis, University of California, Los Angeles, California, 1968.
6. Ron Koymans. Specifying Real-Time Properties with Metric Temporal Logic. *Real-Time Systems*, 2(4):255–299, 1990.
7. Robert McNaughton and Seymour A. Papert. *Counter-Free Automata (M.I.T. research monograph no. 65).* The MIT Press, 1971.
8. Joël Ouaknine and James Worrell. On the Decidability of Metric Temporal Logic. In *LICS*, pages 188–197. IEEE Computer Society, 2005.
9. Joël Ouaknine and James Worrell. On Metric Temporal Logic and Faulty Turing Machines. In *FoSSaCS*, volume 3921 of *Lecture Notes in Computer Science*, pages 217–230. Springer, 2006.
10. Pavithra Prabhakar and Deepak D'Souza. On the expressiveness of MTL with past operators. Technical Report IISc-CSA-TR-2006-5, Indian Institute of Science, Bangalore 560012, India, May 2006. URL: `http://archive.csa.iisc.ernet.in/TR/2006/5/`.

Simulator for Real-Time Abstract State Machines

Pavel Vasilyev[1,2,⋆]

[1] Laboratory of Algorithmics, Complexity and Logic, Department of Informatics, University Paris-12, France
[2] Computer Science Department, University of Saint Petersburg, Russia

Abstract. We describe a concept and design of a simulator of Real-Time Abstract State Machines. Time can be continuous or discrete. Time constraints are defined by linear inequalities. Two semantics are considered: with and without non-deterministic bounded delays between actions. The simulator is easily configurable. Simulation tasks can be generated according to descriptions in a special language. The simulator will be used for on-the-fly verification of formulas in an expressible timed predicate logic. Several features that facilitate the simulation are described: external functions definition, delays settings, constraints specification, and others.

1 Introduction

Usually the process of software development consists of several main steps: analysis, design, specification, implementation, and testing. The steps can be iterated several times, and the last two ones are repeated very often. All the phases of development are accompanied by this or that validation. We are interested in the validation by simulation of a program specification with respect to the requirements. The problem we consider has the following features :

- we consider real-time reactive systems with continuous or discrete time, and time constraints expressed by linear inequalities,
- programs are specified as Abstract State Machines (ASM) [1],
- requirements are expressed in a First Order Timed Logic (FOTL) [2,3].

The specification languages we consider are very powerful. Even rather simple, "basic" ASMs [4] are sufficient to represent any algorithmic state machine with exact isomorphic modeling of runs. This formalism bridges human understanding, programming and logic. The ASM method has a number of successful practical applications, e.g., SDL semantics, UPnP protocol specification, semantics of VHDL, C, C++, Java, Prolog (see [5,6]).

To express properties of real-time ASMs we use FOTL. This logic is clearly undecidable. There exist practical decidable classes [2,3,7], however the decidability algorithm is not so simple to apply. We know from the practice that most

⋆ Partially supported by ECO-NET project No 08112WJ.

errors in software can be revealed on rather simple inputs ; for reactive systems this means that finite models of small complexity are usually sufficient to find very serious errors.

Thus, we design our simulator as some kind of partial, bounded, on-the-fly model-checker. It is presumed to be light, portable and easily configurable. The simulator provides a language to describe a timed semantics and to generate inputs. We consider two semantics, both with instantaneous actions, one without delays between actions, another one, more realistic, with bounded non deterministic delays between actions (by action here we mean an update of the current state, we make it more precise below). The simulator checks the existence of a run for a given input, outputs details of the run that can be specified in a special language, and checks the requirements formula for this run.

There are several implementations of ASM interpreter or compiler. One of them is the Microsoft AsmL [8]. AsmL is integrated with Microsoft's software development, documentation, and runtime environments. It supports specification and rapid prototyping of object oriented and component oriented software. Another language is Distributed ASML [9]. Distributed ASML is also aimed at development of object-oriented and component-oriented software but has additional features for specification of distributed and multi-agent systems. There is also a number of other interpreters such as Michigan interpreter, Gem-Mex, Prolog-based interpreter.

These systems do not deal with real-time ASMs or predicate logic requirements. We simulate real-time ASMs and we wish to have an easily configurable simulator. To do that we need some additional features such as explicit time manipulation, built-in features for time propagation and delays maintenance, and more powerful capabilities for constraints and properties defining. In this paper we do not describe the part that concerns FOTL properties checking, it will be done elsewhere.

2 Timed ASM

In this paper by an ASM we mean *timed basic Gurevich abstract state machine.* A timed ASM is a tuple $(V, IS, Prog)$, where V is a vocabulary, IS is a description of the initial state, and $Prog$ is a program. The vocabulary consists of a set of sorts, a set of function symbols, and a set of predicate symbols. The following pre-interpreted sorts are included: $\mathcal{R}$ is the set of reals; $\mathcal{Z}$ is the set of integers; $\mathcal{N}$ is the set of natural numbers; *Bool* is the set of Boolean values: *true* and *false*; $\mathcal{T} = \mathcal{R}_+$ special time sort; $Undef = \{undef\}$ is a special sort used to represent the *undefined* values.

All functions of an ASM are divided into two categories: *internal* and *external functions.* Internal functions can be modified by the ASM. External functions cannot be changed by the ASM. On the other hand, the functions can also be divided into *static* and *dynamic functions.* Dynamic external functions represent the input of the ASM. A static function has a constant interpretation during any run of the ASM. Among the static pre-interpreted functions of the vocabulary are arithmetical operations, relations, and boolean operations. The equality relation "=" is assumed to be defined for all types.

The timed ASMs use a special time sort $\mathcal{T}$ and a nullary function $CT : \ \rightarrow \mathcal{T}$ which returns the current "physical" time. Only addition, subtraction, multiplication and division by a rational constant, standard equality and inequality relations are supported. Following is an example of a guarded update:

```
if (CT = 5) or (CT >= 11) then
  x := CT + 2.3;
```

Both discrete time (natural numbers including zero) and continuous time (we use only finite sets of intervals and points) are supported by the simulator. A run of an ASM is a mapping from time to states. Each state is an interpretation of the vocabulary of the ASM. Thus, from the "run viewpoint" a dynamic function is a function of time.

In this work we admit only piecewise linear inputs defined on sets of left-closed right-open intervals $[t_k, t_{k+1})$, where t_k are time points, k is natural. The internal functions will be also piecewise linear defined on sets of left-open right-closed $(t_k, t_{k+1}]$ (it is implied by the structure of ASM).

The program of the timed ASM is defined in a usual way as a sequence of instructions of several types. Here we show the main constructions though there are also class and method definitions, **while-do** and **repeat-until** loops, **forall** and **choose** statements, etc.:

- A single *update rule* in the form of an assignment $A = \{f(x_1, \ldots, x_k) := \theta\}$;
- A *parallel block* of update rules $[A_1; \ldots; A_m]$ which are executed simultaneously (this block is called an *update block*);
- A *sequential block* of update rules $\{A_1; \ldots; A_m\}$ which are executed in the order they are written;
- A *guarded rule*, where $Guard_i, i \in 1, \ldots, n$ are guard conditions and $A_i, i \in 1, \ldots, n+1$ are statements:
  if $Guard_1$ **then** A_1 **elseif** $Guard_2$ **then** A_2 ... **else** A_{n+1}

2.1 Lamport's Bakery Algorithm

Here we propose an example of asynchronous distributed algorithm, based on N processes. This algorithm was originally presented by L. Lamport in [10]. A modified version of the algorithm was found in the article [11]. This algorithm shows a way of mutual-exclusive work with a resource. Here the **Bakery** class denotes a process which works with some resource after the **CS** (Critical section) variable turns to **true**. Here is the text in the syntax of Timed ASML:

```
class Bakery {
  var number: Integer;
  var CS: Boolean;
  var x: Bakery -> Integer;
  run() {
    number := 0;
    CS := false;
    forall q: Bakery do x(q) := 0;
    while true do {
// Doorway:
      number := 1;  x(me) := 1;
      forall q: Bakery where q != me do x(q) := q.number;
      x(me) := 1 + max(x);
      number := x(me);
// Bakery:
      forall q: Bakery where q != me do
        repeat x(q) := q.number;
        until (x(q) = 0 or (x(me), me) < (x(q), q));
      CS := true;
// Critical section:
      CS := false;
      number := 0;
    }
  }
}
```

3 External Functions Definition

An external function definition looks as follows: $f : \mathcal{X} \rightarrow \mathcal{Y}$, the corresponding timed version of the function is the following: $f^{\circ} : \mathcal{T} \times \mathcal{X} \rightarrow \mathcal{Y}$, where f is the name of the function, $\mathcal{X}$ is an abstract sort or a direct product of two abstract sorts, $\mathcal{T}$ is the pre-interpreted time sort, $\mathcal{Y}$ is an abstract sort or pre-interpreted sort $\mathcal{R}$. The timed version of a function f° can not be used explicitly in an ASM specification, it is used in requirements. A function can change its interpretation during the run of the ASM, therefore the time dimension $\mathcal{T}$ allows us to use the time variable as an independent variable. Definite time variable values split the whole run of an ASM into a sequence of "slices" in which the interpretations are constant.

The sort $\mathcal{X}$ can be enumerated as a sequence of natural numbers. Thus, we can define a function as follows:

$$f(i) := (t_1^i, f_{12}^i; t_2^i, f_{23}^i; \ldots; t_k^i, f_{kk+1}^i; \ldots)$$

where $i \in 1, \ldots, n$, n is the number of intervals or the cardinality of the sort $\mathcal{X}$, $t_1, t_2, \ldots, t_k, \ldots$ are start points of intervals, $f_{12}^i, f_{23}^i, \ldots, f_{kk+1}^i, \ldots$ are function values defined on the time left-closed right-open intervals. If $\mathcal{X}$ is a product of two abstract sorts, $\mathcal{X} = \mathcal{X}' \times \mathcal{X}''$ then the definition will look as follows:

$$f(i,j) := (t_1^{i,j}, f_{12}^{i,j}; t_2^{i,j}, f_{23}^{i,j}; \ldots; t_k^{i,j}, f_{kk+1}^{i,j}; \ldots)$$

where $i \in 1, \ldots, n, j \in 1, \ldots, m$, n is the number of intervals or the cardinality of the sort $\mathcal{X}'$, m is the cardinality of the sort $\mathcal{X}''$, and all other designations are analogous to the ones in the previous sequence.

