Lecture Notes in Computer Science 6246

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

Krishnendu Chatterjee
Thomas A. Henzinger (Eds.)

Formal Modeling and Analysis of Timed Systems

8th International Conference, FORMATS 2010
Klosterneuburg, Austria, September 8-10, 2010
Proceedings

Springer

Volume Editors

Krishnendu Chatterjee
Thomas A. Henzinger
Institute of Science and Technology
IST Austria
Am Campus 1, 3400 Klosterneuburg, Austria
E-mail: {krishnendu.chatterjee;tah}@ist.ac.at

Library of Congress Control Number: 2010932429

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

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

ISSN 0302-9743
ISBN-10 3-642-15296-1 Springer Berlin Heidelberg New York
ISBN-13 978-3-642-15296-2 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper 06/3180

Preface

This volume contains the papers that were presented at the 8th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS 2010), held September 8–10, 2010, at IST (Institute of Science and Technology) Austria, in Klosterneuburg, Austria.

The modeling and analysis of timing aspects of systems is a key problem that has been treated independently in several different communities in computer science and related areas. Researchers interested in semantics, verification, real-time scheduling, and performance analysis study models such as timed automata and timed Petri nets, the digital design community focuses on propagation and switching delays, and designers of embedded controllers need to take into account the time required by controllers to compute their responses after sampling the environment. Although the timing-related questions in these separate communities have their own specific nature, there is a growing awareness that there are basic problems that are common to all of them. In particular, all of these disciplines model and analyze systems whose behavior depends on combinations of logical and timing constraints 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 an interest in the modeling and analysis of timed systems. Typical topics include (but are not limited to):

- *Foundations and Semantics:* theoretical foundations of timed systems and languages; comparison between different models (timed automata, timed Petri nets, hybrid automata, timed process algebra, max-plus algebra, probabilistic models).
- *Methods and Tools:* techniques, algorithms, data structures, and software tools for analyzing timed systems and resolving timing constraints (scheduling, worst-case execution-time analysis, optimization, model checking, testing, synthesis, constraint solving).
- *Applications:* adaptation and specialization of timing technology in application domains in which timing plays an important role (real-time software, hardware circuits, and problems of scheduling in manufacturing and telecommunication).

This year FORMATS received 31 submissions. Each submission was reviewed by at least 3, and on average by 3.9, Program Committee members. The committee selected 14 submissions for presentation at the conference. In addition, the conference included three invited talks by:

- Tarek Abdelzaher, USA:
  Interdisciplinary Foundations for Open Cyber-Physical Systems
- Marta Kwiatkowska, UK:
  A Framework for Verification of Software with Time and Probabilities

– Jean-François Raskin, Belgium:
Safraless Procedures for Timed Specifications

The conference also included two invited tutorials by:

– Dejan Nickovic, Austria:
Property-Based Monitoring of Analog and Mixed-Signal Systems
– Ulrich Schmid, Austria:
Synchrony and Time in Fault-Tolerant Distributed Algorithms

We thank all invited speakers for accepting our invitation and for providing abstracts of their talks for inclusion in this proceedings volume.

We wish to thank all Program Committee members and reviewers for their competent and timely handling of the submissions. During the selection process and for preparing this volume, we used the EasyChair conference management system, which provided excellent support and allowed us to focus on the scientific issues. We thank the staff and scientists of IST Austria, in particular Barbara Abraham, Dejan Nickovic, and Franz Schäfer for their help in organizing the conference. Finally, we gratefully acknowledge the financial support we received from the European Network of Excellence on Embedded Systems Design (ArtistDesign), from IST Austria, and from Siemens Austria.

June 2010

Krishnendu Chatterjee
Thomas A. Henzinger

Conference Organization

Program Chairs

Krishnendu Chatterjee
Thomas A. Henzinger

Program Committee

Eugene Asarin
Christel Baier
Samarjit Chakraborty
Thao Dang
Laurent Doyen
Marco Faella
Ansgar Fehnker
Pavel Krcal
Kim G. Larsen
Chris Myers
Joel Ouaknine
Nir Piterman
André Platzer
Stefan Ratschan
Oleg Sokolsky
P.S. Thiagarajan
Enrico Vicario
Verena Wolf

Steering Committee

Rajeev Alur
Eugene Asarin
Joost-Pieter Katoen
Kim G. Larsen
Oded Maler
Lothar Thiele
Wang Yi

Local Organization

IST Austria, Klosterneuburg, Austria

External Reviewers

Alessandro Bianco
Anna Philippou
Arnd Hartmanns
David Renshaw
David Safranek
David Spieler
Erik Zawadzki
Ernst Moritz Hahn
Fabio Mogavero
Holger Hermanns
Jan Krcal
Jean-François Raskin
Jeremy Sproston
Johannes Faber
Kenneth Yrke Jørgensen
Laura Carnevali
Ligia Nistor
Lijun Zhang
Line Juhl
Lorenzo Ridi
Louis-Marie Traonouez
Margherita Napoli
Massimo Benerecetti
Mikael Harkjaer Moeller
Nicolas Markey
Nina Narodytska
Oded Maler
Olga Grinchtein
Paul Pettersson
Pontus Ekberg
Rob van Glabbeek
Roland Meyer
Romain Brenguier
Sarah Loos
Scott Cotton
Shaohui Wang
Stavros Tripakis
Stefano Minopoli
Sumit Kumar Jha
Thomas Brihaye
Tim Strazny
Timothy Bourke
Tomas Dzetkulic
Xiaoyue Pan

Table of Contents

Interdisciplinary Foundations for Open Cyber-Physical Systems
(Invited Talk)

Tarek Abdelzaher

Department of Computer Science, University of Illinois at Urbana-Champaign
zaher@illinois.edu

Abstract. A significant amount of literature addressed challenges in building dependable embedded systems. Next-generation cyber-physical systems offer more challenges that arise by virtue of openness, distribution, and scale. Emerging applications interact with both a physical and a social environment in both space and time. They are subjected to unpredictable workloads and are composed of a large number of subsystems of different degrees of criticality, not all of which are well-understood. Analyzing their overall behavior, diagnosing their interaction problems, and ensuring cooperation among their constituents requires new, often interdisciplinary tools and theoretical foundations not typically explored in classical embedded computing. This talk overviews some such tools and foundations, and outlines emerging broad challenges in building next-general open cyber-physical systems.

K. Chatterjee and T.A. Henzinger (Eds.): FORMATS 2010, LNCS 6246, p. 1, 2010.

Safraless Procedures for Timed Specifications*

Barbara Di Giampaolo[1], Gilles Geeraerts[2],
Jean-François Raskin[2], and Nathalie Sznajder[2]

[1] Dipartimento di Informatica ed Applicazioni, Università degli Studi di Salerno, Italy
[2] Département d'Informatique, Université Libre de Bruxelles (U.L.B.)
bardig@dia.unisa.it, {gigeerae,jraskin,nsznajde}@ulb.ac.be

Abstract. This paper presents extensions of Safraless algorithms proposed in the literature for automata on infinite untimed words to the case of automata on infinite timed words.

1 Introduction

In this paper, we investigate the applicability of automata constructions that avoid determinization for solving language inclusion and synthesis for real-time specifications. While timed language inclusion is undecidable for the class of timed languages definable by classical timed automata [AD94], there are interesting subclasses of timed languages for which language inclusion is decidable. In particular, it has been shown that the timed inclusion problem for event-clock automata [AFH99] and recursive generalizations [RS98, HRS98, Ras99] is PSPACE-complete for both finite and infinite words. For infinite words, those results are obtained using an adaptation of the Safra construction [Saf88] to this subclass of timed automata. Unfortunately, this construction leads to state spaces that are highly complex and difficult to handle in practice.

Contributions. Safra-based determinization is difficult to implement even in the context of untimed languages. As a consequence, recent research efforts have investigated alternative decision procedures [KV01, KV05, SF07, FJR09] that avoid the use of this construction. We investigate here extensions of those techniques to timed languages expressed by (alternating) event-clock automata and to a fragment of the Event-Clock Logic for which the realizability problem is decidable [DGRR09].

First, we show, in Section 3, that the techniques of [KV01] can be adapted to alternating event-clock automata. That is, given an alternating event-clock automaton with co-Büchi acceptance condition A, we show how to construct, in quadratic time, an alternating event-clock automaton with Büchi acceptance condition B that accepts the same language as A. From that alternating event-clock automaton B, we show how to construct in exponential time a nondeterministic event-clock automaton C with Büchi acceptance condition such that accepts the same language as B and A. This is done by adapting a

* Work supported by the projects: (i) Quasimodo: "Quantitative System Properties in Model-Driven-Design of Embedded", http://www.quasimodo.aau.dk/, (ii) Gasics: "Games for Analysis and Synthesis of Interactive Computational Systems", http://www.ulb.ac.be/di/gasics/, and (iii) Moves: "Fundamental Issues in Modelling, Verification and Evolution of Software", http://moves.ulb.ac.be, a PAI program funded by the Federal Belgian Gouvernment.

K. Chatterjee and T.A. Henzinger (Eds.): FORMATS 2010, LNCS 6246, pp. 2–22, 2010.

classical construction due to Miyano and Hayashi [MH84] originally proposed for Büchi automata on infinite (untimed) words. Those procedures then can be used to complement nondeterministic event-clock automata with Büchi acceptance conditions, this in turn leads to algorithms for solving the universality and language inclusion problems for that class of timed automata without resorting to the Safra construction.

Second, we generalize, in Section 4, the ideas of [FJR09] to solve the realizability problem for a fragment of the Event Clocks Logic called $\mathsf{LTL}_{\triangleleft}$ [DGRR09]. For each formula of this logic, we can construct, in exponential time, a universal event-clock automaton with co-Büchi acceptance condition that accepts the set of timed words that the formula defines. Then, we show that the co-Büchi acceptance condition can be strengthened into a condition that asks that all runs of the automaton visit less than $K \in \mathbb{N}$ times the set of accepting locations. This allows to reduce the realizability problem for $\mathsf{LTL}_{\triangleleft}$ to the realizability problem for universal K-co-Büchi event-clock automata. Those are easily determinizable and this reduces the original problem to a timed safety game problem. We show, in Section 5, that this timed safety game problem can be solved using the tool UPPAAL TIGA [BCD + 07]. We illustrate this on a simple example.

2 Preliminaries

Words and timed words. An alphabet Σ is a finite set of letters. A *finite (resp. infinite) word* w over an alphabet Σ is a finite (resp. infinite) sequence of letters from Σ. We denote respectively by Σ^* and Σ^ω the sets of all finite and infinite words on Σ. We denote by ε the empty word, and by $|w|$ the length the word w (which is equal to ∞ when w is infinite). A *finite (resp. infinite) timed word* over an alphabet Σ is a pair $\theta = (w, \tau)$ where w is a finite (resp. infinite) word over Σ, and $\tau = \tau_0\tau_1 \dots \tau_{|w|-1}$ is a finite (resp. infinite) sequence of length $|w|$ of positive real values (the time stamps) such that $\tau_i \leq \tau_{i+1}$ for all $0 \leq i \leq |w| - 1$ (resp. for all $i \geq 0$). We let $|(w, \tau)| = |w|$ denote the length of (w, τ).

An infinite timed word $\theta = (w, \tau)$ is *diverging* if for all $t \in \mathbb{R}^{\geq 0}$, there exists a position $i \in \mathbb{N}$ such that $\tau_i \geq t$. We denote respectively by $\mathsf{T}\Sigma^*$, $\mathsf{T}\Sigma^\omega$ and $\mathsf{T}\Sigma^\omega_{\mathrm{td}}$ the sets of all finite, infinite and infinite diverging timed words on Σ. In the sequel, it is often convenient to denote an (infinite) timed word (w, τ) by the sequence $(w_0, \tau_0)(w_1, \tau_1) \dots$ We proceed similarly for finite timed words. Since we are interested mainly in *infinite* timed words, we often refer to them simply as *timed words*.

Remark 1 (Time divergence). In the sequel, we formalize the results for languages of timed words that are not necessarily time divergent. Nevertheless, we systematically explain informally how to obtain the results for diverging timed words.

Event clocks. A *clock* is a real-valued variable whose value evolves with time elapsing. We associate, to every letter $\sigma \in \Sigma$, a *history* clock $\overleftarrow{x_\sigma}$ and a *prophecy* clock $\overrightarrow{x_\sigma}$. We denote respectively by $\mathbb{H}_\Sigma$ the set $\{\overleftarrow{x_\sigma} \mid \sigma \in \Sigma\}$ of *history clocks* and by $\mathbb{P}_\Sigma$ the set $\{\overrightarrow{x_\sigma} \mid \sigma \in \Sigma\}$ of *prophecy clocks* on Σ, and we let $\mathbb{C}_\Sigma = \mathbb{H}_\Sigma \cup \mathbb{P}_\Sigma$ be the set of *event-clocks* on Σ. A valuation v of a set of clocks $C \subseteq \mathbb{C}_\Sigma$ is a function $C \to \mathbb{R}^{\geq 0} \cup \{\bot\}$. We denote by $\mathcal{V}(C)$ the set of all valuations of the set of clocks C. We associate to each position $i \geq 0$ of a timed word $\theta = (w, \tau) \in \mathsf{T}\Sigma^\omega \cup \mathsf{T}\Sigma^*$ a unique valuation $\mathsf{Val}^\theta_{\mathsf{i}}$ of the clocks in $\mathbb{C}_\Sigma$, defined as follows. For any $x \in \mathbb{H}_\Sigma$, $\mathsf{Val}^\theta_{\mathsf{i}}(x) = \bot$ if there

is no $j < i$ s.t. $w_j = \sigma$. Otherwise, $\mathsf{Val}_{\mathsf{i}}^{\theta}(x) = \tau_i - \tau_j$ where j is the largest position s.t. $j < i$ and $w_j = \sigma$. Symmetrically, for any $x \in \mathbb{P}_{\Sigma}$, $\mathsf{Val}_{\mathsf{i}}^{\theta}(x) = \bot$ if there is no $j > i$ s.t. $w_j = \sigma$. Otherwise, $\mathsf{Val}_{\mathsf{i}}^{\theta}(x) = \tau_j - \tau_i$ where j is the least position s.t. $j > i$ and $w_j = \sigma$. Intuitively, this means that, when reading the timed word θ, the history clock $\overleftarrow{x_\sigma}$ always records the amount of time elapsed since the last occurrence of σ, and the prophecy clock $\overrightarrow{x_\sigma}$ always tells us the amount of time before the next occurrence of σ. For a valuation $v \in \mathcal{V}(C)$ such that $\forall x \in \mathbb{P}_{\Sigma} \cap C$: $v(x) \geq d$, we denote by $v + d$ the valuation from $\mathcal{V}(C)$ that respects the following two conditions. First, for any $x \in \mathbb{H}_{\Sigma} \cap C$: $(v + d)(x) = \bot$ if $v(x) = \bot$; otherwise $(v + d)(x) = v(x) + d$. Second, for any $x \in \mathbb{P}_{\Sigma} \cap C$: $(v + d)(x) = v(x) - d$ if $v(x) \neq \bot$; otherwise $(v + d)(x) = \bot$. For a valuation $v \in \mathcal{V}(C)$, and a clock $x \in C$, we write $v[x := 0]$ the valuation that matches v on every clock $x' \neq x$ and such that $v(x) = 0$.

An *atomic clock constraint* over the set of clocks C is either true or a formula of the form $x \sim c$, where $x \in C$, $c \in \mathbb{N}$, and $\sim \in \{<, >, =\}$. A *clock constraint* is a Boolean combination of atomic clock constraints. We denote by $\mathsf{Constr}(C)$ the set of all clock constraints ranging over the set of clocks C. We say that a valuation v satisfies a clock constraint ψ, denoted $v \models \psi$ according to the following rules: $v \models \mathsf{true}$; $v \models x \sim c$ iff $v(x) \neq \bot$ and $v(x) \sim c$; $v \models \neg\psi$ iff $v \not\models \psi$; $v \models \psi_1 \vee \psi_2$ iff $v \models \psi_1$ or $v \models \psi_2$. We say that a timed word θ satisfies a clock constraint ψ at position $i \geq 0$, denoted $(\theta, i) \models \psi$ iff $\mathsf{Val}_{\mathsf{i}}^{\theta} \models \psi$.

Alternating event clock automata. Let X be finite set. A *positive Boolean formula* over X is Boolean formula generated by:

$$\varphi ::= a \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \mathsf{true} \mid \mathsf{false}$$

with $a \in X$, and φ_1, φ_2 positive Boolean formulas. We denote by $\mathcal{B}^+(X)$ the set of all positive Boolean formulas on X. A set $Y \subseteq X$ *satisfies* a positive Boolean formula $\varphi \in \mathcal{B}^+(X)$, denoted $Y \models \varphi$ if and only if replacing each $y \in Y$ by true and each $x \in X \setminus Y$ by false in φ, and applying the standard interpretation for $\vee$ and $\wedge$ yields a formula which is equivalent to true. For example, $\varphi = (q_1 \wedge q_2) \vee q_3$ is a positive Boolean formula on $\{q_1, q_2, q_3\}$. Clearly, $\{q_1, q_2\} \models \varphi$, $\{q_2, q_3\} \models \varphi$, but $\{q_1\} \not\models \varphi$. Given a set X, and a positive Boolean formula $\varphi \in \mathcal{B}^+(X)$, we denote by $\tilde{\varphi}$ the *dual of* φ, which is the positive Boolean formula obtained from φ by swapping the $\vee$ and $\wedge$ operators, as well as the true and false values.

An *alternating event-clock automaton* (AECA) is a tuple $A = \langle Q, q_{in}, \Sigma, \delta, \alpha \rangle$, where Q is a finite set of locations, $q_{in} \in Q$ is the initial location, Σ is a finite alphabet, $\delta : Q \times \Sigma \times \mathsf{Constr}(\mathbb{C}_{\Sigma}) \mapsto \mathcal{B}^+(Q)$ is a partial function, and α is the acceptance condition, which can be:

1. either a *Büchi acceptance condition*; in this case, $\alpha \subseteq Q$,
2. or a *co-Büchi acceptance condition*; in this case, $\alpha \subseteq Q$,
3. or a *K-co-Büchi acceptance condition*, for some $K \in \mathbb{N}$; in this case, $\alpha \subseteq Q$,
4. or a *parity condition*; in this case, $\alpha : Q \mapsto \mathsf{Colours}$, where $\mathsf{Colours} \subseteq \mathbb{N}$ is a *finite* set of *priorities*.

Moreover, δ respects the following conditions:

(A_1) For every $q \in Q$, $\sigma \in \Sigma$, $\delta(q, \sigma, \psi)$ is defined for only finitely many ψ.

(A_2) For every $q \in Q$, $\sigma \in \Sigma$, $v \in \mathcal{V}(\mathbb{C}_{\Sigma})$ there exists one and only one $\psi \in \mathsf{Constr}(\mathbb{C}_{\Sigma})$ s.t. $v \models \psi$ and $\delta(q, \sigma, \psi)$ is defined.

Runs and accepted languages. Runs of AECA are formalised by *trees*. A *tree* T is a prefix closed set $T \subseteq \mathbb{N}^*$. The elements of T are called *nodes*, and the *root* of the tree is the empty sequence ε. For every $x \in T$, the nodes $x \cdot c \in T$, for $c \in \mathbb{N}$ are the *children* of x, and x is the (unique) *father* of all the nodes $x \cdot c$. A node with no child is a *leaf*. We refer to the length $|x|$ of x as its *level* in the tree. A *branch* in the tree T is a sequence of nodes $\pi \subseteq T$ such that $\varepsilon \in \pi$, and for every $x \in \pi$, either x is a leaf, or there is a unique $c \in \mathbb{N}$ such that $x \cdot c \in \pi$. An *X-labelled tree* is a pair $\langle T, \ell \rangle$ where $\ell : T \to X$ is a labelling function of the nodes, that associates a label from X to each node of T. We extend the function ℓ to (finite or infinite) branches: given a branch $\pi = n_1 n_2 \cdots n_j \cdots$ of T, we let $\ell(\pi)$ be the sequence $\ell(n_1)\ell(n_2)\cdots\ell(n_j)\cdots$ Let $A = \langle Q, q_{in}, \Sigma, \delta, \alpha \rangle$ be an AECA, and θ be an timed word on Σ. Then, a Q-labelled tree $R = \langle T, \ell \rangle$ is a *run of A on θ* iff the following hold:

- $\ell(\varepsilon) = q_{in}$,
- for all $x \in T$, there exists a set $S \subseteq Q$ s.t. (i) $q \in S$ iff x has a child $x \cdot c \in T$ with $\ell(x \cdot c) = q$ and (ii) $S \models \delta(\ell(x), w_{|x|}, \psi_x)$, where $\psi_x \in \mathsf{Constr}(\mathbb{C}_\Sigma)$ is the unique clock constraint s.t. $\delta(\ell(x), w_{|x|}, \psi_x)$ is defined and $(\theta, |x|) \models \psi_x$.

Let $R = \langle T, \ell \rangle$ be a run and $x \in T$. We note R_x the sub-run rooted at node x. A run $R = \langle T, \ell \rangle$ is *memoryless* if for all levels $i \in \mathbb{N}$, for all $x, y \in T$ such that $|x| = |y| = i$ and $\ell(x) = \ell(y)$, the sub-runs $R_x = \langle T_x, \ell_x \rangle$ and $R_y = \langle T_y, \ell_y \rangle$ are isomorphic.

Let $\langle T, \ell \rangle$ be an X-labelled tree, and let π be a branch of T. We let $\mathsf{Occ}_\pi : X \to \mathbb{N} \cup \{\infty\}$ be the function that associates, to any element of X, its number of occurrences in π. We further let $\mathsf{Inf}(\pi) = \{x \in X \mid \mathsf{Occ}_\pi(x) = \infty\}$. Let A be an AECA with set of locations Q and acceptance condition α, and $R = \langle T, \ell \rangle$ be a run of A. Then, R is an *accepting run* iff one of the following holds: α is a

- *Büchi condition*, and for all branches $\pi \subseteq T$, $\mathsf{Inf}(\pi) \cap \alpha \neq \varnothing$,
- *co-Büchi condition*, and for all branches $\pi \subseteq T$, $\mathsf{Inf}(\pi) \cap \alpha = \varnothing$,
- *K-co-Büchi condition*, and for all branches $\pi \subseteq T$, $\sum_{q \in \alpha} \mathsf{Occ}_\pi(q) \leq K$,
- *parity condition*, and for all branches $\pi \subseteq T$, $\max\{\alpha(q) \mid q \in \mathsf{Inf}(\pi)\}$ is even.

A timed word θ is *accepted* by an AECA A iff there exists an accepting run of A on θ. We denote by $\mathsf{L}(A)$ the *language* of A, i.e. $\mathsf{L}(A) = \{\theta \mid \theta \text{ is accepted by } A\}$, and by $\mathsf{L}(A)_{td}$ the time diverging language accepted by A, i.e. $\mathsf{L}(A)_{td} = \{\theta \mid \theta \in \mathsf{T}\Sigma^\omega_{\mathrm{td}} \text{ and } \theta \text{ is accepted by } A\}$.

For readability, we often refer to the language of an automaton A with *co-Büchi acceptance condition* as $\mathsf{L}_{\mathsf{coB}}(A)$. Similarly, we use $\mathsf{L}_{\mathsf{B}}(A)$ to denote the accepted language of an automaton A with *Büchi acceptance condition*, $\mathsf{L}_{\mathsf{KcoB}}(A)$ in the case of an automaton A with *K-co-Büchi acceptance condition*, and $\mathsf{L}_{\mathsf{P}}(A)$ for an automaton A with *parity acceptance condition*.

Finally, let $A = \langle Q, q_{in}, \Sigma, \delta, \alpha \rangle$ be an AECA with Büchi acceptance condition. The *dual of* A, denoted $\tilde{A}$ is defined as the AECA $\left\langle Q, q_{in}, \Sigma, \tilde{\delta}, \alpha \right\rangle$ with co-Büchi acceptance condition, where for any $q \in Q$, $\sigma \in \Sigma$ and $\psi \in \mathsf{Constr}(\mathbb{C}_\Sigma)$, $\tilde{\delta}(q, \sigma, \psi)$ is equal to $\widetilde{\delta(q, \sigma, \psi)}$ iff $\delta(q, \sigma, \psi)$ is defined. It is easy to check that $\mathsf{L}_{\mathsf{coB}}(\tilde{A}) = \mathsf{T}\Sigma^\omega \setminus \mathsf{L}_{\mathsf{B}}(A)$.

Remark 2 (Time divergence). It is easy to see that $\mathsf{L}_{\mathsf{coB}}(\tilde{A})_{td} = \mathsf{T}\Sigma^\omega_{\mathrm{td}} \setminus \mathsf{L}_{\mathsf{B}}(A)$.

Syntactic restrictions. Let us now define syntactic restrictions of AECA. Let $A = \langle Q, q_{in}, \Sigma, \delta, \alpha \rangle$ be an AECA. Then:

1. If, for any $q \in Q$, $\sigma \in \Sigma$ and $\psi \in \mathsf{Constr}(\mathbb{C}_\Sigma)$: $\delta(q, \sigma, \psi)$ is either undefined or a purely disjunctive formula, then A is a *non-deterministic event-clock automaton* (NECA for short).
2. If, for any $q \in Q$, $\sigma \in \Sigma$ and $\psi \in \mathsf{Constr}(\mathbb{C}_\Sigma)$: $\delta(q, \sigma, \psi)$ is either undefined or a purely conjunctive formula, then A is an *universal event-clock automaton* (UECA for short).
3. If, for any $q \in Q$, $\sigma \in \Sigma$ and $\psi \in \mathsf{Constr}(\mathbb{C}_\Sigma)$: $\delta(q, \sigma, \psi)$ is either undefined, or $\delta(q, \sigma, \psi) \in Q$, then A is a *deterministic event-clock automaton* (DECA for short).
4. If, for any $q \in Q$, $\sigma \in \Sigma$ and $\psi \in \mathsf{Constr}(\mathbb{C}_\Sigma)$: either $\delta(q, \sigma, \psi)$ is undefined or $\psi \in \mathsf{Constr}(\mathbb{H}_\Sigma)$, then A is a *past event-clock automaton* (PastECA for short).
5. If, for any $q \in Q$, $\sigma \in \Sigma$: $\delta(q, \sigma, \mathsf{true})$ is defined, then A is an *alternating word automaton* (AWA for short). In this case, since the third parameter of δ is always true, we omit it. We refer to such automata as *untimed* word automata. We use the shorthands NWA and DWA to refer to non-deterministic and deterministic (untimed) word automata.

Given a NECA A and a timed word θ on Σ, if there exists an accepting run $R = \langle T, \ell \rangle$ of A on θ, then it is easy to see that there exists an accepting run with one branch π. We denote such a run by the sequence $q_{in}, (\sigma_0, \tau_0), q_1, (\sigma_1, \tau_1), \cdots, q_j, (\sigma_j, \tau_j), \cdots$ where $q_{in} q_1 \cdots q_j \cdots$ is the label $\ell(\pi)$ of the single branch π of T.

Weak and strong equivalences for event-clock valuations. We define two notions of equivalence for valuations of clocks, the former called *weak equivalence* and the latter called *strong equivalence*. The notion of weak equivalence applies to valuations for both history clocks and prophecy clocks, while the notion of strong equivalence applies to valuations for history clocks only. They are defined as follows.

Let $C \subseteq \mathbb{H}_\Sigma \cup \mathbb{P}_\Sigma$, and let $cmax \in \mathbb{N}$. Two valuations $v_1, v_2 \in \mathcal{V}(C)$ are *weakly equivalent*, noted $v_1 \sim_{cmax} v_2$, iff the following two conditions are satisfied:

(C_1) $\forall x \in C$: $v_1(x) = \bot$ iff $v_2(x) = \bot$;
(C_2) $\forall x \in C$: either $v_1(x) > cmax$ and $v_1(x) > cmax$, or $\lceil v_1(x) \rceil = \lceil v_2(x) \rceil$ and $\lfloor v_1(x) \rfloor = \lfloor v_2(x) \rfloor$.