Each function can be also defined by an expression. The following additional pre-interpreted variables can be used in the definitions: t_a — absolute time, t_r — relative time, varying in the time interval $[0, t_{k+1} - t_k)$, for each k. Thus, for $[t_k, t_{k+1})$ the following holds: $t_a = t_k + t_r$.

4 Delay Settings

All updates in ASM [1] are instantaneous. In reality, it may take some time to perform an update. We model this by non-deterministic bounded delays between actions that remain instantaneous. In the current implementation of the simulator the delay is calculated deterministically. The value of a delay depends on the complexity and quantity of instructions used in the specification. Non-deterministic bounded delays will appear in the next version of the simulator. Some ideas of managing the process of time propagation already appeared, for example in [12,11]. We propose several ways of dealing with delays depending on the purposes of the user. We define a function of time delay δ on the set of all statements and expressions which will be denoted as $\mathcal{S}$. The function of time delay can be specified as $\delta : \mathcal{S} \to \mathcal{T}$. For each operation the value of time delay function can be set individually and it will be used in the process of simulation.

There are two different types of calculations: sequential composition and parallel composition. For the sequential composition $f \circ g$ the delay is calculated as a sum of delays of each function, i.e. $\delta(f \circ g) = \delta(f) + \delta(g)$. For the parallel composition $||f_i$ the delay is the maximum of the delays of all functions, i.e. $\delta(||f_i) = \max_i \delta(f_i)$.

This function δ can be used at different levels of abstraction. For example, it can be used at the level where we have only two different time delays for slow and fast operations making difference between operations with internal and *shared variables*. The notion of shared variables is used to mark the variables that can be accessed from several processes. This kind of variables is one of the methods for implementing inter-process communication. Operations with shared variables are usually slower than operations with internal variables of a process, as they imply several inter-process operations. In our Bakery algorithm the field **number** of the **Bakery** class is a shared variable. Thus, we can specify delays in various ways, e.g.:

- Two dedicated delays δ_{ext} (operations with external functions) and δ_{int} (operations which deal only with internal functions), $\delta_{ext} > 0$, $\delta_{int} > 0$;
- The same as the previous one but all internal operations are instantaneous: $\delta_{ext} > 0$, $\delta_{int} = 0$;
- No delay, all operations are instantaneous: $\delta_{ext} = \delta_{int} = 0$. In this degenerated case we have a standard abstract state machine;
- A generalized case: there are no predefined delays for all operations. We have a set of delays each of them corresponds to one function or operation. All of them have to be defined manually.

Examples (in the configuration file we write d(<instruction>)=<delay>):

```
d(+) = 1; d(*) = 2; dext = 3; dint = 1;
```

5 Non-determinism Resolving

A typical example of nondeterministic choice is the **choose** statement when one of the possible values has to be chosen. We consider this situation as a tuple of all possible choices with several ways of element selection. The method should be provided in the configuration of the simulator. It is set by one assignment in the configuration file (the abbreviation **ndr** stands for non-determinisms resolution). Here some methods are shown:

– The first (last) element of the tuple: `ndr = first`,
– The minimal (maximal) element of the tuple, if some order is defined for the elements: `ndr = min`,
– The element defined by its index — a sequence of natural numbers with a predefined step: `ndr = seq start <x> step <y>`,
– The element defined by its index; the index is provided by a random numbers generator: `ndr = random.`

For example, consider a tuple {5, 1, 3, 2, 9, 6, 4}. The first and the last elements will be 5 and 4. The minimal and the maximal elements will be 1 and 9. If we define a sequence of natural numbers starting from 1 with the step of 3 we will get a resulting sequence as follows: {5, 2, 4, 9, 3, 6, 1}.

6 Properties

To express requirements for ASMs one needs a powerful logic. In this work we consider a First Order Timed Logic (FOTL) [2,3] for representing the requirements. For the Lamport's Bakery the following properties are required:
Safety: $\forall pq\ \forall t(p \neq q \rightarrow \neg(CS_p^\circ(t) \wedge CS_q^\circ(t))$
Liveness:

$$\exists c_1 c_2\ \forall p\ \forall t(number_p^\circ(t) > 0 \rightarrow \exists t'(t < t' < t + c_1 \cdot \delta_{int} + c_2 \cdot \delta_{ext} \wedge CS_p^\circ(t')))$$

where $\delta_{int} > 0, \delta_{ext} > 0$ (in the degenerated case we should use another formula).

There are already several existing solutions in which some properties and constraints of a specification can be explicitly formulated. The Abstract State Machine Language proposed by Microsoft supports the Object-Oriented approach. The following properties are supported in MS AsmL2: require — a precondition, ensure — a postcondition, constraint — a general assertion. Preconditions and postconditions can be used in different statement blocks with AsmL operations. The reserved word constraint can be also used in data type declarations. Here are two examples of constraints:

```
structure Rational
  numerator as Integer
  denom as Integer
  constraint NonZeroDivisor: denom <> 0
```

```
var Counter = 1 Increment()
  require Counter >= 0
  ensure resulting Counter = Counter + 1
  Counter := ((((Counter + 1) * 2) - 2) / 2) + 1
```

In Distributed AsmL [13] the following integrity constraints are supported: require — a precondition, ensure — a postcondition, invariant — a formula which holds on the level where it is defined. Invariants can be defined at the level of namespaces and classes. The preconditions and postconditions can be used only for methods. The constructions are similar to Microsoft version of AsmL but there are several differences. For example, in MS AsmL2 one can define constraints regarding the data types, but defining a constraint for the whole namespace is impossible.

6.1 First Order Timed Logic

The main idea of FOTL is to choose a decidable theory to work with arithmetics or other mathematical functions, and then to extend it by abstract functions of time that are needed to specify the problems taken under consideration. In some way, the theory must be minimal to be sufficient for a good expressivity. For the purposes of the present work we take the theory of mixed real/integer addition with rational constants and unary multiplications by rational numbers. This theory is known to be decidable [14]. Though we can consider either discrete time as non negative integers or continuous time as non negative reals, we take the case of continuous time.

Syntax of FOTL. The vocabulary W of a FOTL consists of a finite set of sorts, a finite set of function symbols and a finite set of predicate symbols. To each sort a set of variables is attributed. Some sorts are pre-interpreted and have a fixed interpretation. Such sorts are real numbers $\mathcal{R}$ and the time sort $\mathcal{T}$. Therefore, we have the same sorts as in the description of the Timed ASM including the time sort $\mathcal{T}$. Other sorts are finite and if a finite sort has a fixed cardinality it can be considered as a pre-interpreted sort. Some functions and predicates are also pre-interpreted. These are addition, subtraction, and scalar multiplication of reals by rational numbers. The pre-interpreted predicates are $=, \leq, <$ over real numbers. The vocabulary of FOTL also contains equality for all types of objects and a CT° function to define the time. Any abstract function is of the type $\mathcal{T} \times \mathcal{X} \to \mathcal{Y}$, where $\mathcal{X}$ is a finite sort or a direct product of two finite sorts, and $\mathcal{Y}$ is a finite sort or the pre-interpreted sort $\mathcal{R}$. An abstract predicate is of the following type $\mathcal{T} \times \mathcal{X} \to Bool$ with the same type of sort $\mathcal{X}$.

Semantics of FOTL. For FOTL the notions of interpretation, model, satisfiability and validity are treated as in first order predicate logic except the pre-interpreted symbols of the vocabulary. Therefore, $\mathcal{M} \models F, \mathcal{M} \not\models F$, and $\models F$ where $\mathcal{M}$ is an interpretation and F is a formula, denote correspondingly that $\mathcal{M}$ is a model of F, $\mathcal{M}$ is a counter-model of F, and F is valid.

7 Basic Abstract State Machines' Syntax

In this section most of the constructs of the Timed ASM language are described. The Timed ASML syntax is similar to MS AsmL or Distributed ASML as we tried to preserve the existing syntactical features. The description is provided with several examples of alternative syntaxes.

7.1 Update Instructions

Update is one of the main instructions of the ASM languages as it changes the state of an ASM. As usually a basic update is implemented as an assignment. Here is an example of a basic update:

```
x := a;
```

An important case of an update is a guarded update. This kind of updates is executed when the guarding condition turns to true. This update is implemented as a $if - then - else$ construction. The following is an example of a guarded update:

```
if Condition_1 then Statement_1
elseif Condition_2 then Statement_2
else Statement_3
```

In the last example $Statement_1$ and $Statement_2$ can be a simple update, a sequential or a parallel block of updates.

Another way to describe an update rule is a named update rule, it can be a procedure or a method of some class. Such an update can be called from some point of the specification. Here is an example of a named update rule :

```
method_name (par1 : type1, par2 : type2) : return_type
```

7.2 Sequential Block of Statements

A sequential block of statements is simply a sequence of statements which are executed one after another. Several ways of defining sequential blocks have been presented in different works. They propose to specify a sequential composition of statements P and Q like:

```
P seq Q

step
  P
step
  Q
```

We have chosen a way more familiar to programmers:

```
{
  P
  Q
}
```

7.3 Parallel Block of Statements

A block of parallel is a set of statements for which the order of statements is not important. All statements in a parallel block are executed simultaneously. Among the existing works there are also several ways of parallel block definition, for example:

```
do in-parallel
  P1
  :
  Pn
enddo

par
  P1
  :
  Pn
endpar
```

We have chosen a more short variant:

```
[ P1 ... Pn ]
```

7.4 Types

For type defining the following constructs can be used:

```
type new_type1 = other_type;
type new_type2 = {val1, val2, ... valN};
type new_type3 = type1, type2, ... typeN -> result_type;
```

7.5 Functions (in Terms of ASM)

All functions both of zero and non-zero arity are defined with the help of *var* keyword.

```
var function_name1 : defined_function_type;
var function_name2 : type1, type2, ... typeN -> result_type;
```

7.6 Variables and Constants

To declare a variable the *var* keyword should be also used as a function of zero arity:

```
var variable_name [= initial_value] : variable_type;
```

Constant definition should be made in a similar way:

```
const const_name = const_value : const_type;
```

7.7 Sets and Sequences

Here is an example of constructing several sets:

```
x = {2..5}                       // same as {3, 2, 5, 4}
y = {i | i in x where i < 4}     // same as {2, 3}
z = {3, 2}                       // same as y
```

The following is an example of constructing sequences:

```
x = [2..5]                       // same as [2, 3, 4, 5]
y = [i | i in x where i < 4]     // same as [2, 3]
z = [2, 3]                       // same as y
w = [2, 2, 3]                    // not the same as z
```

7.8 Iterators and Selection

Here are some examples of iterators used to process all elements of a set:

```
forall x with Y do P

do forall x:Y
  P
enddo
```

And the following construct is used to choose some elements of a set:

```
choose x with Y do P

choose x:Y
  P
endchoose
```

We will use a more flexible of the existing notations:

```
foreach u in U where b(u) do S
choose u in U where b(u) do S
```

7.9 Predicate Formulas

To represent a predicate formula we will use a common syntax that is used in most of the ASM based languages:

```
forall x in Y holds F(x)
exists x in Y where F(x)
```

Both keywords "in" and ":" can be used to specify the membership of a variable x to the set Y. The predicate formula itself is specified by F(x) — a FOTL expression of Boolean type.