We note $[v]_{\sim cmax}$ the weak equivalence class of v. We note $\mathsf{wReg}(C, cmax)$ the finite set of equivalence classes of the relation $\sim_{cmax}$, and call them *weak regions*.

Lemma 3. *Let $C \subseteq \mathbb{H}_\Sigma \cup \mathbb{P}_\Sigma$, and let $cmax \in \mathbb{N}$. Two valuations $v_1, v_2 \in \mathcal{V}(C)$ are weakly equivalent iff for all $\psi \in \mathsf{Constr}(C, cmax)$: $v_1 \models \psi$ iff $v_2 \models \psi$.*

Let $C \subseteq \mathbb{H}_\Sigma$, and let $cmax \in \mathbb{N}$. Two valuations $v_1, v_2 \in \mathcal{V}(C)$ are *strongly equivalent*, noted $v_1 \approx_{cmax} v_2$, iff conditions C_1 and C_2 are satisfied and additionally:

(C_3) $\forall x_1, x_2 \in C$: $\lceil v_1(x_1) \rceil - v_1(x_1) \leq \lceil v_1(x_2) \rceil - v_1(x_2)$ iff $\lceil v_2(x_1) \rceil - v_2(x_1) \leq \lceil v_2(x_2) \rceil - v_2(x_2)$.

We note $[v]_{\approx cmax}$ the strong equivalence class of v, we note $\mathsf{Reg}(C, cmax)$ the finite set of equivalence classes of the relation $\approx_{cmax}$, and we call them *strong regions*, or simply *regions*. Note that our notion of strong equivalence for valuations

of history clocks is an adaptation of the classical notion of clock equivalence defined for timed automata [AD94], hence it is a time-abstract bisimulation. For any region $r \in \mathsf{Reg}(C, cmax)$, we say that $r' \in \mathsf{Reg}(C, cmax)$ is a *time-successor* of r (written $r \leq_{\mathsf{t.s.}} r'$) if and only if for any valuation $v \in r$, there is some $t \in \mathbb{R}^{\geq 0}$ such that $v + t \in r'$. Note that the relation $\leq_{\mathsf{t.s.}}$ is a partial order over $\mathsf{Reg}(C, cmax)$. A region $r \in \mathsf{Reg}(C, cmax)$ is *initial* if, for all $v \in r$, for all (history) clock $x \in C$: $v(x) = \bot$. Note that the initial region is unique and denoted r_{in}^C (when C is clear from the context we denote it by r_{in}). Finally, for all $r \in \mathsf{Reg}(C, cmax)$ and all $x \in C$, we note $r[x := 0]$ the region s.t. for all $v \in r[x := 0]$, there is $v' \in r$ with $v'[x := 0] = v$.

Region automaton. Given a set of history clocks $C \subseteq \mathbb{H}_\Sigma$ and $cmax \in \mathbb{N}$, the *region automaton* $\mathsf{RegAut}(C, cmax) = \langle \mathsf{Reg}(C, cmax) \cup \{\bot\}, r_{\mathrm{in}}^C, \Sigma^R, \delta^R, \alpha \rangle$, is a DWA where $\Sigma^R = \Sigma \times \mathsf{Reg}(C, cmax)$ and $\alpha = \mathsf{Reg}(C, cmax)$ is a Büchi acceptance condition. The transition relation δ^R is such that for all $r, r' \in \mathsf{Reg}(C, cmax)$, and for all $\sigma \in \Sigma$:

- $\delta^R(r, (\sigma, r')) = r'[\overleftarrow{x_\sigma} := 0]$ if $r \leq_{\mathsf{t.s.}} r'$, otherwise $\delta^R(r, (\sigma, r')) = \bot$,
- $\delta^R(\bot, (\sigma, r')) = \bot$.

Regionalizations of a timed word. Given $C \subseteq \mathbb{C}_\Sigma$, $cmax \in \mathbb{N}$, and a timed word $\theta = (\sigma_0, \tau_0)(\sigma_1, \tau_1) \cdots \in \mathsf{T}\Sigma^\omega \cup \mathsf{T}\Sigma^*$, let v_i be the restriction of $\mathsf{Val}_{\mathsf{i}}^\theta$ to the set of clocks C. We define the *weak region word associated to* θ, denoted $\mathsf{wrg}(C, cmax, \theta)$ as the (untimed) word $(\sigma_0, [v_0]_{\sim cmax})(\sigma_1, [v_1]_{\sim cmax}) \cdots$ over $\Sigma \times \mathsf{wReg}(C, cmax)$. Intuitively, $\mathsf{wrg}(C, cmax, \theta)$ describes, along with the sequence of letters, the sequence of weak regions visited by θ. If $C \subseteq \mathbb{H}_\Sigma$, we also define the *(strong) region word associated to* θ, denoted $\mathsf{rg}(C, cmax, \theta)$ as the (untimed) word $(\sigma_0, [v_0]_{\approx cmax})(\sigma_1, [v_1]_{\approx cmax}) \cdots$ over $\Sigma \times \mathsf{Reg}(C, cmax)$. We extend wrg and rg to set of words L: $\mathsf{wrg}(C, cmax, L) = \{\mathsf{wrg}(C, cmax, \theta) \mid \theta \in L\}$ and $\mathsf{rg}(C, cmax, L) = \{\mathsf{rg}(C, cmax, \theta) \mid \theta \in L\}$.

Proposition 4. *For all set of clocks $C \subseteq \mathbb{H}_\Sigma$ and $cmax \in \mathbb{N}$:* $\mathsf{L_B}(\mathsf{RegAut}(C, cmax)) = \mathsf{rg}(C, cmax, \mathsf{T}\Sigma^\omega)$.

Remark 5 (Time divergence). We can extend the definition of *region automaton* to obtain an automaton $\mathsf{RegAut}_{\mathsf{td}}(C, cmax)$ that accepts all the infinite words over $\Sigma \times \mathsf{Reg}(C, cmax)$ associated to *diverging* timed words. To achieve this, we must use a *generalized Büchi acceptance condition* that guarantees time divergence on the regions (see [AD94] for the details). Then, $\mathsf{L}(\mathsf{RegAut}_{\mathsf{td}}(C, cmax)) = \mathsf{rg}(C, cmax, \mathsf{T}\Sigma_{\mathrm{td}}^\omega)$.

Regionalizations of an AECA. Let $A = \langle Q, q_{in}, \Sigma, \delta, \alpha \rangle$ be an AECA, let $C \subseteq \mathbb{C}_\Sigma$ be the set of clocks and $cmax \in \mathbb{N}$ be the maximal constant appearing in A. We define the *weak regionalization of* A as the AWA $\mathsf{wRg}(A) = \langle Q, q_{in}, \Sigma \times \mathsf{wReg}(C, cmax), \delta', \alpha \rangle$ s.t. for all $q \in Q$, and $(\sigma, r) \in \Sigma \times \mathsf{wReg}(C, cmax)$: $\delta'(q, (\sigma, r)) = \delta(q, \sigma, \psi)$ where ψ is the unique constraint such that $\delta(q, \sigma, \psi)$ is defined and $v \models \psi$ for all $v \in r$.

Let $A = \langle Q, q_{in}, \Sigma, \delta, \alpha \rangle$ be a PastECA, let $C \subseteq \mathbb{H}_\Sigma$ be the set of history clocks and $cmax \in \mathbb{N}$ be the maximal constant appearing in A. We define the *(strong) regionalization of* A as the AWA $\mathsf{Rg}(A) = \langle Q, q_{in}, \Sigma \times \mathsf{Reg}(C, cmax), \delta', \alpha \rangle$ s.t. for all $q \in Q$, and $(\sigma, r) \in \Sigma \times \mathsf{Reg}(C, cmax)$: $\delta'(q, (\sigma, r)) = \delta(q, \sigma, \psi)$ where ψ is the unique constraint such that $\delta(q, \sigma, \psi)$ is defined and $v \models \psi$ for all $v \in r$.

The following lemma links runs in an AECA, and its weak and (strong) regionalization (when A is a PastECA).

Lemma 6. *Let A be an* AECA. *For every timed word $\theta \in \mathsf{T}\Sigma^\omega$, $R = \langle T, \ell \rangle$ is an accepting run tree of A over θ iff it is an accepting run tree of* $\mathsf{wRg}(A)$ *over* $\mathsf{wrg}(C, cmax, \theta)$. *Moreover, if A is a* PastECA, *$R = \langle T, \ell \rangle$ is an accepting run tree of A over θ iff it is an accepting run tree of* $\mathsf{Rg}(A)$ *over* $\mathsf{rg}(C, cmax, \theta)$.

The following lemma states that, for all PastECA A, the words accepted by both $\mathsf{Rg}(A)$ and by RegAut $(C, cmax)$ are exactly the (strong) regionalizations of the timed words accepted by A (whatever the acceptance condition of A is):

Lemma 7. *For all* PastECA $A = \langle Q, q_{in}, \Sigma, \delta, \alpha \rangle$, *with set of clocks $C \subseteq \mathbb{H}_\Sigma$ and maximal constant $cmax$:* $\mathsf{L}(\mathsf{Rg}(A)) \cap \mathsf{L}_\mathsf{B}(\mathsf{RegAut}\,(C, cmax)) = \mathsf{rg}(C, cmax, \mathsf{L}(A))$.

Proof. Let θ be a word in $\mathsf{L}(A)$. Then there is an accepting run $R = \langle T, \ell \rangle$ of A over θ. By Proposition 4, $\mathsf{rg}(C, cmax, \theta) \in \mathsf{L}_\mathsf{B}(\mathsf{RegAut}\,(C, cmax))$. By Lemma 6, R is also a accepting run of $\mathsf{Rg}(A)$ over $\mathsf{rg}(C, cmax, \theta)$. Thus, $\mathsf{rg}(C, cmax, \theta) \in \mathsf{L}(\mathsf{Rg}(A))$.

Conversely, let w be a word in $\mathsf{L}(\mathsf{Rg}(A)) \cap \mathsf{L}_\mathsf{B}(\mathsf{RegAut}\,(C, cmax))$. Since $w \in \mathsf{L}_\mathsf{B}(\mathsf{RegAut}\,(C, cmax))$, by Proposition 4, there is $\theta \in \mathsf{T}\Sigma^\omega$ such that $\mathsf{rg}(C, cmax, \theta) = w$. Let $R = \langle T, \ell \rangle$ be an accepting run of $\mathsf{Rg}(A)$ over w. By Lemma 6, R is also a accepting run of A over θ and thus $\theta \in \mathsf{L}(A)$. □

Remark 8 (Time divergence). If we restrict our attention to diverging timed words, then: $\mathsf{L}(\mathsf{Rg}(A)) \cap \mathsf{L}(\mathsf{RegAut}_\mathsf{td}\,(C, cmax)) = \mathsf{rg}(C, cmax, \mathsf{L}(A)_{td})$

3 Solving Language Inclusion without Determinization

In this section, we show how to complement AECA with Büchi acceptance condition. This procedure allows us to solve the universality and language inclusion problems for NECA with Büchi acceptance condition without resorting to determinization procedures (like the one defined by Safra in [Saf88]) that are resistant to efficient implementation.

We start by showing how to transform a co-Büchi acceptance condition into a Büchi condition when considering AECA. For that, we need the existence of memoryless runs:

Lemma 9. *Let A be an* AECA *with co-Büchi acceptance condition. For all timed words θ such that $\theta \in \mathsf{L}_\mathsf{coB}(A)$: A has an accepting memoryless run on θ.*

Proof. Let C be the set of clocks and $cmax$ be the maximal constant of A. Let θ be a timed word accepted by A. By Lemma 6, $\mathsf{wrg}(C, cmax, \theta)$ is accepted by the AWA $\mathsf{wRg}(C, cmax, A)$. Let $R = \langle T, \ell \rangle$ be an accepting run of $\mathsf{wRg}(C, cmax, A)$ on $\mathsf{wrg}(C, cmax, \theta)$. By the result of Emerson and Jutla [EJ91, Theorem 4.4], we can make the hypothesis that R is memoryless. By Lemma 6, R is an accepting run of A on θ. □

The memoryless property of accepting runs in AECA with co-Büchi acceptance condition allows us to represent those runs as DAGs where isomorphic subtrees are merged. Formally, we associate to every memoryless run $R = \langle T, \ell \rangle$ of $A = \langle Q, q_{in}, \Sigma, \delta, \alpha \rangle$ the DAG $G_R = \langle V, E \rangle$, where the set of vertices $V \subseteq Q \times \mathbb{N}$ represents the labels of the nodes of R at each level. Formally, $(q, l) \in V$ if and only if there is a node $x \in T$ such that $|x| = l$ and $\ell(x) = q$. The set of edges $E \subseteq \bigcup_{l \geq 0} (Q \times \{l\}) \times (Q \times \{l+1\})$ relates the nodes of one level to their children. Formally, $((q, l), (q', l+1)) \in E$ if and only if there exists some node $x \in T$ and $c \in \mathbb{N}$ such that $x \cdot c \in T$ and $|x| = l$, $\ell(x) = q$, and $\ell(x \cdot c) = q'$. Note that the width of the DAG is bounded by $|Q|$.

Now, we can apply results of [KV01] that characterize the structure of accepting runs of *alternating automata* with co-Büchi acceptance condition. For that we need some additional notations. For $k \in \mathbb{N}$ we write $[k]$ for the set $\{0, 1, \ldots, k\}$ and $[k]^{\text{odd}}$ for the set of odd elements of $[k]$. The following lemma is adapted from [KV01]:

Lemma 10. *Let A be an* AECA *with n locations and co-Büchi accepting condition α. The vertices of the DAG G_R associated to a memoryless accepting run R of A can be labelled by a ranking function $f : V \to [2n]$ having the following properties:*

(P_1) *for $(q, l) \in Q \times \mathbb{N}$, if $f(q, l)$ is odd, then $q \notin \alpha$,*
(P_2) *for (q, l) and (q', l') such that (q', l') is reachable from (q, l), $f(q', l') \leq f(q, l)$,*
(P_3) *in every infinite path π in G_R, there exists a node (q, l) such that $f(q, l)$ is odd and, for all (q', l') in π reachable from (q, l): $f(q', l') = f(q, l)$.*

We use this ranking function to justify the transformation of an AECA with co-Büchi acceptance condition into an AECA with Büchi acceptance condition.