8 Timed Abstract State Machine Semantics

8.1 Simulation Process

In brief the process of simulating a system defined by an ASML specification in our approach consists of the following steps:

1. Calculation of the next time point in which at least one guard is true;
2. State update, which is represented by one or more statement blocks;
3. Evaluation of constraints before and after the state update.

As it was defined earlier, the value of the time function can only be read and used in further calculations. The formulas for representing guards can include arithmetical operations ($+$, $-$, $*$, $/$, $\%$), equality, inequality, logical relations (not, and, or, xor), and time variable CT. The delays are also taken into consideration. So, if we have a block with a sequence of guarded updates we must even skip some updates in case of the delay of the previous update overlaps the time point where the guard of the next update becomes true.

In definitions of constraints we can use all features of FOTL language. We consider only time quantified formulas, so we can provide verification of the properties on-the-fly. Additionally, we can add some useful operators which can help the user to formulate some specific properties. These can be temporal operators regarding the future as “soon”, “always”, “never”, or specific “next update”, “in the nearest k updates”, as we have a sequence of time intervals for model interpretation.

Similar to “C” programming language and Java the Main() method is used as the top-level entry point of a specification. From the beginning of this method the simulation of the top-level abstract state machine starts. At first, the main ASM on the top level of the specification is executed. Further the process of simulation runs as follows.

8.2 Sequential Execution

In the sequential mode of execution the next operation is taken, all required calculations are made and the state of the machine is changed. Then the next construct or instruction is taken and so on.

If the user has specified the time delays for Timed ASML operations then the time value is changed according to the given configuration. The current time value is simply incremented by the value of current instruction time delay.

8.3 Block of Parallel Instructions

All instructions in a parallel block are considered to be executed in parallel. We consider it as block of *sub-machines* running in the same way as the top-level ASM but in its own space of states. Each sub-machine does not interact or affect other sub-machines. If several sub-machines have been executed from a parallel block they have the same initial state but their end states after the execution, in general, are different. When all of them finish their execution the total state change of the upper level sub-machine has to be calculated. Let each sub-machine be designated as SM_i, $i = 1, 2, \ldots, N$, where N is the number of parallel instructions in the block. Let SC_i be the final state change of each sub-machine. The total change of state will be calculated as $\bigcup_{i=1}^{N} SC_i$. If $\bigcup_{i \neq j} SC_i \cap SC_j = \emptyset$, the state changes are consistent and the situation is entirely correct. Otherwise, we have some inconsistencies and the user will be informed about it. But, if the user has configured the non-determinism resolving procedure the execution can be continued. In this case the new values of the functions from the intersection set are chosen in accordance to the rules of simulation configuration. After the total state change is calculated all further instructions start from the new time point (see Delay settings). For example, consider the following parallel block:

```
[
  x := 1;
  y := 2;
  z := 3;
  x := x + y + z;
]
```

It is clear that the variable x is affected two times and the user can choose if the simulation process has to be stopped or it will be continued like in the situation of a non-deterministic choose operation. If the first assignment `x := 1;` is chosen the total time delay of the whole parallel block should be smaller than the total time delay of the parallel block with the second assignment `x := x + y + z;`.

8.4 Looped Parallel Block

If a parallel block is looped it can happen that there are no instructions for the current time moment to be executed. Therefore, infinite looping is possible. In this case we have to wait for the closest time moment at which at least one of the guards turns to true and some instructions can be executed. If such a time point does not exist a message is sent to the user. This time point is calculated and if it is not the time to exit the loop, the instructions concerning this time point are executed. If there are several guards that become true at this time point then all of the guarded instructions are executed. The execution of parallel instructions is performed just as in the case of a basic parallel block. The summary time delay and the resulting state change is calculated according to the rules specified for the parallel block. After the state is updated the simulation proceeds to the next iteration with a new time value. A case when all the delays are set to zero is detected by the simulator and a warning message can be sent to the user. The following is a simple example with guarded updates in a loop.

```
{
  x := 0; y := 0; z := 0;
  while (CT < 16) do [
    if (CT >= 12) then x := x + 1;
    if (CT >=  8) then y := y + 1;
    if (CT >= 17) then z := z + 1;
  ]
}
```

In this example it is clear that the third guarded update will not be executed. The others will be executed if the value of CT is below 16 when the simulator starts processing the loop statement. Each of these updates will be executed several times depending on the value of time delay specified for operations used in the update. In the first update there are three operations: read the value of x, add 1 to the value of x, save the calculated value to x. Therefore, the time delay for both of updates is the same and the time intervals are (12,16) and (8, 16) correspondingly. So if the value of CT was below 8 when the loop was entered the first guarded update will be executed about two times less than the second one.

8.5 Classification of Functions

In general, all functions can be separated into two classes: static functions and dynamic functions. A dynamic function can change its value, so it can be an internally used function or a function used as input or output. In the works of Egon Börger and Robert Stärk (e.g. [6]) the following notations for different function types are used: static (this type of functions cannot be modified); dynamic: in (read-only, input from the environment), out (write-only), controlled (internal), shared (no restrictions).

In this work a similar but more general notation is used for specifying the properties of a function. The possible configurations are considered from the point of view of read and write operations executed from the specified module or some external module (e.g., environment). The following is the table that shows all modifiers.

ASM module	Others	Comment
read	—	private static function
read	read	public static function
read	write	input function (in)
read	read & write	input controlled by other module
write	read	output function (out)
write	read & write	output controlled by other module
read & write	—	private dynamic function (controlled)
read & write	read	output that can be read
read & write	read & write	public dynamic function (shared)
others		not used

8.6 Send and Receive Operations

For communication between different processes we shall use special operations Send and Receive:

```
Send(To: Object; From: Object; Message: String?);
Receive(To: Object; From: Object?; Message: String?): Boolean;
WaitReceive(To: Object; From: Object?; Message: String?;
            WaitTime = inf: Time): Boolean;
```

The “To” field denotes the recipient object, the “From” field denotes the sender object and the “Message” contains the information we want to send. The Send method adds the given message to the end of the message queue. The Receive and WaitReceive operations deliver a message to its recipient. When one of these operations is executed the next message is extracted from the message queue and is passed to the executor. If the “From” field is undef, then the next message will be taken. If the “From” parameter is defined then only messages from the specified sender will be taken. The Receive method returns the next message or null if there is no one and WaitReceive waits for some time defined by the WaitTime parameter. The WaitTime parameter by default is **inf**, i.e. it will wait until a message comes. The Receive method can be modeled by WaitReceive if we set the WaitTime parameter to zero.

9 Conclusion

In this paper several new important features concerning the semantics of the ASM based language were described. These features will help us to build and verify specifications of real-time systems via a customizable simulation of the models. The most important parameters of simulation can be configured, i.e. external functions, time delays for language operations and constructs, non-determinism resolving. The semantics of Send and Receive statements is described. These statements are useful for encapsulation of data in a class and provide a method for communication between agents and synchronising the processes.

The whole project is aimed at development of a simulator for the described version of ASM language extension where the results of the current work are used. At the moment a simulator prototype is ready, which implements most of the specified features. The following is the list of the implemented features:

– Lexical and syntactical analysis with building a parse tree;
– Loading definitions of external functions from a file;
– Loading the simulation parameters;
– Simulation of most constructs and operations of Timed ASML;
– Checking the specified formulas during the simulation;
– Simulation results output with the history of all state changes.

References

1. Gurevich, Y.: Evolving algebras 1993: Lipari Guide. In Egon, B., ed.: Specification and Validation Methods. Oxford University Press (1995) 9–36
2. Beauquier, D., Slissenko, A.: A first order logic for specification of timed algorithms: Basic properties and a decidable class. Annals of Pure and Applied Logic **113** (2002) 13–52
3. Beauquier, D., Slissenko, A.: Periodicity based decidable classes in a first order timed logic. (Annals of Pure and Applied Logic) 38 pages. To appear.
4. Gurevich, Y.: Sequential abstract-state machines capture sequential algorithms. ACM Transactions on Computational Logic **1** (2000) 77–111
5. Huggins, J.: (University of Michigan, ASM homepage) http://www.eecs.umich.edu/gasm/.
6. Börger, E. Stärk, R.: Abstract State Machines: A Method for High-Level System Design and Analysis. (2003)
7. Beauquier, D., Crolard, T., Prokofieva, E.: Automatic parametric verification of a root contention protocol based on abstract state machines and first order timed logic. In Jensen, K., Podelski, A., eds.: Tools and Algorithms for the Construction and Analysis of Systems: 10th International Conference, TACAS 2004, Barcelona, Spain, March 29 – April 2, 2004. Lect. Notes in Comput. Sci., vol. 2988, Springer-Verlag Heidelberg (2004) 372–387
8. Foundations of Software Engineering — Microsoft Research, Microsoft Corporation: AsmL: The Abstract State Machine Language. (2002) http://research.microsoft.com/fse/asml/.
9. Soloviev, I. Usov, A.: The language of interpreter of distributed abstract state machines. Tools for Mathematical Modeling. Mathematical Research. **10** (2003) 161–170
10. Lamport, L.: A new solution of Dijkstra's concurrent programming problem. In: Communications of ACM, 17(8). (1974) 453–455
11. Cohen, J., Slissenko, A.: On verification of refinements of timed distributed algorithms. In Gurevich, Y., Kutter, P., Odersky, M., Thiele, L., eds.: Proc. of the Intern. Workshop on Abstract State Machines (ASM'2000), March 20–24, 2000, Switzerland, Monte Verita, Ticino. Lect. Notes in Comput. Sci., vol. 1912, Springer-Verlag (2000) 34–49
12. Börger, E. Gurevich, Y., Rosenzweig, D.: The bakery algorithm: yet another specification and verification. In Börger, E., ed.: Specification and Validation Methods. Oxford University Press (1995) 231–243
13. Usov, A.: Development of the Interpreter for Distributed ASMs. Electronic Notes in Theoretical Computer Science (2004)
14. Weispfenning, V.: Mixed real-integer linear quantifier elimination. In: Proc. of the 1999 Int. Symp. on Symbolic and Algebraic Computations (ISSAC'99), ACM Press (1999) 129–136

A Characterization of Meaningful Schedulers for Continuous-Time Markov Decision Processes

Nicolás Wolovick[1,⋆] and Sven Johr[2,⋆⋆]

[1] Fa.M.A.F., Universidad Nacional de Córdoba, Ciudad Universitaria, 5000 Córdoba, Argentina

[2] Universität des Saarlandes, FR 6.2 Informatik, D-66123 Saarbrücken, Germany

Abstract. Continuous-time Markov decision process are an important variant of labelled transition systems having nondeterminism through labels and stochasticity through exponential fire-time distributions. Nondeterministic choices are resolved using the notion of a *scheduler*. In this paper we characterize the class of *measurable* schedulers, which is the most general one, and show how a measurable scheduler induces a unique probability measure on the sigma-algebra of infinite paths. We then give evidence that for particular reachability properties it is sufficient to consider a subset of measurable schedulers. Having analyzed schedulers and their induced probability measures we finally show that each probability measure on the sigma-algebra of infinite paths is indeed induced by a measurable scheduler which proves that this class is complete.

1 Introduction

Continuous-time Markov decision processes (CTMDP) [1,2,3,4] are an important class of finite labelled transition systems (LTS). They have external nondeterminism through the interaction with edge labels and internal stochasticity given by the rates of negative exponential distributions. Besides their applications in, e. g., stochastic control theory [2], stochastic scheduling [5,6] and dynamic power management [7], these systems are interesting in their own, because they introduce a continuous quantity, namely the fire-time of transitions.

The external nondeterminism is usually reduced to probabilism using the notion of *schedulers*, also called *adversaries* or *policies*. Given that the system is in a particular state, a randomized scheduler eliminates the nondeterminism by making a decision for a certain probability distribution over edge labels. This decision making can be based on the present state alone, or together with all

⋆ Supported by the Programme Alßan, the European Union Programme of High Level Scholarships for Latin America, scholarship No.(E04D030410AR). Partially supported by PIP 6391: Métodos y Fundamentos para el Análisis de Corrección y Desempeño de Programas Concurrentes y Aleatorios, funded by CONICET, and PICT 26135: Verification of Probabilistic Distributed Systems, funded by Agencia.

⋆⋆ This work is 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.

previous states, edge labels and time points, leading to memoryless or timed history-dependent schedulers, respectively. Besides, it is also possible to have deterministic schedulers as a particularization of randomized schedulers. However, as shown in [8], timed history-dependent schedulers are strictly more powerful than schedulers abstracting from time. Therefore time has an explicit role in the global behavior of the model of CTMDPs and cannot, in principle, be avoided. We take the most general class of schedulers, namely randomized timed history-dependent (THR) schedulers as our starting point.

Time is continuous in its nature and thus particular issues quickly arise. Many of them refer to practical aspects as discretization, abstraction, approximation and computation, but others like measurability are more fundamental [9]. It is well known, that there are sets, namely *Vitali sets*, in the real interval $[0, 1]$ that cannot be measured. A measure is a function that generalizes the usual notions of cardinality, length, area and probability. With these unreasonable sets, all the questions about our system asking for a quantitative answer start to be endangered. The motivation of this paper is to tackle this particular problem.

We give a *soundness* result, showing that measurable schedulers generate a well-defined probability measure for timed paths. For this we first construct a combined transition that merges scheduler and CTMDP probabilism in a single transition probability. We establish that measurability of combined transitions is inherited from schedulers and vice versa, improving a result from [9]. Based on transition probabilities we construct a probability measure on the set of infinite timed paths with the aid of canonical product measure theorems.

A proper subclass of measurable schedulers inducing positive probability measures on infinite paths is identified. Besides that and the construction of scheduler dependent probability measures, we also answer the question if there is a proper subclass of measurable schedulers which is easily definable and generates arbitrary measurable schedulers (called *probabilistically complete*) in the affirmative.

Finally a *completeness* result is presented, showing that every timed path probability is generated by some measurable scheduler. To do this we motivate the need of this probability to be related or *compatible* with the underlying CTMDP. This compatibility cannot be stated using the probability measure. Instead, (continuous) conditional probabilities are obtained by the so-called Radon-Nikodym derivatives. However, those conditional probabilities turn out to be insufficient for the intended compatibility result. Taking advantage of our particular setting (timed paths as denumerable product of discrete spaces and positive reals), we use *disintegrability* to obtain transition probabilities instead of conditional ones. Deconstructing the probability into transition probability lead us to state compatibility and the completeness theorem.

Outline. The rest of the paper is organized as follows. In Section 2 we give an overview of related work. Section 3 establishes mathematical background and notation, including the CTMDP model with schedulers, as well as the measurability problem. Section 4 develops a timed path probability measure w.r.t. measurable schedulers. In Section 5 we propose the simple scheduler subclass and discuss its limiting properties. Section 6 shows that measurable schedulers can capture every probability measure generated by a CTMDP. Finally, Section 7 concludes the paper.

2 Related Work

In systems with continuous state spaces featuring nondeterminism and probabilism, measurability issues of schedulers have been studied from the point of view of timed systems [10], extensions of discrete probabilistic automata [9] and Markov decision process (MDP) with Borel state space [11].

In [10] continuous probabilistic timed automata are defined and its concrete semantics is given in terms of dense Markov process once the nondeterminism is resolved by a timed history-dependent scheduler. Although the model is not comparable with CTMDPs because of the denseness of the state space, scheduler measurability issues also appear. The authors disregard this particular problem by considering it pathological.

Stochastic transition systems [9] are a generalization of CTMDPs and the measurability of schedulers is an important issue here. Cattani et al. show how to construct a measure on so called *executions*. For this they have to, and indeed do discard nonmeasurable schedulers. In particular, although we discuss our results in a more restrictive modelling formalism, this work is an improvement over their construction in two aspects. First, we use standard measure-theoretic results in conjunction with product space measures, leading to results that are more compact and provide deeper insight than the results presented in [9]. Secondly, the need for measurable functions in the construction of the probability measure is in this paper directly inherited from the definition of scheduler measurability. Instead, in [9] measurability properties are kind of directly given for the according functions [9, Definition 5] without giving a connection to schedulers. We added to this work a complete characterization of measurable schedulers, something that has not been considered in [9].

A result similar to our deconstruction discussed in Section 6 is given in the context of epistemic game theory [12]. The proof-tools used there are disintegration theorems for the product of two spaces [13] and the so-called *Ionescu-Tulcea* theorem. However, we prove a similar theorem by resorting to the well-known Radon-Nikodym theorem together with canonical product measures for denumerable product spaces.

To the best of our knowledge, the decomposition of particular probability measures on the infinite behavior of probabilistic systems with nondeterminism leading to measurable schedulers has not been considered in the literature so far.

3 Background and Problem Statement

3.1 Mathematical Notation and Background

By $\mathbb{R}$ and $\mathbb{N}$ the set of *real numbers* and *natural numbers* are denoted, $\mathbb{R}^+$ is the set of positive real numbers including 0. *Disjoint union* is denoted $\Omega_1 \uplus \Omega_2$, while *partial function application* is denoted $f(\cdot, \omega_2)$.

Given a set Ω and a collection $\mathcal{F}$ of subsets of Ω, we call $\mathcal{F}$ a *σ-algebra* iff $\Omega \in \mathcal{F}$ and $\mathcal{F}$ is closed under complement and denumerable disjoint union. The *σ-algebra generated by* the family $\mathcal{H} \in 2^{\Omega}$ is the minimal σ-algebra containing $\mathcal{H}$. We call the pair $(\Omega, \mathcal{F})$ a *measurable space.* A *measurable set* is denoted as $A \in \mathcal{F}$. When dealing with a discrete set Ω, we take the powerset σ-algebra 2^{Ω} as the default, however we still denote a *measurable set* as $A \in \mathcal{F}$ instead of the simpler $A \subseteq \Omega$. Let $(\Omega_i, \mathcal{F}_i), i = 1, 2, \ldots, n$ be arbitrary measurable spaces and $\Omega = \Omega_1 \times \Omega_2 \times \cdots \times \Omega_n$. A *measurable rectangle* in Ω is a set $A = A_1 \times A_2 \times \cdots \times A_n$, where $A_i \in \mathcal{F}_i$. The *product space σ-algebra* denoted $\mathcal{F}_{\Omega_1 \times \Omega_2 \times \cdots \times \Omega_n}$ is the σ-algebra generated by measurable rectangles. Given a measurable space $(\Omega, \mathcal{F})$, a σ-additive function $\mu : \mathcal{F} \to \mathbb{R}^+$ is called *measure*, and if $\mu(\Omega) = 1$ it is called *probability measure.* The triple $(\Omega, \mathcal{F}, \mu)$ is a *measure space* or *probability space* depending on μ. For a measurable set $(\Omega, \mathcal{F})$ we denote by $\mathrm{Distr}(\Omega)$ the *set of all probability distributions* over Ω, and given $\mu \in \mathrm{Distr}(\Omega)$ a *support* of μ is a measurable set A such that $\mu(A) = 1$. A function $f : (\Omega_1, \mathcal{F}_1) \to (\Omega_2, \mathcal{F}_2)$ is *measurable* if $\forall A_2 \in \mathcal{F}_2, f^{-1}(A_2) \in \mathcal{F}_1$, i.e., the inverse function maps measurables to measurables. A measurable predicate $P : \Omega \to \mathit{Bool}$ in a measure space $(\Omega, \mathcal{F}, \mu)$ is *μ-almost everywhere valid*, P a.e. $[\mu]$, iff $\mu(P^{-1}(\mathit{false})) = 0$, that is the set of non-valid points is negligible. We call a function $f : \Omega_1 \times \mathcal{F}_2 \to [0, 1]$ *transition probability* or *Markov kernel* iff for all $\omega_1 \in \Omega_1$, $f(\omega_1, \cdot)$ is a probability measure on $(\Omega_2, \mathcal{F}_2)$ and for all $A_2 \in \mathcal{F}_2$, $f(\cdot, A_2)$ is a measurable function.

3.2 Continuous-Time Markov Decision Processes

In the following we present the basic definitions of continuous-time Markov decision processes. We use essentially the same notation as in [3].