Let $A = \langle Q, q_{in}, \Sigma, \delta, \alpha \rangle$ be an AECA with co-Büchi acceptance condition, and let $|Q| = n$. We define the AECA $\mathsf{Rank}(A)$ as $\langle Q', q'_{in}, \Sigma, \delta', \alpha' \rangle$ with Büchi acceptance condition, where $Q' = Q \times [2n]$, $q'_{in} = (q_{in}, 2n)$, and δ' is defined using the auxiliary function (we use the notations of [KV01]): $\mathsf{release} : \mathcal{B}^+(Q) \times [2n] \to \mathcal{B}^+(Q')$, which maps a formula $\phi \in \mathcal{B}^+(Q)$ and an integer $i \in [2n]$ to a formula obtained from ϕ by replacing each atom $q \in Q$ by the disjunction $\bigvee_{j \leq i}(q, j)$. Then, for any $(q, i) \in Q'$, $\sigma \in \Sigma$ and $\psi \in \mathsf{Constr}(\mathbb{C}_\Sigma)$ such that $\delta(q, \sigma, \psi)$ is defined,

$$\delta'((q, i), \sigma, \psi) = \begin{cases} \mathsf{release}(\delta(q, \sigma, \psi), i) & \text{if } q \notin \alpha \text{ or } i \text{ is even,} \\ \mathsf{false} & \text{if } q \in \alpha \text{ and } i \text{ is odd.} \end{cases}$$

Finally, $\alpha' = Q \times [2n]^{\text{odd}}$ is a Büchi acceptance condition.

Remark that, by condition A_2 of the definition of the transition relation in AECA, for all $q \in Q$, $\sigma \in \Sigma$ and valuation $v \in \mathcal{V}(C)$, there is exactly one clock constraint ψ such that $\delta(q, \sigma, \psi)$ is defined and $v \models \psi$. Thus, by construction of $\mathsf{Rank}(A)$, for all $q \in Q$, $i \in [n]$ and valuation $v \in \mathcal{V}(C)$, there is exactly one clock constraint ψ such that $\delta'((q, i), w_k, \psi)$ is defined and $v \models \psi$. Thus, δ' is well-formed. Let us establish the relationship between the accepted languages of A and $\mathsf{Rank}(A)$

Proposition 11. *For all* AECA *A with co-Büchi condition:* $\mathsf{L_B}(\mathsf{Rank}(A)) = \mathsf{L_{coB}}(A)$.

Proof. Let $\theta = (\sigma, \tau) \in \mathsf{T}\Sigma^\omega$ be a timed word in $\mathsf{L_B}(\mathsf{Rank}(A))$ and let us show that $\theta \in \mathsf{L_{coB}}(A)$. Let $R' = \langle T, \ell' \rangle$ be an accepting run of $\mathsf{Rank}(A)$ on θ. Consider $R = \langle T, \ell \rangle$ where for all $x \in T$, $\ell(x) = q$ if $\ell'(x) = (q, j)$ for some rank j. By definition of $\mathsf{Rank}(A)$, R is a run of A on θ. Let us now show that it is an accepting run of A. As R' is accepting for $\mathsf{Rank}(A)$, we know that every branch has the following property: from some level $i \in \mathbb{N}$, the rank j is not changing anymore. This is because the definition of the transition function of $\mathsf{Rank}(A)$ requires the ranks to decrease along a path, while staying positive. Moreover, the acceptance condition imposes that this rank is odd. Let π be such a branch. As accepting locations of A are associated to odd ranks and cannot appear in runs of $\mathsf{Rank}(A)$ (it is forbidden by the transition relation), we know that the branch π in R visits only finitely many accepting locations and so it respects the acceptance condition of A.

Conversely, let $\theta \in \mathsf{L}_{\mathsf{coB}}(A)$ and let us show that $\theta \in \mathsf{L}_{\mathsf{B}}(\mathsf{Rank}(A))$. Let $R = (T, \ell)$ be an accepting run of A on θ. Now consider the tree $R' = (T, \ell')$, where ℓ' is s.t. $\ell(\varepsilon) = (q_{in}, 2n)$ and for all $x \in T$, $\ell'(x) = (\ell(x), f(x))$. Following properties P_1 and P_2 of Lemma 10, $R' = (T, \ell')$ is a run of $\mathsf{Rank}(A)$ over the timed word θ. Let π be a branch of R'. Then, property P_3 in Lemma 10 ensures that at some point, all the states in π are labelled by the same odd rank. Thus, any branch of R' visits infinitely often a state in $Q \times [2n]^{\text{odd}}$, and $\mathsf{Rank}(A)$ is accepting. □

Next, we show that the construction due to Miyano and Hayashi [MH84] to transform an alternating Büchi automaton into a nondeterministic one can be easily adapted to AECA with Büchi acceptance condition. Formally, given an AECA with Büchi acceptance condition $A = \langle Q, q_{in}, \Sigma, \delta, \alpha \rangle$, we define a NECA $\mathsf{MH}(A)$ as follows. For any $\sigma \in \Sigma$, for any $q \in Q$, let $\Phi_q^\sigma = \{\psi \in \mathsf{Constr}(\mathbb{C}_\Sigma) \mid \delta(q, \sigma, \psi) \text{ is defined}\}$. By condition A_1 of the definition of an AECA, Φ_q^σ is finite. We also define, for any $\sigma \in \Sigma$, for any subset $S \subseteq Q$, the set of formulas $\Psi_S^\sigma = \{\bigwedge_{q \in S} \psi_q \mid \psi_q \in \Phi_q^\sigma\}$. Intuitively, Ψ_S^σ contains all the conjunctions that contain exactly one conjunct from each set Φ_q^σ (for $q \in S$). Finally, for $S \subseteq Q$, $O \subseteq Q$, $\sigma \in \Sigma$, $\psi = \bigwedge_{q \in S} \psi_q \in \Psi_S^\sigma$, we let $P(S, O) = \{(S', O') \mid S' \models \bigwedge_{q \in S} \delta(q, \sigma, \psi_q), O' \subseteq S', O' \models \bigwedge_{q \in O} \delta(q, \sigma, \psi_q)\}$ if $O \neq \varnothing$, and $P(S, \varnothing) = \{(S', S') \mid S' \models \bigwedge_{q \in S} \delta(q, \sigma, \psi_q)\}$.

Then, we define $\mathsf{MH}(A)$ as the AECA $\left\langle 2^Q \times 2^Q, (\{q_{in}\}, \varnothing), \Sigma, \delta', 2^Q \times \{\varnothing\} \right\rangle$ with Büchi acceptance condition where, for any $(S, O) \in 2^Q \times 2^Q$, for any $\sigma \in \Sigma$, for any $\psi \in \Psi_S^\sigma$: $\delta'((S, O), \sigma, \psi) = \bigvee_{(S', O') \in P(S, O)} (S', O' \setminus \alpha)$ (and δ' is undefined otherwise). Remark that, by conditions A_1 and A_2, Ψ_S^σ is a finite set, and for any valuation v, there is exactly one $\psi \in \Psi_S^\sigma$ s.t. $v \models \psi$. Hence, δ' respects the definition of the transition relation of an AECA. The next proposition proves the correctness of the construction.

Proposition 12. *For all* AECA A *with Büchi condition:* $\mathsf{L}_{\mathsf{B}}(\mathsf{MH}(A)) = \mathsf{L}_{\mathsf{B}}(A)$.

Proof. Assume $A = \langle Q, q_{in}, \Sigma, \delta, \alpha \rangle$. Let θ be a timed word in $\mathsf{L}_{\mathsf{B}}(A)$ and $R = \langle T, \ell \rangle$ be an accepting run of A over θ. Then, let $\rho = (\{q_{in}\}, \varnothing), (\sigma_0, \tau_0), (S_1, O_1), (\sigma_1, \tau_1), \cdots$ be the sequence such that, for all $i \in \mathbb{N}$: (i) $S_i = \{q \mid \exists x \in T, |x| = i, \ell(x) = q\}$ and (ii) $O_i = S_i \setminus \alpha$ if $O_{i-1} = \varnothing$; $O_i = \{q \mid \exists x \cdot c \in T, |x \cdot c| = i, \ell(x \cdot c) = q, \ell(x) \in O_{i-1}\} \cap (Q \setminus \alpha)$ otherwise (with the convention that $O_0 = \varnothing$). It is easy to see that, as in the original construction of [MH84], ρ is an accepting run of $\mathsf{MH}(A)$ over θ.

Conversely, given a run $(\{q_{in}\}, \varnothing)(\sigma_0, \tau_0)(S_1, O_1)(\sigma_1, \tau_1)(S_2, O_2) \cdots$ of $\mathsf{MH}(A)$, we consider a labelled tree $\langle T, \ell \rangle$ s.t. (i) $\ell(\varepsilon) = q_{in}$ and (ii) for any $x \in T$: $\{\ell(x \cdot i) \mid i \in \mathbb{N}\} \subseteq S_{|x|+1}$ and $\{\ell(x \cdot i) \mid i \in \mathbb{N}\} \models \delta(\ell(x), \sigma_{|x|}, \psi)$, where ψ is the unique constraint s.t. $\delta(\ell(x), \sigma_{|x|}, \psi)$ is defined and $(\theta, |x|) \models \psi$. Clearly, R is an accepting run tree of A over θ. □

Applications. Let us show how to apply these constructions to complement an NECA. Given a NECA $A = \langle Q, q_{in}, \Sigma, \delta, \alpha \rangle$ with Büchi acceptance condition, we first construct its dual $\widetilde{A}$ which is thus a UECA with co-Büchi acceptance condition s.t. $\mathsf{L}_{\mathsf{coB}}(\widetilde{A}) = \mathsf{T}\Sigma^\omega \setminus \mathsf{L}_{\mathsf{B}}(A)$. Then, thanks to Proposition 11 and Proposition 12, it easy to check that $\mathsf{MH}(\mathsf{Rank}(\widetilde{A}))$ is a NECA with Büchi condition s.t. $\mathsf{L}_{\mathsf{B}}(\mathsf{MH}(\mathsf{Rank}(\widetilde{A}))) = \mathsf{T}\Sigma^\omega \setminus \mathsf{L}_{\mathsf{B}}(A)$.

This construction can be applied to solve the *language inclusion* and *language universality* problems, because $\mathsf{L}_{\mathsf{B}}(A)$ is universal iff $\mathsf{T}\Sigma^\omega \setminus \mathsf{L}_{\mathsf{B}}(A)$ is empty and $\mathsf{L}_{\mathsf{B}}(B) \subseteq \mathsf{L}_{\mathsf{B}}(A)$ iff $\mathsf{L}_{\mathsf{B}}(B) \cap (\mathsf{T}\Sigma^\omega \setminus \mathsf{L}_{\mathsf{B}}(A))$ is empty.

Remark 13 (Time divergence). All the constructions presented above are valid if we consider the time divergent semantics. Indeed, $\mathsf{L}(A)_{td} \subseteq \mathsf{L}(B)_{td}$ if and only if $(\mathsf{L}(A) \cap \mathsf{L}(B)) \cap \mathsf{T}\Sigma^{\omega}_{\mathrm{td}} = \varnothing$

Remark 14 (Efficient implementation). In [DR10], it is shown how to use subsumption to implement efficient emptiness test for automata defined by the Miyano and Hayashi construction without explicitly constructing them. Those methods can be readily extended to the case of event-clock automata.

4 Safraless Algorithm for Realizability

In this section, we study the *realizability problem* for timed specifications expressed by UECA. We restrict to event-clock automata with history clocks only as the use of prophecy clocks leads to undecidability [DGRR09]. To formalize the *realizability problem* in this context, we rely on the notion of *timed game*.

Timed games. A *timed game* (TG for short) is a tuple $\langle \Sigma_1, \Sigma_2, W \rangle$ where Σ_i $(i = 1, 2)$ is a finite alphabet for player i (with $\Sigma_1 \cap \Sigma_2 = \varnothing$), and $W \subseteq \mathsf{T}\Sigma^{\omega}$ is a set of timed words, called the *objective* of the game (for player 1).

A TG is played for infinitely many rounds. At each round i, player 1 first chooses a delay t^1_i and a letter $\sigma^1_i \in \Sigma_1$. Then, player 2 chooses either to pass or to overtake player 1 with a delay $t^2_i \leq t^1_i$ and a letter $\sigma^2_i \in \Sigma_2$. A *play* in a timed game is a timed word (w, τ) s.t. for any $i \geq 0$ either (i) player 2 has passed at round i, $w_i = \sigma^1_i$ and $\tau_i = \tau_{i-1} + t^1_i$, or (ii) player 2 has overtaken player 1 at round i, $w_i = \sigma^2_i$ and $\tau_i = \tau_{i-1} + t^2_i$ (with the convention that $\tau_{-1} = 0$). A timed word θ is *winning* in $\langle \Sigma_1, \Sigma_2, W \rangle$ iff $\theta \in W$. A *strategy* for player 1 is a function π that associates to every finite prefix of a timed word $(w_0, \tau_0) \dots (w_k, \tau_k)$ an element from $\Sigma_1 \times \mathbb{R}^{\geq 0}$. A play $\theta = (w, \tau)$ is consistent with strategy π for player 1 iff for every $i \geq 0$, *either* player 1 has played according to its strategy i.e., $(w_i, \tau_i - \tau_{i-1}) = \pi((w_0, \tau_0) \dots (w_{i-1}, \tau_{i-1}))$ *or* player 2 has overtaken the strategy of player 1 i.e., $w_i \in \Sigma_2$, and $\pi((w_0, \tau_0) \dots (w_{i-1}, \tau_{i-1})) = (\sigma, \tau)$ with $\tau \geq \tau_i - \tau_{i-1}$. The *outcome* of a strategy π in a game $G = \langle \Sigma_1, \Sigma_2, W \rangle$, noted $\mathsf{Outcome}\,(G, \pi)$ is the set of all plays of G that are consistent with π. A strategy π is *winning* iff $\mathsf{Outcome}\,(G, \pi) \subseteq W$.

The *realizability problem* asks, given a universal PastECA A with co-Büchi acceptance condition, whose alphabet Σ is partitioned into Σ_1 and Σ_2, if player 1 has a winning strategy in $G = \langle \Sigma_1, \Sigma_2, \mathsf{L}_{\mathsf{coB}}(A) \rangle$.

To solve this problem without using Safra determinization, we show how to reduce it to a timed safety objective via a strengthening of the winning objective using K-co-Büchi acceptance condition. We state the main result of this section:

Theorem 15. *Given a universal* PastECA *A with co-Büchi acceptance condition, whose alphabet Σ is partitioned into Σ_1 and Σ_2, player 1 has a winning strategy in $G^T = \langle \Sigma_1, \Sigma_2, \mathsf{L}_{\mathsf{coB}}(A) \rangle$ iff he has a winning strategy in $G^T_K = \langle \Sigma_1, \Sigma_2, \mathsf{L}_{\mathsf{KcoB}}(A) \rangle$, for any $K \geq (2n^{n+1}n! + n) \times |\mathsf{Reg}\,(\mathbb{H}_{\Sigma}, cmax)|$ where n is the number of locations in A.*

To establish this result, we use several intermediary steps. First, we show that we can associate a game with an ω-regular objective, played on untimed words, to any timed game whose objective is defined by a UECA with co-Büchi acceptance condition.

Region games. A *region game* is a tuple $G^R = \langle \Sigma_1, \Sigma_2, cmax, W\rangle$ where $\Sigma = \Sigma_1 \cup \Sigma_2$, $\Sigma_1 \cap \Sigma_2 = \varnothing$, $cmax \in \mathbb{N}$, W is a set of infinite words on the alphabet $(\Sigma_1 \uplus \Sigma_2) \times \mathsf{Reg}\,(\mathbb{H}_\Sigma, cmax)$, called the *objective* of the game (for player 1).

A *play* of a region game is an infinite (untimed) word on the alphabet $(\Sigma_1 \cup \Sigma_2) \times \mathsf{Reg}\,(\mathbb{H}_\Sigma, cmax)$. The game is played for infinitely many rounds. In the initial round, player 1 first chooses a letter $\sigma^1 \in \Sigma_1$. Then, either player 2 lets player 1 play and the first letter of the play is $(\sigma^1, r_{\text{in}})$, or player 2 overtakes player 1 with a letter $\sigma^2 \in \Sigma_2$ and the first letter of the play is $(\sigma^2, r_{\text{in}})$. In all the subsequent rounds, and assuming that the prefix of the current play is $(\sigma_0, r_0), \cdots (\sigma_k, r_k)$, player 1 first chooses a pair $(\sigma^1, r^1) \in \Sigma_1 \times \mathsf{Reg}\,(\mathbb{H}_\Sigma, cmax)$ such that $r_k[\overleftarrow{x_{\sigma_k}} := 0] \leq_{\text{t.s.}} r^1$. Then, either player 2 lets player 1 play and the new prefix of the play is $\rho_{k+1} = \rho_k \cdot (\sigma^1, r^1)$, or player 2 decides to overtake player 1 with a pair $(\sigma^2, r^2) \in \Sigma_2 \times \mathsf{Reg}\,(\mathbb{H}_\Sigma, cmax)$, respecting $r_k[\overleftarrow{x_{\sigma_k}} := 0] \leq_{\text{t.s.}} r^2 \leq_{\text{t.s.}} r^1$. In this case, the new prefix of the play is $\rho_{k+1} = \rho_k \cdot (\sigma^2, r^2)$. A play ρ is *winning* in $\langle \Sigma_1, \Sigma_2, cmax, W\rangle$ iff $\rho \in W$. As for timed games, a *strategy* for player 1 is a function π^R that associates to every finite prefix $(w_0, r_0) \ldots (w_k, r_k)$ an element $(\sigma, r) \in \Sigma_1 \times \mathsf{Reg}\,(\mathbb{H}_\Sigma, cmax)$ such that $r_k[\overleftarrow{x_{w_k}} := 0] \leq_{\text{t.s.}} r$. A play $\rho = (\sigma_0, r_{\text{in}})(\sigma_1, r_1) \cdots$ is consistent with strategy π for player 1 iff for all $i \geq 0$, *either* player 1 has played according to its strategy, i.e., $(\sigma_i, r_i) = \pi((\sigma_0, r_{\text{in}}) \ldots (\sigma_{i-1}, r_{i-1}))$ (with the convention that $(\sigma_{-1}, r_{-1}) = \varepsilon$), *or* player 2 has overtaken the strategy of player 1 i.e., $\sigma_i \in \Sigma_2$, $\pi((\sigma_0, r_{\text{in}}) \ldots (\sigma_{i-1}, r_{i-1})) = (\sigma, r)$ and $r_i \leq_{\text{t.s.}} r$.

Remark 16. All plays of $\langle \Sigma_1, \Sigma_2, cmax, W\rangle$ are in $\mathsf{L_B}(\mathsf{RegAut}\,(\mathbb{H}_\Sigma, cmax))$.

The *outcome* $\mathsf{Outcome}\,(G, \pi)$ of a strategy π on a region game G and winning strategies are defined as usual. The next proposition shows how a timed game can be reduced to a region game.

Proposition 17. *Let A be a universal* PastECA *with maximal constant $cmax$. Player 1 has a winning strategy in the timed game $G^T = \langle \Sigma_1, \Sigma_2, \mathsf{L_{coB}}(A)\rangle$ iff he has a winning strategy in the region game $G^R = \langle \Sigma_1, \Sigma_2, cmax, \mathsf{L_{coB}}(\mathsf{Rg}(A))\rangle$. Moreover, for any $K \in \mathbb{N}$, player 1 has a winning strategy in $G^T = \langle \Sigma_1, \Sigma_2, \mathsf{L_{KcoB}}(A)\rangle$ iff he has a winning strategy in $G^R = \langle \Sigma_1, \Sigma_2, cmax, \mathsf{L_{KcoB}}(\mathsf{Rg}(A))\rangle$.*

Proposition 17 tells us that we can reduce the realizability problem of timed games to that of *region games*. Next we show that *region games* can be won thanks to a *finite memory strategy*. For that, we expose a reduction from region games to *parity games*.

Parity games. A *parity game* is a tuple $G = \langle Q, E, q_0, \mathsf{Colours}, \lambda\rangle$ where $Q = Q_1 \uplus Q_2$ is the set of *positions*, partitioned into the player 1 and player 2 positions, $E \subseteq Q \times Q$ is the set of *edges*, $q_0 \in Q$ is the initial position, and $\lambda : Q \mapsto \mathsf{Colours}$ is the *coloring function*.

A *play* of a parity game $G = \langle Q, E, q_0, \mathsf{Colours}, \lambda\rangle$ is an infinite sequence $\rho = q_0 q_1 \cdots q_j \cdots$ of positions s.t. for any $j \geq 0$: $(q_j, q_{j+1}) \in E$. Given a play $\rho = q_0 q_1 \cdots q_j \cdots$, we denote by $\mathsf{Inf}\,(\rho)$ the set of positions that appear infinitely often in ρ, and by $\mathsf{Par}\,(\rho)$ the value $\max\{\lambda(q) \mid q \in \mathsf{Inf}\,(\rho)\}$. A play ρ is *winning* for player 1 iff $\mathsf{Par}\,(\rho)$ is *even*. A *strategy* for player 1 in G is a function $\pi : Q^* Q_1 \to Q$ that associates, to each finite prefix ρ of play ending in a Player 1 state $\mathsf{Last}(\rho)$, a successor position $\pi(\rho)$

s.t. $(\mathsf{Last}(\rho), \pi(\rho)) \in E$. Given a parity game G and a strategy π for player 1, we say that a play $\rho = q_0 q_1 \cdots q_j \cdots$ of G *is consistent with* π iff for $j \geq 0$: $q_j \in Q_1$ implies that $q_{j+1} = \pi(q_0 \cdots q_j)$. We denote by $\mathsf{Outcome}\,(G, \pi)$ the set of plays that are consistent with π. A strategy π is *winning* iff every play $\rho \in \mathsf{Outcome}\,(G, \pi)$ is winning.

It is well-known that parity games admit *memoryless strategies*. More precisely, if there exists a winning strategy for player 1 in a parity game G, then there exists a winning strategy π for player 1 s.t. for any pair of prefixes ρ and ρ': $\mathsf{Last}(\rho) = \mathsf{Last}(\rho')$ implies $\pi(\rho) = \pi(\rho')$. A memoryless strategy π can thus be finitely represented by a function $f_\pi : Q_1 \to Q$, where, for any $q \in Q_1$, $f_\pi(q)$ is the (unique) position q' s.t. for any prefix $\rho = q_0 \cdots q$, $\pi(\rho) = q'$. In the sequel we often abuse notations and confuse f_π with π when dealing with memoryless strategies in parity games.

Let us show how to reduce the region game $\langle \Sigma_1, \Sigma_2, cmax, \mathsf{L_{coB}}(\mathsf{Rg}(A))\rangle$ to a parity game. First consider the NWA $\widetilde{\mathsf{Rg}(A)}$ that dualizes $\mathsf{Rg}(A)$ and such that $\mathsf{L_B}(\widetilde{\mathsf{Rg}(A)}) = \Sigma^\omega \setminus \mathsf{L_{coB}}(\mathsf{Rg}(A))$. Then, using Piterman's construction [Pit07], we can obtain a deterministic parity automaton $\widetilde{D}$ such that $\mathsf{L_P}(\widetilde{D}) = \mathsf{L_B}(\widetilde{\mathsf{Rg}(A)})$, and by complementing $\widetilde{D}$, we obtain a deterministic (and complete) parity automaton D such that $\mathsf{L_P}(D) = \mathsf{L_{coB}}(\mathsf{Rg}(A))$. We use this automaton and the region automaton $\mathsf{RegAut}\,(\mathbb{H}_\Sigma, cmax)$ as a basis for the construction of the parity game.

A play in the parity game simulates runs over words in $(\Sigma \times \mathsf{Reg}\,(\mathbb{H}_\Sigma, cmax))^\omega$ of both $D = \langle Q^D, q^D_{in}, \Sigma \times \mathsf{Reg}\,(\mathbb{H}_\Sigma, cmax), \delta^D, \alpha^D\rangle$ and $\mathsf{RegAut}\,(\mathbb{H}_\Sigma, cmax) = \langle Q^R, q^R_{in}, \Sigma^R, \delta^R, \alpha^R\rangle$. Formally, $G_D = \langle q^G_{in}, Q^G, E^G, \mathsf{Colours}, \lambda^G\rangle$, where the positions of player 1 are $Q^G_1 = (Q^D \times \mathsf{Reg}\,(\mathbb{H}_\Sigma, cmax))$, and the positions of player 2 are $Q^G_2 = (Q^D \times \mathsf{Reg}\,(\mathbb{H}_\Sigma, cmax)) \times (\Sigma_1 \times \mathsf{Reg}\,(\mathbb{H}_\Sigma, cmax))$. Intuitively, $(q, r) \in Q^G_1$ means that the simulated runs are currently in the states q and r of respectively D and $\mathsf{RegAut}\,(\mathbb{H}_\Sigma, cmax)$. From a position in Q^G_1, player 1 can go to a position memorizing the current states in D and $\mathsf{RegAut}\,(\mathbb{H}_\Sigma, cmax)$, as well as the next move according to player 1's strategy. Thus, $(q, r, \sigma^1, r^1) \in Q^G_2$ means that we are in the states q and r in the automata, and that (σ^1, r^1) is the letter proposed by player 1. Then, from (q, r, σ^1, r^1), player 2 chooses either to let player 1 play, or decides to overtake him. In the former case, the game moves a position (q', r') where q' and r' are the new states in D and $\mathsf{RegAut}\,(\mathbb{H}_\Sigma, cmax)$ after a transition on (σ^1, r^1). In the latter case (overtake player 1), the game moves to a position (q'', r''), assuming there are $\sigma^2 \in \Sigma_2$, $r^2 \leq_{\text{t.s.}} r^1$ such that q'' and r'' are the new states of D and $\mathsf{RegAut}\,(\mathbb{H}_\Sigma, cmax)$ after a transition on (σ^2, r^2). These moves are formalized by the set of edges $E^G = E^G_1 \uplus E^G_2$ where:

$$
\begin{aligned}
E^G_1 &= \{\big((q, r), (q, r, \sigma^1, r^1)\big) \mid \sigma^1 \in \Sigma_1, \delta^R(r, (\sigma^1, r^1)) \neq \bot\}\\
E^G_2 &= \{\big((q, r, \sigma^1, r^1), (q', r')\big) \mid (q', r') = (\delta^D(q, (\sigma^1, r^1)), \delta^R(r, (\sigma^1, r^1)))\}\\
&\quad \cup \left\{ \big((q, r, \sigma^1, r^1), (q', r')\big) \;\middle|\; \begin{array}{c} \exists \sigma^2 \in \Sigma_2, r^2 \leq_{\text{t.s.}} r^1, \delta^R(r, (\sigma^2, r^2)) \neq \bot, \mathit{and}\\ (q', r') = (\delta^D(q, (\sigma^2, r^2)), \delta^R(r, (\sigma^2, r^2))) \end{array} \right\}
\end{aligned}
$$

Intuitively, player 1 chooses its next letter in Σ_1 and a region. The definition of E^G_1 uses transitions of $\mathsf{RegAut}\,(\mathbb{H}_\Sigma, cmax)$ and hence enforces the fact that player 1 can only propose to go to a region that is a time successor of the current region, and thus respects the rules of the region game. Symmetrically, player 2 can either let player 1 play, or play a letter from Σ_2 with a region which is a time predecessor of the region proposed by player

1. Again, the automaton D being complete, player 2 can play any letter in Σ_2, but he can only play in regions that are time successors of the current region. The initial position is $q^G_{in} = (q^D_{in}, r_{\text{in}})$. Finally, the labelling of the positions reflects the colouring of the states in D: $\lambda^G(q,r) = \lambda^G(q,r,\sigma^1,r^1) = \alpha^D(q)$. Hence, a play in the parity game is winning for player 1 if and only if the word simulated is accepted by D. The next proposition shows the relationship between G_R and the corresponding parity game G_D.