Definition 1 (CTMDP). *A* continuous-time Markov decision process *(CTMDP) is a tuple* $(S, L, \mathbf{R}, \Pr_{init})$ *where S is a finite non-empty set of* states*, L is a finite non-empty set of* transition labels *also called* actions*,* $\mathbf{R} : S \times L \times S \to \mathbb{R}^+$ *is the three-dimensional* rate matrix*,* $\Pr_{init} \in \mathit{Distr}(S)$ *is the* initial distribution *over states.*

Given a CTMDP tuple $\mathcal{C} = (S, L, \mathbf{R}, \Pr_{\mathrm{init}})$, we define the projection function $\mathcal{C}_{\Pr_{\mathrm{init}}} \doteq \Pr_{\mathrm{init}}$, and so on for the other coordinates. For set $Q \subseteq S$ we denote by $\mathbf{R}(s, a, Q) \doteq \sum_{s' \in Q} \mathbf{R}(s, a, s')$ the cumulative rate to leave state s under label a. In $L_s \doteq \{a \in L \mid \mathbf{R}(s, a, S) > 0\}$ we collect all labels that belong to transitions emanating from s.

Behavior. The behavior of a CTMDP is as follows. $\mathbf{R}(s, a, s') > 0$ denotes that there exists a transition from s to s' under label a where $\mathbf{R}(s, a, s')$ corresponds to the rate of a negative exponential distribution. When s has more than one successor state under label a, one of them is selected according to the *race condition.* The discrete branching probability $\mathbf{P}_s(a, s')$ from s to s' under label a is given by $\mathbf{P}_s(a, s') \doteq \frac{\mathbf{R}(s,a,s')}{\mathbf{R}(s,a,S)}$, where $\mathbf{R}(s, a, S)$ is the overall exit rate of s

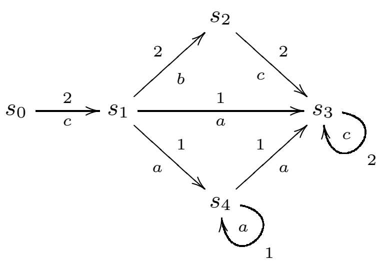


Fig. 1. Continuous-time Markov decision process

under label a. The probability that one of the successors of s is reached within time t is given by $1 - e^{-\mathbf{R}(s,a,S)\cdot t}$. In CTMCs this time is also referred to as the *sojourn time* in state s. Given that $a \in L$ is chosen, the sojourn time in $s \in S$ is determined by a negative exponential distribution with rate $\mathbf{R}(s,a,S)$.

Figure 1 shows a simple CTMDP. Arrows indicate transitions between states. They are labelled by an action and a rate, e. g., under label a there exists a transition from s_1 to s_3 whose fire-time is exponentially distributed with rate 1.

3.3 Timed Paths

A *timed path* σ in CTMDP $\mathcal{C} = (S, L, \mathbf{R}, \mathrm{Pr}_{\mathrm{init}})$ is a possibly infinite sequence of states, labels and time points, i. e., $\sigma \in \big(S \times (L \times \mathbb{R}^+ \times S)^*\big) \uplus \big(S \times (L \times \mathbb{R}^+ \times S)^\omega\big)$. For a given path $\sigma = s_0 a_1 t_1 s_1 \ldots$ we denote by *first*$(\sigma) = s_0$ its first state, $\sigma[k]$ denotes its $(k+1)$-st state, e. g., $\sigma[0] = $ *first*(σ). A finite path $\sigma' = s_0 a_1 t_1 s_1 \ldots a_k t_k s_k$ has *length* k, denoted as $|\sigma'| = k$ and its last state equals *last*$(\sigma) = s_k$. For arbitrary paths σ we denote by σ^i the *prefix of length* i of σ, i. e., $\sigma^i = s_0 a_1 t_1 s_1 \ldots a_i t_i s_i$. For finite $\sigma' = s_0 a_1 t_1 s_1 \ldots a_k t_k s_k$, the prefix of length $i > k$ equals σ'. Path^n, Path^*, Path^ω and Path denote the sets of paths of length n, *finite* paths, *infinite* paths and the union thereof, respectively, where $\mathit{Path} = \mathit{Path}^* \uplus \mathit{Path}^\omega$ and $\mathit{Path}^* = \biguplus_{n \in \mathbb{N}} \mathit{Path}^n$. A *time abstract* path is a timed path where time points are omitted. It will be clear from context which type of path we use in particular sections. Thus, the above given definitions carry over to time abstract paths.

Paths are defined as *arbitrary* sequences of states, labels and time points w.r.t. $\mathcal{C}$. We do not require that $\mathbf{R}(s_i, a_i, s_{i+1}) > 0$ but we will overcome this obstacle by giving all power to our schedulers. Given CTMDP $\mathcal{C}$ we distinguish between *observable* and *unobservable* paths w.r.t. $\mathcal{C}$. A path $\sigma = s_0 a_1 t_1 s_1 \ldots$ is called observable iff $\mathbf{R}(s_i, a_{i+1}, s_{i+1}) > 0$ for $i = 0, 1, \ldots$. Otherwise we call σ *unobservable*. As will be evident in Section 4, the probability measure on paths induced by a given scheduler will evaluate to 0 for unobservable paths. But before we introduce the measure, we have to define the measurable space over paths.

σ-Algebra over timed paths. We endow states and labels with the powerset σ-algebra, and $\mathbb{R}^+$ with the standard Borel σ-algebra. The σ-algebra $\mathcal{F}_{Path^n}$ is the standard product space σ-algebra, $\mathcal{F}_{S\times(L\times\mathbb{R}^+\times S)^n}$, generated by the rectangles $Q_0 \times M_1 \times \cdots \times M_n$, where $Q_0 \in \mathcal{F}_S$ and $M_i \in \mathcal{F}_{L\times\mathbb{R}^+\times S}$. The σ-algebra $\mathcal{F}_{Path^\omega}$ is defined using the concept of *cylinders* [14, Definition 2.7.1]. Given the set of timed paths Λ with $|\Lambda| = k$, $C_\Lambda \doteq \{\sigma \in Path^\omega \mid \sigma^k \in \Lambda\}$ defines a cylinder with *base* Λ. A *measurable cylinder* has measurable base, and a *measurable rectangle* (in $Path^\omega$) is a cylinder whose base is a measurable rectangle. The σ-algebra $\mathcal{F}_{Path^\omega}$ is the minimal σ-algebra generated by either measurable cylinders or measurable rectangles. Finally $\mathcal{F}_{Path}$ is standard σ-algebra for the disjoint union of $\{\mathcal{F}_{Path^n}\}_{n\in\mathbb{N}}$ and $\mathcal{F}_{Path^\omega}$.

3.4 Schedulers

A scheduler resolves the nondeterminism inherent in a CTMDP. Most generally, schedulers can be considered as functions from finite paths to probability distributions over labels. Such a general definition allows for all special cases as, e. g., given in [3].

Definition 2 (Scheduler over CTMDP). *Let $\mathcal{C}$ be a CTMDP with label set L. A* scheduler D over $\mathcal{C}$ *is a function $D : Path^* \to Distr(L)$ such that the support of $D(\sigma)$ is equal to $L_{last(\sigma)}$.*

The support condition says that the scheduler distributes its whole mass among the outgoing labels. We can also denote D as a two argument function $D : Path^* \times \mathcal{F}_L \to [0, 1]$, being a measure in its second argument $D(\sigma, \cdot)$ for all finite path σ. We now turn our attention to the *measurability problem.* This is motivated by the fact that there exist "bad-behaved" schedulers. Reconsider the CTMDP depicted in Figure 1 and the randomized timed history-dependent (THR) scheduler D with

$$D(s_0cts_1) = \begin{cases} \delta_a & \text{if } t \in \mathcal{V}, \\ \delta_b & \text{otherwise,} \end{cases}$$

where $\delta_i, i \in \{a, b\}$ denotes that action i is chosen with probability 1 and $\mathcal{V}$ is the *nonmeasurable Vitali set* in $[0, 1]$, i. e., for each time point in a nonmeasurable set, D will choose deterministically a and for all other time points, D will choose deterministically b. So the question, "*What is the probability to reach s_3 in two steps?*", cannot be answered, because it necessarily involves the evaluation of a measure in the set $\{t|D(s_0cts_1) = \delta_a\} = \mathcal{V}$ that is nonmeasurable.

In the subsequent definition we restrict to the class of schedulers that respects measurability issues.

Definition 3 (Measurable scheduler). *We call a scheduler D over CTMDP C* measurable scheduler *iff for all $A \in \mathcal{F}_L$, $D(\cdot, A) : Path^* \to [0, 1]$ is a measurable function.*

4 Constructing Timed Path Probability

Probabilistic schedulers allow to quantify uncertainty. Therefore composing CTMDP negative exponential distributions with a particular scheduler, we can construct probability measures usually called *combined transitions* [9].

Definition 4 (Combined transition). *For a given CTMDP $\mathcal{C}$, scheduler D and finite path σ, the* combined transition $\mu_D : \mathit{Path}^* \times \mathcal{F}_{L\times\mathbb{R}^+\times S} \to [0,1]$ *is defined in the measurable rectangles by*

$$\mu_D(\sigma, A\times\mathcal{I}\times Q) \doteq \sum_{a\in A} D(\sigma,\{a\})\cdot\mathbf{P}_{\mathit{last}(\sigma)}(a,Q)\cdot\int_{\mathcal{I}}\mathbf{R}(\mathit{last}(\sigma),a,S)e^{-\mathbf{R}(\mathit{last}(\sigma),a,S)\cdot t}dt\ . \tag{1}$$

The second factor in the terms of the above summation is the discrete branching probability from $\mathit{last}(\sigma)$ to Q by the label a, while the third one is the probability that the transition triggers within measurable time set $\mathcal{I}$.

We have, thanks to support restriction in the scheduler, $\mu_D(\sigma, L\times\mathbb{R}^+\times S) = 1$, however μ_D is just defined for the $\mathcal{F}_{L\times\mathbb{R}^+\times S}$ generators. The extension to the whole σ-algebra is standard and follows by applying Carathéodory Extension Theorem [14, Theorem 1.3.10] to the field of finite disjoint unions of measurable rectangles [14, Problem 2.6.1].

Lemma 1 (Combined transition is a probability measure). *Given a CTMDP $\mathcal{C}$ and a finite path σ, the combined transition $\mu_D(\sigma,\cdot)$ as defined in (1) extends uniquely to a probability measure on $\mathcal{F}_{L\times\mathbb{R}^+\times S}$.*

Additionally we deduce the following lemma, where the *only if* implication is given by Lemma 1 and the *if* implication can be established by considering Definitions 2 and 4.

Lemma 2. *Given a CTMDP $\mathcal{C}$ and scheduler D, $\forall\sigma\in\mathcal{F}_{\mathit{Path}^*}$, it is the case that $D(\sigma,\cdot) : \mathcal{F}_L \to [0,1]$ is a measure iff $\forall\sigma\in\mathcal{F}_{\mathit{Path}^*}$, we have $\mu_D(\sigma,\cdot) : \mathcal{F}_{L\times\mathbb{R}^+\times S} \to [0,1]$ is a measure.*

Next we prove that combined transition measurability is inherited from scheduler measurability and vice versa, and this will be crucial to integrate and disintegrate a probability measure on timed paths.

Theorem 1 (Combined transition measurability). *Given a CTMDP $\mathcal{C}$ and scheduler D, $\forall A\in\mathcal{F}_L$, it is the case that $D(\cdot,A) : \mathit{Path}^* \to [0,1]$ is measurable iff $\forall M\in\mathcal{F}_{L\times\mathbb{R}^+\times S}$, we have $\mu_D(\cdot,M) : \mathit{Path}^* \to [0,1]$ is measurable.*

Proof. We prove each direction separately.

Only if: Having $D(\cdot,A)$ measurable, function $\mu_D(\cdot, A\times\mathcal{I}\times Q)$ is measurable in its first argument because projection functions like $\mathit{last}()$ are measurable, as well as functions coming from powerset σ-algebras like $\mathcal{F}_S$ and $\mathcal{F}_L$. Closure properties of measurable functions (sum, product, composition) make the entire expression measurable.

We have to extend this result to the whole σ-algebra $\mathcal{F}_{L\times\mathbb{R}^+\times S}$. We resort to the *good sets principle* [14]. Let $\mathcal{G} = \{M \in \mathcal{F}_{L\times\mathbb{R}^+\times S} \mid \mu_D(\cdot, M)$ *is a measurable function*$\}$ and show that the set $\mathcal{G}$ forms a σ-algebra. Let $\{M_i\}_{i\in\mathbb{N}}$ be a disjoint collection such that $M_i \in \mathcal{G}$, then by σ-additivity $\mu_D(\cdot, \biguplus_{i\in\mathbb{N}} M_i) = \sum_{i\in\mathbb{N}} \mu_D(\cdot, M_i)$, and this is measurable by closure properties of measurable functions, therefore $\mathcal{G}$ is closed under denumerable disjoint union. It is also closed under complement as $\mu_D(\cdot, M^c) = \mu_D(\cdot, L\times\mathbb{R}^+\times S) - \mu_D(\cdot, M)$. Hence all sets in $\sigma(\mathcal{G})$ are good, and since rectangles are included in $\mathcal{G}$ we have $\sigma(\mathcal{G}) = \mathcal{F}_{L\times\mathbb{R}^+\times S}$, concluding $\mu_D(\cdot, M)$ is measurable for all $M \in \mathcal{F}_{L\times\mathbb{R}^+\times S}$.

If: By hypothesis the function $\mu_D(\cdot, A\times\mathbb{R}^+\times S)$ is measurable for all $A \in \mathcal{F}_L$, but is easy to check that $\mu_D(\cdot, A\times\mathbb{R}^+\times S) = D(\cdot, A)$, finishing our result. □

Given the previous results, we can deduce the following corollary.

Corollary 1. *Let D be a measurable scheduler on CTMDP $\mathcal{C}$. D is a transition probability iff μ_D is so.*

From now on we use canonical product measure theory [14, Sections 2.6, 2.7] to define finite and infinite timed path probability measure, where the combined transition plays a central role.

Definition 5 (Probability measure for measurable rectangle finite paths). *Let $\mathcal{C}$ be a CTMDP, D a measurable scheduler, and μ_D their combined transition. The* probability measure for finite paths consisting of measurable rectangles *Pr_D is given inductively as follows:*

$$Pr_D^0(S_0) = \mathcal{C}_{\mathrm{Pr}_{init}}(S_0) \ , \tag{2}$$

$$Pr_D^{n+1}(\Lambda \times M_{n+1}) = \int_{\sigma\in\Lambda} \mu_D(\sigma, M_{n+1}) dPr_D^n(\sigma) \ . \tag{3}$$

where Pr_{init} *is the initial distribution, Λ is a measurable rectangle of $\mathcal{F}_{Path^n}$, and $M_{n+1} \in \mathcal{F}_{L\times\mathbb{R}^+\times S}$.*

Combined transitions are measurable, thus the above Lebesgue integral is well-defined. However, the definition only captures the measurable rectangles and we have to extend the measure to the complete σ-algebra. The unique extension of Pr_D^n is given by [14, Theorem 2.6.7].

Lemma 3 (Probability measure for finite paths). *For each n there is a unique probability measure Pr_D^n over the σ-algebra $(Path^n, \mathcal{F}_{Path^n})$ that extends Pr_D^n defined in (2),(3).*

Note that Pr_D^n is not a probability measure for finite paths but a denumerable set of probability measures one for each path length. They can be put together in a single probability measure space, namely the infinite timed path measure space $(Path^\omega, \mathcal{F}_{Path^\omega}, Pr_D^\omega)$ [14, Theorem 2.7.2] [15, Ionescu-Tulcea Theorem].

Lemma 4 (Probability measure for infinite paths). *Given the measurable space* $(Path^{\omega}, \mathcal{F}_{Path^{\omega}})$*, if we define a probability measure on measurable rectangle bases as in Definition 5, then there is a unique probability measure* Pr_D^{ω} *on* $\mathcal{F}_{Path^{\omega}}$ *such that for all* n*,* $Pr_D^{\omega}(C_{\Lambda}) = Pr_D^{n}(\Lambda)$ *with* $\Lambda \in \mathcal{F}_{Path^{n}}$*.*

The previous results can be summarized in the following theorem.

Theorem 2 (Soundness). *Given a measurable scheduler* D *over CTMDP* $\mathcal{C}$*, then a probability measure* Pr_D^{ω} *over* $\mathcal{F}_{Path^{\omega}}$ *can be constructed.*

Lemma 3 and Lemma 4 define probability measures for each of the disjoint measurable spaces $(\uplus_{n\in\mathbb{N}}(Path^{n}, \mathcal{F}_{Path^{n}}))\uplus(Path^{\omega}, \mathcal{F}_{Path^{\omega}})$. A unifying measure (that is not a probability) on the disjoint union space can be defined in a standard way and is denoted by Pr_D. It is important to remark that under this way of constructing Pr, there is no CTMDP that can "hide" a nonmeasurable scheduler, namely having a well defined timed path probability, implies that the combined transition is measurable, and by Theorem 1 this implies a measurable scheduler.

5 Meaningful Schedulers

In the previous section we have defined a measure on paths induced by measurable schedulers by using measure theoretic results and in particular abstract Lebesgue integration [14]. Now we examine the integral in more detail and characterize various classes of measurable schedulers that respect more than just the initial probability distribution over states. This boils down to investigate basic properties of abstract Lebesgue integrals over combined transitions. These classes of schedulers are of special interest because they comprise all schedulers that contribute to, e. g., particular reachability properties like: "what is the maximum probability to reach a set B of states within t time units" [8].

The following results are taken from [14] and give the theoretical background for our observations. Each nonnegative Borel measurable function f is the limit of a sequence of increasing simple functions that are nonnegative and finite-valued [14, Theorem 1.5.5]. And so, given a measurable scheduler D the combined transition $\mu_D(\cdot, M)$ is measurable for each $M \in \mathcal{F}_{L\times\mathbb{R}^{+}\times S}$ and is the limit of an increasing sequence of simple functions, say, $\mu_D^{i}(\cdot, M)$. Formally, we define $\mu_D^{i}(\cdot, M)$, for fixed $M \in \mathcal{F}_{L\times\mathbb{R}^{+}\times S}$ as

$$\mu_D^{i}(\sigma, M) \doteq \frac{k-1}{2^{i}} \text{ if } \frac{k-1}{2^{i}} \leq \mu_D(\sigma, M) < \frac{k}{2^{i}}, k = 1, 2, \ldots, 2^{i} \ . \tag{4}$$

Due to convergence properties of the abstract Lebesgue integral [14, Theorem 1.6.2] it holds that $Pr_D^{n}(\Lambda \times M) = \lim_{i\mapsto\infty} \int_{\sigma\in\Lambda} \mu_D^{i}(\sigma, M) dPr_D^{n-1}(\sigma)$, i. e., the probability distribution over infinite paths induced by D coincides with the limiting sequence of integrals over μ_D^{i}. Recall, that the probability measure over infinite paths is defined w.r.t. finite path-prefixes. Thus we deduce for given Λ with $|\Lambda| = n - 1$.

$$\begin{aligned} Pr^n_D(C_{\Lambda a \mathcal{I} s}) &= \int_{\sigma \in \Lambda} \mu_D(\sigma, (a, \mathcal{I}, s)) dPr^{n-1}_D(C_\sigma) \\ &= \lim_{i \mapsto \infty} \int_{\sigma \in \Lambda} \mu^i_D(\sigma, (a, \mathcal{I}, s)) dPr^{n-1}_D(C_\sigma) \\ &= \lim_{i \mapsto \infty} \left[\sum_{j=1}^{2^i} \frac{j-1}{2^i} \cdot Pr^{n-1}_D(A^i_j) \right] , \end{aligned} \tag{5}$$

where $A^i_j \doteq \left\{\sigma \in \Lambda \mid \frac{j-1}{2^i} \leq \mu_D(\sigma, (a, \mathcal{I}, s)) < \frac{j}{2^i}\right\}$. The summation in Equation 5 is a direct consequence of the definition of Lebesgue integration over simple functions. As a result we see, that the induced probability measure evaluates to zero iff the limiting sequence evaluates to zero.

Almost Always Measure Zero Schedulers. First of all we characterize the class of schedulers that gives measure zero to almost all sets of paths of a given CTMDP $\mathcal{C}$. We refer to the schedulers of this class as *almost always measure zero schedulers.* In this class we summarize all schedulers for which it is not possible to find cylinders on which they give positive probability. We restrict to particular cylinders as, e. g., cylinders $C_s, s \in S$ give for each scheduler D that $Pr^\omega_D(C_s)$ equals $\text{Pr}_{\text{init}}(s)$. Dependent on scheduler D we can always give at least one cylinder base Λ of length two such that $Pr_D(C_\Lambda)$ is positive, e. g., for $\Lambda = s_0 a_1 I_1 s_1$, where I_1 is a right-open interval with $|I_1| > 0$ and $D(s_0, \{a_1\}) > 0$. This motivates the following definition.