Proposition 18. *Player 1 has a winning strategy in the region game G_R* if and only if *he has a winning strategy in the corresponding parity game G_D.*

Because parity games admit memoryless strategies, and thanks to Proposition 18, we can deduce a bound on the memory needed to win a region game whose objective is given by $\mathsf{L}_{\mathsf{coB}}(\mathsf{Rg}(A))$ for a universal PastECA A.

Lemma 19. *Let A be a universal* PastECA *with n locations and maximal constant $cmax$. If player 1 has a winning strategy in $G_R = \langle \Sigma_1, \Sigma_2, cmax, \mathsf{L}_{\mathsf{coB}}(\mathsf{Rg}(A))\rangle$, then he has a finite-state strategy, represented by a deterministic finite state transition system with at most m states, where $m = (2n^n n! + 1) \times |\mathsf{Reg}(\mathbb{H}_\Sigma, cmax)|$.*

Proof. If player 1 has a winning strategy in $\langle \Sigma_1, \Sigma_2, cmax, \mathsf{L}_{\mathsf{coB}}(\mathsf{Rg}(A))\rangle$, then by Proposition 18, and by the memoryless property of parity games, he has a memoryless winning strategy in the parity game G_D, $\pi^G : Q^G_1 \to Q^G_2$. From this memoryless strategy, one can define a finite-state strategy for player 1 in the original region game. We first define $\pi : Q^G_1 \to \Sigma_1 \times \mathsf{Reg}(\mathbb{H}_\Sigma, cmax)$ as follows. For all $q \in Q^D$, $r \in \mathsf{Reg}(\mathbb{H}_\Sigma, cmax)$: $\pi(q,r) = (\sigma^1, r^1)$ iff $\pi^G(q,r) = (q,r,\sigma^1,r^1)$. Then, we let A^π be the finite transition system $\left\langle Q^G_1, q^G_{in}, \Sigma \times \mathsf{Reg}(\mathbb{H}_\Sigma, cmax), \delta^\pi \right\rangle$ where, for all $q = (q_1, r_1) \in Q^G_1$, $(\sigma, r) \in \Sigma \times \mathsf{Reg}(\mathbb{H}_\Sigma, cmax)$: $\delta^\pi(q,(\sigma,r)) = (q'_1, r'_1)$ iff (i) $q'_1 = \delta^D(q_1,(\sigma,r))$ and (ii) $r'_1 = \delta^R(r_1,(\sigma,r)))$ and (iii) either $\pi(q) = (\sigma,r)$, or $\pi(q) = (\sigma', r')$ and $\sigma \in \Sigma_2$, and $r \leq_{\text{t.s.}} r'$. In the other cases, δ is undefined.

From π and A^π, we can define the strategy π^R to be played in the region game as follows. Let $\Delta : Q^G_1 \times (\Sigma \times \mathsf{Reg}(\mathbb{H}_\Sigma, cmax))^* \to Q^G_1 \cup \{\bot\}$ be the function s.t. $\Delta(q,w)$ is the location reached in A^π after reading the finite word w from location q, or $\bot$ if w cannot be read from q. Then, π^R is defined as follows. For any $\rho^R = (\sigma_1, r_1)\cdots(\sigma_n, r_n)$, we let $\pi^R(\rho^R) = \pi(\Delta(q^G_{in}, \rho^R))$ if $\Delta(q^G_{in}, \rho^R) \neq \bot$; otherwise: $\pi^R(\rho^R) = (\sigma, r_n)$ where σ is any letter in Σ_1. Remark that, by definition of π and A^π, the proposed region is always a time successor of the last region of the play, so the strategy is correctly defined. Let us show that π^R is winning: let $\rho^R = (\sigma_0, r_0)\cdots(\sigma_j, r_j)\cdots$ be a play consistent with π^R. By definition of π^R, there is a run $R = q^G_{in}, (\sigma_0, r_0), q^G_1, \cdots, q^G_j, (\sigma_j, r_j), \cdots$ of A^π over ρ^R. It is easy to see that one can construct from this run a play in G^D that is consistent with π^G. Then, since π^G is winning, $\rho^R \in \mathsf{L}_{\mathsf{P}}(D) = \mathsf{L}_{\mathsf{coB}}(\mathsf{Rg}(A))$. Since this is true for any run that is consistent with π^R, it is a winning strategy.

Finally, observe that the number of states of A^π is $|Q^D| \times |\mathsf{Reg}(\mathbb{H}_\Sigma, cmax)|$. By the result of [Pit07], $|Q^{\tilde{D}}| = 2n^n n!$. Then $|Q^D| = 2n^n n! + 1$, and this establishes the bound $m = (2n^n n! + 1) \times |\mathsf{Reg}(\mathbb{H}_\Sigma, cmax)|$. □

Thanks to Lemma 19, we can now prove that we can strengthen the co-Büchi condition of the objective of the region game, to a K-co-Büchi condition:

Proposition 20. *Let A be a universal* PastECA *with co-Büchi acceptance condition, n locations and maximal constant $cmax$. Then, player 1 has a winning strategy in $G^R = \langle \Sigma_1, \Sigma_2, cmax, \mathsf{L}_{\mathsf{coB}}(\mathsf{Rg}(A))\rangle$* if and only if *he has a winning strategy in $G^R_K = \langle \Sigma_1, \Sigma_2, cmax, \mathsf{L}_{\mathsf{KcoB}}(\mathsf{Rg}(A))\rangle$, with $K = (2n^{n+1}n! + n) \times |\mathsf{Reg}(\mathbb{H}_\Sigma, cmax)|$.*

Proof. First, observe that, since $\mathsf{L}_{\mathsf{KcoB}}(\mathsf{Rg}(A)) \subseteq \mathsf{L}_{\mathsf{coB}}(\mathsf{Rg}(A))$, any winning strategy for player 1 in G^R_K, is winning in G^R.

Conversely, suppose player 1 has a winning strategy in G^R. Then, by Lemma 19, there is a strategy π and a transition system $A^\pi = \langle Q^\pi, q^\pi_{in}, \Sigma \times \mathsf{Reg}(\mathbb{H}_\Sigma, cmax), \delta^\pi\rangle$ with m locations (where $m = (2n^n n! + 1) \times |\mathsf{Reg}(\mathbb{H}_\Sigma, cmax)|$) s.t. $\mathsf{Outcome}(G_R, \pi) = \mathsf{L}(A^\pi)$ and $\mathsf{L}(A^\pi) \subseteq \mathsf{L}_{\mathsf{coB}}(\mathsf{Rg}(A))$. Let $\mathsf{Rg}(A) = \langle Q, q_{in}, \Sigma \times \mathsf{Reg}(\mathbb{H}_\Sigma, cmax), \delta, \alpha\rangle$, and let $A^\pi \times \mathsf{Rg}(A) = \langle Q^\pi \times Q, (q^\pi_{in}, q_{in}), \Sigma \times \mathsf{Reg}(\mathbb{H}_\Sigma, cmax), \delta'\rangle$ be the transition system s.t. for all $(q^\pi, q) \in Q^\pi \times Q$, for all $(\sigma, r) \in \Sigma \times \mathsf{Reg}(\mathbb{H}_\Sigma, cmax)$: $(q^\pi_2, q_2) \in \delta'((q^\pi_1, q_1), (\sigma, r))$ iff $\delta^\pi(q^\pi_1, (\sigma, r)) = q^\pi_2$ and q_2 appears as a conjunct in $\delta(q_1, (\sigma, r))$ (recall that $\mathsf{Rg}(A)$ is universal). Clearly, each run of $A^\pi \times \mathsf{Rg}(A)$ simulates a run of A^π, together with a branch that has to appear in a run of $\mathsf{Rg}(A)$.

Then, let us show that there is, in $A^\pi \times \mathsf{Rg}(A)$, no cycle that contains a location from $Q^\pi \times \alpha$. This is established by contradiction. Assume such a cycle exists, and let $(q^\pi_{in}, q_{in})(q^\pi_1, q_1)(q^\pi_2, q_2) \cdots (q^\pi_j, q_j) \cdots$ be an infinite run of $A^\pi \times \mathsf{Rg}(A)$ that visits a location from $Q^\pi \times \alpha$ infinitely often. Moreover, let w be the infinite word labeling this run. Then, clearly, $q^\pi_{in} q^\pi_1 q^\pi_2 \cdots q^\pi_j \cdots$ is a run of A^π that accepts w. On the other hand, the run of $\mathsf{Rg}(A)$ on w necessarily contains a branch labelled by $q_{in} q_1 q_2 \cdots q_j \cdots$. Since this branch visits α infinitely often, $\mathsf{Rg}(A)$ rejects w because the acceptance condition α of $\mathsf{Rg}(A)$ is co-Büchi. This contradicts the fact that $\mathsf{L}(A^\pi) \subseteq \mathsf{L}_{\mathsf{coB}}(\mathsf{Rg}(A))$.

Then, any word accepted by A^π visits at most $m \times n$ times an accepting state of $\mathsf{Rg}(A)$, and $\mathsf{L}(A^\pi) \subseteq \mathsf{L}_{\mathsf{KcoB}}(\mathsf{Rg}(A))$, with $K = (2n^n n! + 1) \times |\mathsf{Reg}(\mathbb{H}_\Sigma, cmax)| \times n = (2n^{n+1}n! + n) \times |\mathsf{Reg}(\mathbb{H}_\Sigma, cmax)|$. Thus, player 1 has a winning strategy in $\langle \Sigma_1, \Sigma_2, cmax, \mathsf{L}_{\mathsf{KcoB}}(\mathsf{Rg}(A))\rangle$ too. □

Thanks to these results, we can now prove Theorem 15:

Proof of Theorem 15. Let $\overline{K} \geq (2n^{n+1}n! + n) \times |\mathsf{Reg}(\mathbb{H}_\Sigma, cmax)|$. If there is a winning strategy for player 1 in $G^T_{\overline{K}}$ then obviously there is a winning strategy for player 1 in G^T. Conversely, suppose there is a winning strategy for player 1 in G^T. Then, by Proposition 17, he has a winning strategy in $G^R = \langle \Sigma_1, \Sigma_2, cmax, \mathsf{L}_{\mathsf{coB}}(\mathsf{Rg}(A))\rangle$, and by Proposition 20, he has a winning strategy in $G^R_K = \langle \Sigma_1, \Sigma_2, cmax, \mathsf{L}_{\mathsf{KcoB}}(\mathsf{Rg}(A))\rangle$, with $K = (2n^{n+1}n! + n) \times |\mathsf{Reg}(\mathbb{H}_\Sigma, cmax)|$. By applying again Proposition 17, he has a winning strategy in the timed game $G^T_K = \langle \Sigma_1, \Sigma_2, \mathsf{L}_{\mathsf{KcoB}}(A)\rangle$. Since $K \leq \overline{K}$, $\mathsf{L}_{\mathsf{KcoB}}(A) \subseteq \mathsf{L}_{\overline{K}\mathsf{coB}}(A)$. Hence player 1 has a winning strategy in $G^T_{\overline{K}}$. □

Solving games defined by UECA *with K-co-Büchi acceptance condition.* For solving those games, we show how to build, from a UECA $A = \langle Q, q_{in}, \Sigma, \delta, \alpha\rangle$ with K-co-Büchi acceptance condition, a DECA with 0-co-Büchi acceptance condition which is denoted $\mathsf{Det}_K(A)$, that accepts the same timed language. The construction of this DECA is based on a generalization of the *subset construction*. When applied to an untimed universal automaton A with set of locations Q, the classical subset construction consists in building a new automaton A' whose locations are subsets of Q. Thus, each location of A' encodes the set of locations of A that are active at each level of the run

tree. In the case of K-co-Büchi automata, one needs to remember how many times accepting states have been visited on the branches that lead to each active location. As a consequence, the locations of the subset construction should be sets of the form $\{(q_1, n_1), \ldots, (q_\ell, n_\ell)\}$, where each q_i is an active location that has been reached by a branch visiting exactly n_i accepting states. However, in this case, the set of locations in the subset construction is not finite anymore. This can be avoided by observing that we can keep only the maximal number of visits (up to $K+1$) to accepting locations among all the branches that reach q. So, the states of the deterministic automaton are functions $F : Q \mapsto \{-1, 0, 1, \ldots, K, K+1\}$, where $F(q) = -1$ means that q is not currently active, $F(q) = k$ with $0 \leq k \leq K$ means that q is currently active and that the branch with maximal number of visits to α that leads to q has visited accepting states k times, and $F(q) = K+1$ means that q is currently active and that the branch with maximal numbers of visits to α that leads to q has visited accepting states more than K times. In this last case, the timed word which is currently read has to be rejected, because of the K-co-Büchi condition.

Formally, $\mathsf{Det}_K(A) = \langle \mathcal{F}, F_0, \Sigma, \Delta, \alpha_K \rangle$ where the following holds. $\mathcal{F} = \{F \mid F : Q \to \{-1, 0, 1, \ldots, K, K+1\}\}$. If we let $(q \in \alpha)$ be the function that returns 1 if $q \in \alpha$ and 0 otherwise, $F_0 \in \mathcal{F}$ is such that $F_0(q_0) = (q_0 \in \alpha)$ and $F_0(q) = -1$ for all $q \in Q$ and $q \neq q_0$. Now, $\Delta(F, \sigma, \psi)$ is defined if there exists a function $h : \{q \in Q \mid F(q) \geq 0\} \to \mathsf{Constr}(\mathbb{P}_\Sigma)$ s.t. (i) ψ is equal to $\bigwedge_{q|F(q)\geq 0} h(q)$ and this formula is satisfiable, (ii) for all $q \in Q$ such that $F(q) \geq 0$, $\delta(q, \sigma, h(q))$ is defined. In this case, $\Delta(q, \sigma, \psi) = F'$ where F' is the counting function such that for all $q \in Q$, $F'(q)$ equals: $\max\left\{ \min\left(K+1, F(p)+(q \in \alpha)\right) \;\middle|\; q \in \delta(p, \sigma, h(p)) \wedge F(p) \neq -1 \right\}$. Finally, $\alpha_K = \{F \in \mathcal{F} \mid \exists q \in Q \cdot F(q) = K+1\}$.