Definition 6 (Almost always measure zero scheduler). *We call scheduler D* almost always measure zero scheduler *(AAMZ) iff $Pr_D(C_\Lambda) = 0$ for all Λ with $|\Lambda| \geq 2$.*

The probability measure Pr^ω_D on the set of infinite paths induced by scheduler D evaluates to 0 iff μ_D equals 0 a.e. $[Pr_D]$. Due to the inductive definition of Pr^ω_D it can be observed, that $\mu_D(\sigma', (a', I', s')) = 0$ at some stage of the computation for all $(\sigma', (a', I', s'))$. We now describe for which type of schedulers this is generally true. For this we use the notion of time abstract paths and show that AAMZ schedulers do not give positive probability to all paths σ with corresponding time abstract path σ' that is observable in the system. We put this restriction in our analysis since the probability measure of unobservable paths is always zero, independently of the scheduler. Thus, the probability measure only depends on the set of time points $\mathcal{I}$ that has to be considered.

Based on $\mathcal{I}$ we compute the sojourn time distribution inside of the combined transition. As sojourn times are distributed according to negative exponential distributions, this reduces to the standard computation of a Riemann integral. Thus it is sufficient to investigate when this integral evaluates to zero which is the case for all point-intervals I. Thus, if $\mathcal{I}$ comprises only point-intervals the probability measure will evaluate to zero. This means, if we can not group paths together in a set Λ' for which D behaves *friendly*, the probability measure will not be positive.

Definition 7 (Friendly scheduler). *Let $\sigma = s_0 a_1 t_1 s_1 \ldots a_k t_k s_k$ be an arbitrary finite path. We say that D behaves* friendly on σ *iff*

$$\forall t_i : \exists I_i : t_i \in I_i \ \wedge \ |I_i| > 0 \ \wedge \ \forall t \in I_i : D(s_0 a_1 t_1 s_1 \ldots a_i t s_i, \{a_{i+1}\}) > 0, \quad (6)$$

where I_i is an interval on $\mathbb{R}^+$.

A scheduler that does not behave friendly on path σ is called *unfriendly.* We can establish the following lemma.

Lemma 5 (Almost always measure zero). *If D behaves unfriendly on all finite paths σ then D is an AAMZ.*

Suppose D is a scheduler behaving unfriendly on all finite paths. Further assume that we have as cylinder base $\Lambda = s_0 a_1 \mathcal{I}_1 s_1 \ldots a_k \mathcal{I}_k s_k$. It holds for D (by Definition 7) that $\mathcal{I}_i$ cannot be partitioned into intervals I_i^j with $|I_i^j| > 0$ such that $D(s_0 a_1 t_1 s_1 \ldots a_i t_i s_i, \{a_{i+1}\}) > 0$ for all $t_i \in I_i^j$. As a consequence, we can only partition Λ into singular paths σ but $Pr_D^\omega(C_\sigma) = 0$.

Restricted Class of Schedulers. Now we discuss a restricted class of schedulers that gives positive probability to particular sets of infinite paths. This restriction depends on the property we want to check for a given CTMDP, i. e., when checking a timed reachability property like "what is the maximum probability to reach set B within time t" we are only interested in schedulers that *contribute* to that probability in a sense that they give positive probability to all sets of paths hitting B before time t. It is thus sufficient to consider schedulers that behave friendly on at least one time abstract path that hits B.

The following example shows how simple the computation of the probability measure is, when the given scheduler behaves friendly on the given set of paths. Assume that $\Lambda = s_0 a_1 I_1 s_1 \ldots a_k I_k s_k$ is a given set of finite paths (which all share the same time-abstract path), where I_i is a nonsingular interval of length greater than 0. Now suppose scheduler D is such that each decision is *consistent* in each of the I_i, i. e., $D(s_0 a_0 t_0 s_1 \ldots a_i t_i^m s_{i+1}) = D(s_0 a_0 t_0 s_1 \ldots a_i t_i^n s_{i+1})$ for all $t_i^m, t_i^n \in I_i$. The probability of cylinder C_Λ induced by D is given by

$$\begin{aligned}
Pr_D^\omega(C_\Lambda) &= Pr_D^\omega(C_{\Lambda^{k-1} a_k I_k s_k}) \\
&= \int_{\sigma \in \Lambda'} \mu_D(\sigma, (a_k, I_k, s_k)) dPr_D^\omega(C_\sigma) \\
&= \int_{\sigma \in \Lambda'} \mathbf{1}_{\Lambda'}(\sigma) \mu_D(\sigma, (a_k, I_k, s_k)) dPr_D^\omega(C_\sigma) \\
&= \mu_D(\Lambda', (a_k, I_k, s_k)) \cdot Pr_D^\omega(C_{\Lambda'}) \\
&\ \vdots \\
&= \mathrm{Pr}_{\mathrm{init}}(s_0) \cdot \mu_D(s_0, (a_1, I_1, s_1)) \cdot \mu_D(s_0 a_1 I_1 s_1, (a_2, I_2, s_2)) \cdot \cdots \\
&\quad \cdot \mu_D(s_0 a_1 I_1 s_1 \ldots a_{k-1} I_{k-1} s_{k-1}, (a_k, I_k, s_k)) \ ,
\end{aligned}$$

where we use, e. g., $D(\Lambda')$ to denote $D(\sigma)$ for arbitrary $\sigma \in \Lambda'$. In this case non zero probability is given as long as $D(s_0 a_1 I_1 s_1 \ldots a_i I_i s_i, \{a_{i+1}\}) > 0$ which is a weak form of a friendly behaving scheduler.

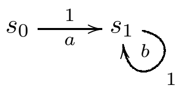
Simple Scheduler. In this section we give the definition of an important class of schedulers, namely *simple schedulers*. Simple functions are functions that take on only finitely many different values. Along this line we define simple schedulers to have a finite range only, too. In particular, simple scheduler D gives a partition P of the set of finite paths in finitely many blocks B^j, for $j, n \in \mathbb{N}$ with $n > 0$ and $j \leq n$. Given arbitrary block B^j of P it holds that $D(\sigma) = D(\sigma')$ for all $\sigma, \sigma' \in B^j_i$. Thus, a simple scheduler D is a simple function on the set of finite paths. As a direct consequence, it can be observed that when D is simple, μ_D is also a simple function in its first argument, i. e., $\mu_D(\cdot, M)$ is a simple function for given $M \in \mathcal{F}_{L\times\mathbb{R}^+\times S}$. As we have discussed earlier in this section every measurable combined transition is the limit of simple combined transitions. This, and the fact that each simple combined transition can only be generated by a simple scheduler leads to the result that the set of simple schedulers can be used to generate arbitrary measurable schedulers and thus, this class is *probabilistically complete*.

Simple schedulers can be partitioned in two sets, namely *friendly simple* and *AAMZ simple* schedulers. An easy example of a friendly simple scheduler is as follows. Suppose label a is in L_s for all $s \in S$. A scheduler that always chooses a with probability 1 is friendly and simple. As an example of an AAMZ let D be a scheduler and $\mu_1, \mu_2 \in \mathrm{Distr}(L)$. When D schedules for all σ where $t_{|\sigma|}$ is an irrational number μ_1, and μ_2 otherwise, then D is simple and an AAMZ scheduler. Simplicity follows from the fact that D is only supposed to decide between μ_1 and μ_2. There exists no path σ such that Equation 6 can be fulfilled for D, thus D is an unfriendly scheduler and by definition an AAMZ scheduler.

6 Deconstructing Timed Path Probability

In Section 4 a measure Pr_D for finite and infinite paths was constructed from a measurable scheduler D. The goal now is to recover a measurable scheduler from an arbitrary probability measure Pr^ω (note there is no subscript D indicating the scheduler generating it) on the infinite timed path σ-algebra. This shows that measurable schedulers are sufficient to generate all quantitative behaviors of a CTMDP.

Given a CTMDP not every $Pr^\omega \in \mathrm{Distr}(Path^\omega)$ is related or *compatible* with it. For example in the CTMDP $\mathcal{C}$ below, a probability measure Pr^ω such that $Pr^\omega(s_0a[0,1]s_1a[0,1]s_1) > 0$ is not compatible with $\mathcal{C}$. We postpone the definition of compatibility until we settle down some continuous space measure theoretic results.



For a given timed path σ finishing at s, we are aiming at identifying two independent random sources: the scheduler selecting a label after history σ, and the sojourn time probability induced by the rate matrix.

Given Pr^{ω}, the tool to compute this product probability is a *conditional probability*, because we want to know the probability of the event $C_{\sigma a \mathcal{I} s}$ given that event C_{σ} happened.

$$Pr^{\omega}(C_{\sigma a \mathcal{I} s'} \mid C_{\sigma}) = \frac{Pr^{\omega}(C_{\sigma a \mathcal{I} s'} \cap C_{\sigma})}{Pr^{\omega}(C_{\sigma})} = D(\sigma, \{a\}) \cdot \mathbf{R}(s, a, s') \int_{\mathcal{I}} e^{-\mathbf{R}(s,a,S) \cdot t} dt \ . \quad (7)$$

For continuous systems like CTMDPs the numerator will usually be zero, so we resort to the continuous version of conditional probability the *Radon-Nikodym derivative* [16]. For this, some definitions are needed.

Definition 8. *The* marginal *of Pr^{ω} for the first n steps or coordinates is $Pr^{n} \in$ Distr$(Path^{n})$ where $Pr^{n}(\Lambda) \doteq Pr^{\omega}(\Lambda \times (L \times \mathbb{R}^{+} \times S)^{\omega}) = Pr^{\omega}(C_{\Lambda})$ and $|\Lambda| = n$.*

Definition 9. *Given two probability measures $\mu, \nu \in$ Distr(X), we say that μ* is absolutely continuous with respect to *ν, notation $\mu \ll \nu$, if $\forall A \in \mathcal{F}_{X}$, $\nu(A) = 0 \Rightarrow \mu(A) = 0$.*

It is straightforward to see $Pr^{n+1}(\cdot \times M) \ll Pr^{n}$, $\forall M \in \mathcal{F}_{L \times \mathbb{R}^{+} \times S}$, and this is the condition to apply Radon-Nikodym [14, Theorem 2.2.1] to obtain *(continuous) conditional probability* $f^{n} : Path^{n} \times \mathcal{F}_{L \times \mathbb{R}^{+} \times S} \to [0, 1]$, such that $\forall \Lambda \in \mathcal{F}_{Path^{n}}, M \in \mathcal{F}_{L \times \mathbb{R}^{+} \times S}$:

1. $f^{n}(\cdot, M) : Path^{n} \to [0, 1]$ is measurable,
2. $Pr^{n+1}(\Lambda \times M) = \int_{\Lambda} f^{n}(\sigma, M) dPr^{n}(\sigma)$.