Proposition 21. *For all* UECA A*, for all* $K \in \mathbb{N}$*:* $\mathsf{L}_{K\mathsf{coB}}(A) = \mathsf{L}_{0\mathsf{coB}}(\mathsf{Det}_K(A))$.

From this deterministic automaton, it is now easy to construct a *timed safety game* for solving the realizability problem. We do that in the next section when solving the realizability problem of a real-time extension of the logic LTL.

Remark 22 (Time divergence). Handling time divergence in timed games requires techniques that are more involved than the ones suggested in previous sections. In the timed games considered in this section, if the set of winning plays only contains divergent timed words, then clearly player 1 can not win the game, no matter what the objective is. Indeed, as player 2 can always overtake the action proposed by player 1, he can easily block time and ensure that the output of the game is a convergent timed word. To avoid such pathological behaviors, the specification should declare player 1 winning in those cases. In [dAFH+03], the interested reader will find an extensive discussion on how to decide winner in the presence of time convergence.

5 Application: Realizability of $\mathsf{LTL}_\triangleleft$

The $\mathsf{LTL}_\triangleleft$ *logic.* The logic $\mathsf{LTL}_\triangleleft$ we consider here is a fragment of the Event Clock Logic (ECL for short) [Ras99, RS98, HRS98]. ECL is an extension of LTL with two real-time operators: the history operator $\triangleleft_I \varphi$ expressing that φ was true for the last time t time units ago for some $t \in I$, and the prediction operator $\triangleright_I \varphi$ expressing that the

next time φ will be true is in t time units for some $t \in I$ (where I is an interval). $\mathsf{LTL}_\triangleleft$ is obtained by *disallowing prediction operators*. The realizability problem for ECL is as in the previous section with the exception that the set of winning plays is defined by an ECL formula instead of a UECA. The *realizability problem* has recently [DGRR09] been shown 2EXPTIME-complete for $\mathsf{LTL}_\triangleleft$ but undecidable[1] for the full ECL. In this paper, we further restrict ourselves to the case where expressions of the form $\triangleleft_{\mathrm{I}}\varphi$ appear with $\varphi = a$ only, where a is some alphabet letter. Remark that this last restriction is not necessary to obtain decidability [DGRR09], but it makes the presentation easier. Our results carry on to the more general case.

Formally, given an alphabet Σ, the syntax of $\mathsf{LTL}_\triangleleft$ is as follows (with $a \in \Sigma$):

$$\psi \in \mathsf{LTL}_\triangleleft ::= a \mid \neg\psi \mid \psi \vee \psi \mid \psi\,\mathcal{S}\,\psi \mid \psi\,\mathcal{U}\,\psi \mid \triangleleft_{\mathrm{I}} a$$

The models of an $\mathsf{LTL}_\triangleleft$ formula are infinite timed words. A timed word $\theta = (w, \tau)$ *satisfies* a formula $\varphi \in \mathsf{LTL}_\triangleleft$ at position $i \in \mathbb{N}$, written $\theta, i \models \varphi$, according to the following rules:

- if $\varphi = a$, then $w_i = a$;
- if $\varphi = \neg\varphi_1$, then $\theta, i \not\models \varphi_1$;
- if $\varphi = \varphi_1 \vee \varphi_2$, then $\theta, i \models \varphi_1$ or $\theta, i \models \varphi_2$;
- if $\varphi = \varphi_1\,\mathcal{S}\,\varphi_2$, then there exists $0 \leq j < i$ such that $\theta, j \models \varphi_2$ and for all $j < k < i$, $\theta, k \models \varphi_1$;
- if $\varphi = \varphi_1\,\mathcal{U}\,\varphi_2$, then there exists $j > i$ such that $\theta, j \models \varphi_2$ and for all $i < k < j$, $\theta, k \models \varphi_1$;
- if $\varphi = \triangleleft_{\mathrm{I}} a$, then there exists $0 \leq j < i$ such that $w_j = a$, $\tau_i - \tau_j \in I$, and for all $j < k < i$, $w_k \neq a$;

When $\theta, 0 \models \varphi$, we simply write $\theta \models \varphi$ and we say that θ satisfies φ. We denote by $[\![\varphi]\!]$ the set $\{\theta \mid \theta \models \varphi\}$ of models of φ. Finally, we define the following shortcuts: $\mathsf{true} \equiv a \vee \neg a$ with $a \in \Sigma$, $\mathsf{false} \equiv \neg\mathsf{true}$, $\varphi_1 \wedge \varphi_2 \equiv \neg(\neg\varphi_1 \vee \neg\varphi_2)$, $\varphi_1 \rightarrow \varphi_2 \equiv \neg\varphi_1 \vee \varphi_2$, $\Diamond\varphi \equiv \mathsf{true}\,\mathcal{U}\,\varphi$, $\Box\varphi \equiv \varphi \wedge \neg\Diamond(\neg\varphi)$, $\bigcirc\varphi \equiv \mathsf{false}\,\mathcal{U}\,\varphi$, $\ominus\varphi \equiv \mathsf{false}\,\mathcal{S}\,\varphi$, and $\Diamond\!\!\!-\varphi \equiv \mathsf{true}\,\mathcal{S}\,\varphi$. We also freely use notations like $\geq x$ to denote the interval $[x, \infty)$, or $< x$ for $[0, x)$, etc. in the $\triangleleft$ operator.

Let $\Sigma = \Sigma_1 \uplus \Sigma_2$ be an alphabet that is partitioned into a set Σ_1 of player 1 events (*controllable* events), and Σ_2 of player 2 events (*uncontrollable* events), and let φ be an $\mathsf{LTL}_\triangleleft$ formula on Σ. Then, φ is *realizable* iff Player 1 has a winning strategy in the TG $\langle \Sigma_1, \Sigma_2, [\![\varphi]\!] \rangle$. The *realizability problem* for $\mathsf{LTL}_\triangleleft$ asks, given an $\mathsf{LTL}_\triangleleft$ formula φ whether φ is realizable.

An efficient algorithm to solve realizability of $\mathsf{LTL}_\triangleleft$. Let us now show how to exploit the results from the previous section to obtain an *incremental* algorithmic schema that solves the realizability problem of $\mathsf{LTL}_\triangleleft$. From an $\mathsf{LTL}_\triangleleft$ formula φ, we build, using standard techniques [Ras99, RS98], a NECA with Büchi acceptance condition $A_{\neg\varphi}$ s.t. $\mathsf{L}_{\mathsf{B}}(A_{\neg\varphi}) = [\![\neg\varphi]\!]$. Then, we consider its dual $\tilde{A}_{\neg\varphi}$, which is thus a UECA with

[1] Note that the undecidability proof has been made for a slightly different definition of timed games, but the proof can be adapted to the definition we rely on in the present paper.

co-Büchi acceptance condition s.t. $\mathsf{L}_{\mathsf{coB}}(\tilde{A}_{\neg\varphi}) = [\![\varphi]\!]$. As a consequence, solving the *realizability problem* for φ now amounts to finding a winning strategy for player 1 in the timed game $\left\langle \Sigma_1, \Sigma_2, \mathsf{L}_{\mathsf{coB}}(\tilde{A}_{\neg\varphi}) \right\rangle$. Theorem 15 tells us that we can reduce this to finding a winning strategy in a timed game whose objective is given by an automaton with *K-co-Büchi acceptance condition* (for a precise value of K). In this game, the objective of player 1 is thus to *avoid visiting accepting states too often* (no more than K times), and this is thus a *safety condition*. The automaton $\mathsf{Det}_K(\tilde{A}_{\neg\varphi})$ can be used to define a timed safety game. Such games can be solved by tools such as UPPAAL TIGA [BCD ŕ 07].

The drawback of this approach is that the value K is potentially intractable: it is doubly-exponential in the size of φ. As a consequence, $\mathsf{Det}_K(\tilde{A}_{\neg\varphi})$ and its underlying timed safety game are unmanageably large. To circumvent this difficulty, we adopt an incremental approach. Instead of solving the game underlying $\mathsf{Det}_K(\tilde{A}_{\neg\varphi})$, we solve iteratively the games underlying $\mathsf{Det}_i(\tilde{A}_{\neg\varphi})$ for increasing values of $i = 0, 1, \ldots$. As soon as player 1 can win a game for some i, we can stop and conclude that φ is *realizable*. Indeed, $\mathsf{L}_{0\mathsf{coB}}(\mathsf{Det}_i(\tilde{A}_{\neg\varphi})) = \mathsf{L}_{i\mathsf{coB}}(\tilde{A}_{\neg\varphi})$ by Proposition 21, and $\mathsf{L}_{i\mathsf{coB}}(\tilde{A}_{\neg\varphi}) \subseteq \mathsf{L}_{K\mathsf{coB}}(\tilde{A}_{\neg\varphi}) \subseteq [\![\varphi]\!]$. In other words, realizability of $\mathsf{L}_{0\mathsf{coB}}(\mathsf{Det}_i(\tilde{A}_{\neg\varphi}))$ implies realizability of φ. Unfortunately, if φ is *not realizable*, this approach fails to avoid considering the large theoretical bound K. To circumvent this second difficulty, we use the property that our games are determined: φ is not realizable by player 1 iff $\neg\varphi$ is realizable by player 2. So in practice, we execute two instances of our incremental algorithm in parallel and stop whenever one of the two is conclusive. The details of this incremental approach are given in [FJR09], and it is experimentally shown there, in the case of LTL specifications, that the values that one needs to consider for i are usually very small.

To sum up, our incremental algorithm works as follows. Fix an $\mathsf{LTL}_{\lhd}$ formula φ, and set i to 0. Next, **if** player 1 has a winning strategy in $\left\langle \Sigma_1, \Sigma_2, \mathsf{L}_{0\mathsf{coB}}(\mathsf{Det}_i(\tilde{A}_{\neg\varphi})) \right\rangle$, **then** φ is *realizable*; **else if** player 2 has a winning strategy in $\left\langle \Sigma_1, \Sigma_2, \mathsf{L}_{0\mathsf{coB}}(\mathsf{Det}_i(\tilde{A}_{\varphi})) \right\rangle$, **then** φ is *not realizable*; **else**, increment i by 1 and iterate.

Experiments with UPPAAL TIGA. We have thus reduced the realizability problem of $\mathsf{LTL}_{\lhd}$ to solving a sequence of TG of the form $\langle \Sigma_1, \Sigma_2, \mathsf{L}_{0\mathsf{coB}}(A) \rangle$, where A is a DECA. Solving each of these games amounts to solving a *safety game* played in an arena which is defined by A (where the edges are partitioned according to Σ_1 and Σ_2). In practice, this can be done using UPPAAL TIGA [BCD ŕ 07], as we are about to show thanks to a *simple* yet *realistic* example. Our example consists of a system where a controller monitors an input line that can be in two states: *high* or *low*. The state of the input line is controlled by the environment, thanks to the actions *up* and *down*, that respectively change the state from low to high and high to low. Changes in the state of the input line might represent *requests* that the controller has to *grant*. More precisely, whenever consecutive *up* and *down* events occur separated by at least two time units, the controller has to issue a *grant* after the corresponding *down* but before the next *up*. Moreover, successive *grants* have to be at least three time units apart, and *up* and *down* events have to be separated by at least one time unit. This informal requirement is captured by

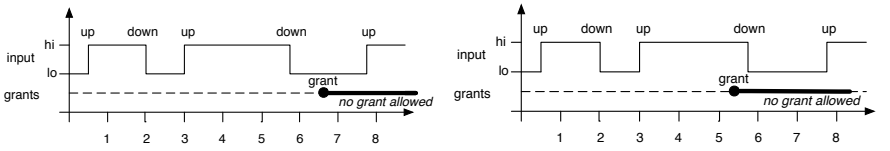


Fig. 1. Two examples of execution of the systems. The state of the input is represented on top, grants are represented at the bottom. Each dot represents a *grant* event. Thick lines represent the period during which the controller cannot produce any *grant* because of Req_2.

the $\mathsf{LTL}_{\lhd}$ formula $\varphi \equiv \mathsf{Hyp} \rightarrow \mathsf{Req}_1 \wedge \mathsf{Req}_2$ on $\Sigma = \Sigma_1 \uplus \Sigma_2$ where $\Sigma_1 = \{grant\}$, $\Sigma_2 = \{up, down\}$ and:

$$\begin{aligned}
\mathsf{Hyp} &\equiv \Box\Big(up \rightarrow \big(\neg down\, \mathcal{U}(down \wedge \lhd_{\geq 1} up)\big)\Big) \wedge \\
&\quad \Box\Big(down \rightarrow \big(\neg up\, \mathcal{U}(up \wedge \lhd_{\geq 1} down)\big)\Big) \\
\mathsf{Req}_1 &\equiv \Box\big((down \wedge \lhd_{>2} up) \rightarrow (\neg up\, \mathcal{U}\, grant)\big) \\
\mathsf{Req}_2 &\equiv \Box(grant \rightarrow \neg \lhd_{<3} grant)
\end{aligned}$$

Remark that φ does not forbid the controller from producing *grant* events that have not been requested by the environment. However, a controller producing *grants too often* might hinder itself because Req_2 requires each pair of grants to be separated from each other by at least 3 time units. Fig. 1 illustrates this by showing two prefixes of executions. The left part shows a prefix that respects φ. The right part of the figure shows a case where the controller has issued an unnecessary grant that prevents him from granting the request that appears with the *down* event at time 5.75.

Let us now apply the algorithmic schema presented above to this example. We first build the NECA with Büchi acceptance condition $A_{\neg\varphi}$, given in Fig. 2. This automaton has two parts, identified by the names of the states: the top part (corresponding to the states $1, \ldots 7$) accepts the models of $[\![\neg(\mathsf{Hyp} \rightarrow \mathsf{Req}_1)]\!]$ and the lower part (states $1, \overline{2}, \ldots, \overline{6}$) accepts the models of $[\![\neg(\mathsf{Hyp} \rightarrow \mathsf{Req}_2)]\!]$, so the whole automaton accepts exactly $[\![\neg\varphi]\!]$. Fig. 2 can also be regarded as a depiction of the dual UECA with co-Büchi acceptance condition $\tilde{A}_{\neg\varphi}$, by interpreting non-determinism as universal branching.