Note that conditional probability is defined up-to sets of Pr^{n}-measure 0, therefore there are (potentially) infinite *versions* of it. This object shares with transition probability the measurability in its first argument. However if the measurable space is not restricted conveniently, it may be the case that from all versions of the conditional probability, none of them is a probability measure in its second argument [17, Section 6.4] [18, Problem 33.13]. Note this would be inconvenient for our purposes since equation (7) extends straightforwardly to a probability measure for all timed paths σ.

In our particular measure space (product of discrete spaces and positive reals), transition probabilities exists among conditional probabilities, and according to [13,19] we use *disintegration.* Namely Pr^{ω} can be disintegrated into transition probabilities, and this is where the compatibility with CTMDP can be defined.

Lemma 6 (Disintegration). *Given a probability measure $Pr^{\omega} \in$ Distr$(Path^{\omega})$, then there is a transition probability $\mu : Path^{*} \times \mathcal{F}_{L \times \mathbb{R}^{+} \times S} \to [0, 1]$ unique Pr^{ω}-almost everywhere, that generates this probability measure.*

Proof. Using the same arguments of conditional probability we have for each $M \in \mathcal{F}_{L \times \mathbb{R}^{+} \times S}$ there is a measurable function $\mu^{n}(\cdot, M) : Path^{n} \to [0, 1]$ such that $\forall \Lambda \in \mathcal{F}_{Path^{n}}$:

$$Pr^{n+1}(\Lambda \times M) = \int_{\Lambda} \mu^{n}(\sigma, M) dPr^{n}(\sigma) \quad (8)$$

From all the versions μ^n has, there is transition probability if the underlying space is *analytic* [20, Theorem 2.2] (the existence argument is over *regular conditional probabilities* but this problem is equivalent to transition probabilities [19, Theorem 3.1]). Examples of analytic spaces includes the discrete ones and the reals [9], and adding the fact that analytic spaces are closed under finite and denumerable product, we conclude $\mathcal{F}_{Path^n}$ is a space where we can choose a particular version of μ^n being a transition probability. The union $\cup_{0\leq n}\mu^n$ is denoted as $\mu : Path^* \times \mathcal{F}_{L\times\mathbb{R}^+\times S} \to [0,1]$ and gives the transition dynamics of the CTMDP.

Finally using $Pr^0 \in \mathrm{Distr}(S)$ and μ, by [14, Theorem 2.6.7] expression (8) extends uniquely to the marginals Pr^n. As $Pr^\omega(C_\Lambda) = Pr^n(\Lambda)$, by [14, Theorem 2.7.2] Pr^ω is the unique extension of this marginals to the infinite timed path probability measure, concluding that Pr^0 and μ generate Pr^ω. □

Having the transition probabilities underlying Pr^ω, we can state precisely what *compatible* means.

Definition 10 (Compatibility). *Given a CTMDP $\mathcal{C}$, we say probability measure Pr^ω on $\mathcal{F}_{Path^\omega}$ is* compatible *with $\mathcal{C}$ if: i) $Pr^0 = \mathcal{C}_{\mathrm{Pr}_{init}}$, ii) there is some scheduler D and transition probability μ from Lemma 6 that satisfies:*

$$\mu(\sigma, \{a\} \times \mathcal{I} \times \{s'\}) = D(\sigma, \{a\}) \cdot \mathbf{R}(last(\sigma), a, s') \int_{\mathcal{I}} e^{-\mathbf{R}(last(\sigma),a,S)\cdot t} dt \ . \quad (9)$$

It is clear that Definition 4 satisfies (9), and as both are additive in its second argument's first and third component, it is easy to show that $\mu(\sigma, A \times \mathcal{I} \times Q) = \mu_D(\sigma, A \times \mathcal{I} \times Q)$. Using Lemma 1, the main theorem follows.

Theorem 3 (Completeness). *Given a probability measure Pr^ω over $\mathcal{F}_{Path^\omega}$ compatible with CTMDP $\mathcal{C}$, then the underlying transition probability is generated by some measurable scheduler.*

Following the discussion of Section 5 it can be observed that given a positive probability measure Pr^ω for, e. g., C_Λ with $|\Lambda| \geq 2$, yields a friendly scheduler that generates the underlying transition probability. In contrast, when Pr^ω is almost always zero in the sense of Definition 6 the obtained scheduler falls in the class of AAMZ schedulers.

7 Conclusion

This work studies CTMDPs where external nondeterminism is resolved through timed history-dependent randomized schedulers. Taking time into account is the key difference that makes continuity and its related problems unavoidable, and this is where measurability issues arise.

We have obtained a complete characterization of those measurable schedulers, first using them to construct (integrate) a probability measure on timed paths, and later to deconstruct (disintegrate) an arbitrary probability measure on timed paths compatible with the CTMDP into a measurable scheduler. We showed that friendly schedulers generate positive probability measures and, vice versa, deconstructing positive measures yields friendly schedulers. We also showed an easily definable and probabilistically complete subclass of measurable schedulers.

As mentioned in Section 2, most of the results presented in this paper, however, do also carry over to less restrictive systems, e. g., stochastic transition systems [9]. We leave the consideration of a similar characterization of schedulers and the precise adaption of definitions and theorems to future work.

Acknowledgements. We would like to thank Holger Hermanns and Pedro D'Argenio for their inspiring ideas and helpful comments. We are also indebted to the anonymous referees for their valuable comments.

References

1. Bertsekas, D.P.: Dynamic Programming and Optimal Control. Volume II. Athena Scientific (1995)
2. Feinberg, E.A.: Continuous Time Discounted Jump Markov Decision Processes: A Discrete-Event Approach. Mathematics of Operations Research **29** (2004) 492–524
3. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. Wiley (1994)
4. Sennot, L.: Stochastic Dynamic Programming and the Control of Queueing Systems. John Wiley & Sons (1999)
5. Abdeddaïm, Y., Asarin, E., Maler, O.: On Optimal Scheduling under Uncertainty. TACAS (2003) 240–253
6. Bruno, J., Downey, P., Frederickson, G.N.: Sequencing Tasks with Exponential Service Times to Minimize the Expected Flow Time or Makespan. J. ACM **28** (1981) 100–113
7. Qiu, Q., Pedram., M.: Dynamic power managment based on continuous-time Markov decision processes. In: Proceedings DAC. (1999) 555 – 561
8. Baier, C., Haverkort, B., Hermanns, H., Katoen, J.P.: Efficient computation of time-bounded reachability probabilities in uniform continuous-time markov decision processes. In: TACAS: International Workshop on Tools and Algorithms for the Construction and Analysis of Systems, LNCS. (2004)
9. Cattani, S., Segala, R., Kwiatkowska, M.Z., Norman, G.: Stochastic Transition Systems for Continuous State Spaces and Non-determinism. In: FoSSaCS. (2005) 125–139
10. Kwiatkowska, M., Norman, G., Segala, R., Sproston, J.: Verifying quantitative properties of continuous probabilistic timed automata. In Palamidessi, C., ed.: Proc. CONCUR 2000 - Concurrency Theory. Volume 1877 of Lecture Notes in Computer Science., Springer (2000) 123–137
11. O.Hernndez-Lerma, Lassere, J.B.: Discrete-time Markov control processes: Basic optimality criteria. Volume 30 of Appl. Math. Springer-Verlag, Berlin and New York (1996)
12. K.Grabiszewski: Type space with disintegrability. Draft (2005)
13. Valadier, M.: Désintégration d'une mesure sur un produit. C.R. Acad. Sc. Paris **276** (1973) 33–35 Serie A.

14. Ash, R.B., Doléans-Dade, C.A.: Probability & Measure Theory. second edn. Academic Press (2000)
15. Shiryaev, A.N.: Probability. second edn. Springer (1995)
16. Panangaden, P.: Measure and probability for concurrency theorists. TCS: Theoretical Computer Science **253** (2001)
17. Panangaden, P.: Stochastic Techniques in Concurrency. Unpublished Lecture Notes from a course given at BRICS (1997)
18. Billingsley, P.: Probability and Measure. Wiley, New York, NY (1986) Second Edition.
19. Jr., D.L., Fragoso, M., Ruffino, P.: Regular conditional probability, disintegration of probability and radon spaces. Proyecciones **23** (2004) 15–29
20. Edalat, A.: Semi-pullbacks and bisimulation in categories of markov processes. Mathematical Structures in Computer Science **9** (1999) 523–543

Author Index

Aljazzar, Husain 33

Bérard, Béatrice 52, 67
Berthomieu, Bernard 82

Chandrasekaran, Prakash 98
Chevallier, Remy 113

David, Alexandre 128
Daws, Conrado 143
D'Souza, Deepak 322
Duguid, Adam 156

Encrenaz-Tiphène, Emmanuelle 113

Fainekos, Georgios E. 171
Finkel, Olivier 187
Frehse, Goran 200
Fribourg, Laurent 113
Furia, Carlo A. 215

Gastin, Paul 52, 67
Girard, Antoine 171

Haddad, Serge 230
Håkansson, John 128
Henzinger, Thomas A. 1

Johr, Sven 352

Kordy, Piotr 143

Laroussinie, François 245
Larsen, Kim G. 128
Lee, Edward A. 18
Leue, Stefan 33
Lodaya, Kamal 260

Maler, Oded 274
Markey, Nicolas 245
Mukund, Madhavan 98

Nickovic, Dejan 274
Niebert, Peter 290

Oreiby, Ghassan 245

Pace, Gordon 306
Pandya, Paritosh K. 260
Pappas, George J. 171
Peres, Florent 82
Petit, Antoine 52, 67
Pettersson, Paul 128
Pnueli, Amir 274
Prabhakar, Pavithra 322
Prabhu, Vinayak S. 1

Qu, Hongyang 290

Rabinovich, Alexander 32
Recalde, Laura 230
Rossi, Matteo 215

Schneider, Gerardo 306
Silva, Manuel 230

Vasilyev, Pavel 337
Vernadat, François 82

Wolovick, Nicolás 352

Xu, Weiwen 113