From $\tilde{A}_{\neg\varphi}$, we have applied the counting functions construction described above, for $i = 1$. In order to ease the presentation, we have applied this construction separately on the two parts of the automaton, to obtain G_1 and G_2, given in Fig. 3. These automata are shown as they appear in their UPPAAL TIGA encoding: controllable transitions are plain, and uncontrollable transitions are dashed. The history clocks corresponding to *up*, *down* and *grant* are respectively denoted u, d and g. Remark that since UPPAAL TIGA uses classical Alur-Dill timed automata, and not NECA, we have to explicitly manage the reset of those clocks. Finally, observe that we have used the synchronisation mechanism offered by UPPAAL TIGA to ensure that the game is played on the synchronous product of these two automata (which corresponds to the counting function construction applied to $A_{\neg\varphi}$).

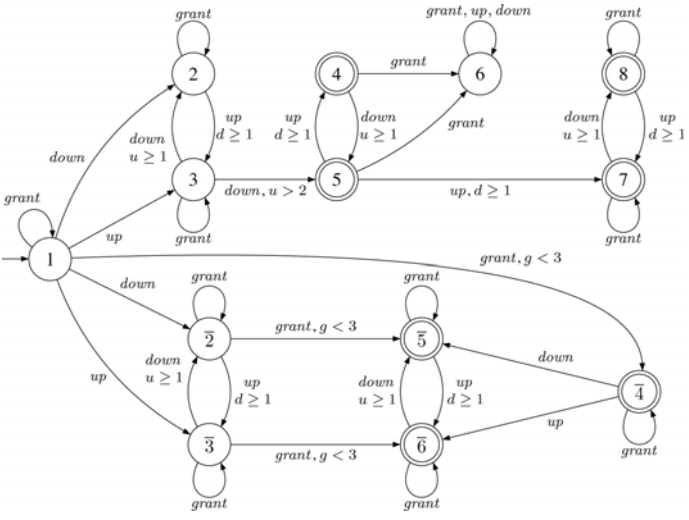


Fig. 2. The NECA $A_{\neg\varphi}$

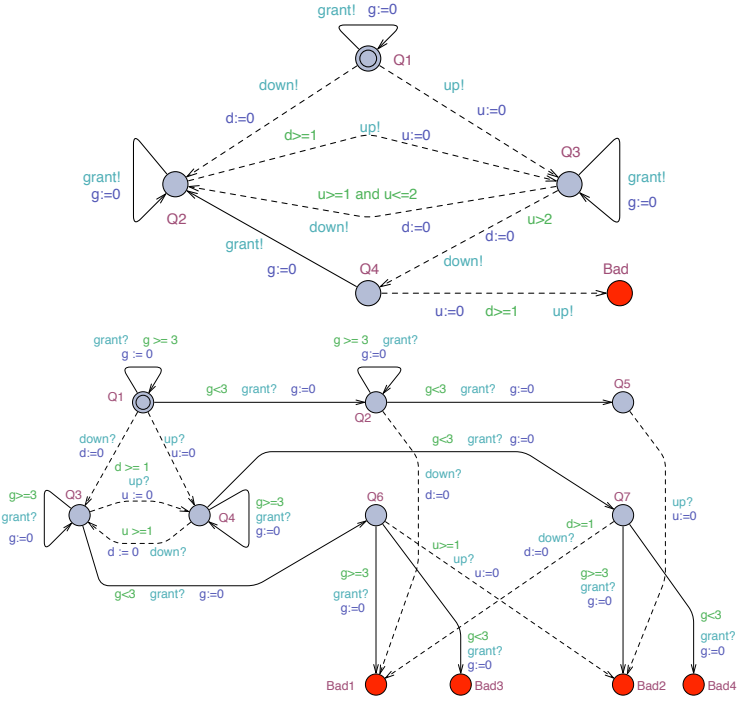


Fig. 3. The DECA obtained from the two parts of $A_{\neg\varphi}$, when applying the counting functions construction for $i = 1$. Unreachable states, as well as transitions to the state F with $F(q) = -1$ for any q are not shown.

We provided this model to UPPAAL TIGA together with the synthesis objective `control: A[not BadState]`, where `BadState` is true iff one of the automata reaches one of its Bad locations (that corresponds to one of the counters being > 1). In this case, UPPAAL TIGA can compute a *winning strategy* for player 1, which means that player 1 is capable of ensuring that, on any branch of any run of $\tilde{A}_{\neg\varphi}$, accepting states occur at most one time. This strategy thus ensures that all the plays are accepted by $\tilde{A}_{\neg\varphi}$, and so they all satisfy φ. Hence, φ is realizable. This example shows that, although an exponentially-large K might be needed to prove realizability of an $\mathsf{LTL}_{\triangleleft}$ formula, in practice, small values of i (here, 1) might be sufficient. A larger set of experiments (on large LTL formulas) exploiting the same techniques can be found in [FJR09]. These experiments confirm that small values of i are sufficient in practice.

Remark 23 (Time divergence). In this example, time divergence is not an issue. Indeed, the objective is such that, on the one hand, player 1 wins the game if player 2 proposes to play *up* followed by *down*, or *down* followed by *up* without waiting at least one time unit (because of Hyp), and, on the other hand, player 1 violates Req_2 if he plays two *grant* actions too close in time (less than 3 t.u. apart).

References

[AD94] Alur, R., Dill, D.L.: A Theory of Timed Automata. TCS 126(2) (1994)

[AFH99] Alur, R., Fix, L., Henzinger, T.A.: Event-clock automata: a determinizable class of timed automata. TCS 211(1-2) (1999)

[BCD+07] Behrmann, G., Cougnard, A., David, A., Fleury, E., Larsen, K., Lime, D.: Uppaal-tiga: Time for playing games! In: Damm, W., Hermanns, H. (eds.) CAV 2007. LNCS, vol. 4590, pp. 121–125. Springer, Heidelberg (2007)

[dAFH+03] de Alfaro, L., Faella, M., Henzinger, T.A., Majumdar, R., Stoelinga, M.: The element of surprise in timed games. In: Amadio, R.M., Lugiez, D. (eds.) CONCUR 2003. LNCS, vol. 2761, pp. 144–158. Springer, Heidelberg (2003)

[DGRR09] Doyen, L., Geeraerts, G., Raskin, J.-F., Reichert, J.: Realizability of real-time logics. In: Ouaknine, J., Vaandrager, F.W. (eds.) FORMATS 2009. LNCS, vol. 5813, pp. 133–148. Springer, Heidelberg (2009)

[DR10] Doyen, L., Raskin, J.-F.: Antichain algorithms for finite automata. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 2–22. Springer, Heidelberg (2010)

[EJ91] Emerson, E.A., Jutla, C.S.: Tree automata, mu-calculus and determinacy. In: FCS'91, IEEE, Los Alamitos (1991)

[FJR09] Filiot, E., Jin, N., Raskin, J.-F.: An antichain algorithm for ltl realizability. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 263–277. Springer, Heidelberg (2009)

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

[KV01] Kupferman, O., Vardi, M.Y.: Weak alternating automata are not that weak. ACM Trans. Comput. Log. 2(3) (2001)

[KV05] Kupferman, O., Vardi, M.Y.: Safraless decision procedures. In: FOCS'05. IEEE, Los Alamitos (2005)

[MH84] Miyano, S., Hayashi, T.: Alternating finite automata on ω-words. TCS 32 (1984)
[Pit07] Piterman, N.: From nondeterministic Büchi and Streett automata to deterministic parity automata. LMCS 3(3) (2007)
[Ras99] Raskin, J.-F.: Logics, Automata and Classical Theories for Deciding Real Time. PhD thesis, FUNDP, Belgium (1999)
[RS98] Raskin, J.-F., Schobbens, P.-Y.: The logic of event clocks: decidability, complexity and expressiveness. Automatica 34(3) (1998)
[Saf88] Safra, S.: On the complexity of ω-automata. In: FOCS'88. IEEE, Los Alamitos (1988)
[SF07] Schewe, S., Finkbeiner, B.: Bounded synthesis. In: Namjoshi, K.S., Yoneda, T., Higashino, T., Okamura, Y. (eds.) ATVA 2007. LNCS, vol. 4762, pp. 474–488. Springer, Heidelberg (2007)

Property-Based Monitoring of Analog and Mixed-Signal Systems

John Havlicek[1], Scott Little[1], Oded Maler[2], and Dejan Nickovic[3]

[1] Freescale Semiconductor, Austin, TX, USA
[2] Verimag, Grenoble, France
[3] IST Austria, Klosterneuburg, Austria

Abstract. In the recent past, there has been a steady growth of the market for consumer *embedded devices* such as cell phones, GPS and portable multimedia systems. In embedded systems, digital, analog and software components are combined on a single chip, resulting in increasingly complex designs that introduce richer functionality on smaller devices. As a consequence, the potential insertion of errors into a design becomes higher, yielding an increasing need for automated analog and mixed-signal validation tools. In the purely digital setting, formal verification based on properties expressed in industrial specification languages such as PSL and SVA is nowadays successfully integrated in the design flow. On the other hand, the validation of analog and mixed-signal systems still largely depends on simulation-based, ad-hoc methods. In this tutorial, we consider some ingredients of the standard verification methodology that can be successfully exported from digital to analog and mixed-signal setting, in particular *property-based monitoring* techniques. Property-based monitoring is a lighter approach to the formal verification, where the system is seen as a "black-box" that generates sets of traces, whose correctness is checked against a property, that is its high-level specification. Although incomplete, monitoring is effectively used to catch faults in systems, without guaranteeing their full correctness.

In the first part of the tutorial, we present a technique for property-based analog and mixed-signal monitoring. In the heart of the framework lies *signal temporal logic* STL, that is a high-level specification language allowing to express transient properties of analog, mixed and timed signals. STL is an extension of the real-time *metric interval temporal logic* MITL, where one can specify temporal relations between relevant "events" in the continuous signals that are captured using numerical predicates. We then present procedures for automatic translation of arbitrary STL specifications into *monitors*, i.e. programs that check the correctness of a set of simulation traces with respect to the given property. We introduce *analog monitoring tool* AMT that implements the presented monitoring techniques and illustrate the usefulness and the limitations of the approach on two industrial case studies, considering properties of a FLASH memory cell and a DDR2 memory interface.

Although the STL framework implemented in AMT provides a theoretical basis for analog and mixed-signal verification, it is not adequate for industrial-strength verification. The DDR2 example shows that STL lacks the expressiveness to specify complex timing relationships required

K. Chatterjee and T.A. Henzinger (Eds.): FORMATS 2010, LNCS 6246, pp. 23–24, 2010.

by industrial designs. The STL framework provides real-time extensions to *linear temporal logic*, but it does not provide extensions to the regular expressions present in both PSL and SVA. However, the industrial interest in the methodology resulted in the creation of the A-SVA subcommittee of the Accellera Verilog-AMS committee that is investigating extending SVA with features that would enable industrial-strength property-based monitoring of analog and mixed-signal systems. This includes in particular real-time regular expressions, local variables, and requirements for the properties to access accurate continuous values. The committee have agreed upon a preliminary syntax and semantics for real-time regular expressions and local variables. They are in the process of integrating the real-time regular expressions with the digital and real-time properties. These results are expected to be integrated into the next revisions of Verilog-AMS and SystemVerilog.

A Framework for Verification of Software with Time and Probabilities

Marta Kwiatkowska[1], Gethin Norman[2], and David Parker[1]

[1] Oxford University Computing Laboratory, Parks Road, Oxford, OX1 3QD, UK
[2] Department of Computing Science, University of Glasgow, Glasgow, G12 8RZ, UK

Abstract. Quantitative verification techniques are able to establish system properties such as "the probability of an airbag failing to deploy on demand" or "the expected time for a network protocol to successfully send a message packet". In this paper, we describe a framework for quantitative verification of software that exhibits both real-time and probabilistic behaviour. The complexity of real software, combined with the need to capture precise timing information, necessitates the use of abstraction techniques. We outline a quantitative abstraction refinement approach, which can be used to automatically construct and analyse abstractions of probabilistic, real-time programs. As a concrete example of the potential applicability of our framework, we discuss the challenges involved in applying it to the quantitative verification of SystemC, an increasingly popular system-level modelling language.

1 Introduction

Computerised systems pervade all aspects of modern society, including safety-critical application domains such as the automotive and avionics industries. This, combined with the growing complexity of such devices, necessitates the development of rigorous techniques to verify their correctness. Furthermore, this analysis must often take into account the *quantitative* aspects of the systems that are being verified. This includes both *real-time* characteristics and *probabilistic* behaviour. Embedded devices, in safety-critical applications for example, will often have strict timing requirements. Similarly, it is important to quantify the effect of inherently stochastic behaviour, such as component failures or message loss in communication between networked devices. Another source of probabilistic behaviour is the use of randomisation, an essential ingredient of many network protocols such as FireWire or Ethernet and wireless technologies including Bluetooth and ZigBee.

Quantitative verification is a formal method for the analysis of timed and probabilistic systems. It is based on the construction of a mathematical model capturing the system's behaviour, followed by the analysis of formally specified quantitative properties. These might include, for example, "the probability of an airbag failing to deploy within 0.02 seconds", "the expected time for a network protocol to send a packet" or "the expected power consumption of a sensor network during 1 hour of operation". Notice that this permits an analysis not just of a system's correctness, but also its performance and reliability.

K. Chatterjee and T.A. Henzinger (Eds.): FORMATS 2010, LNCS 6246, pp. 25–45, 2010.

Quantitative verification techniques have seen a great deal of progress in recent years. For real-time systems, a prominent modelling formalism is timed automata, for which mature verification tools such as UPPAAL [1] exist. For probabilistic systems, the most commonly used models are Markov chains or Markov decision processes (MDPs). Probabilistic model checking tools such as PRISM [2] and MRMC [3] are widely used and have been successfully applied to the verification of a range of systems. Recent work [4–7] has seen progress in the development of tools for systems with time *and* probabilities.

A weakness of all these tools, however, is that they require the user to specify the system model in a custom modelling language. In order to minimise the chance of errors introduced in the modelling phase and to encourage the use of these tools, there is a need to extend quantitative verifications techniques to the languages used by real system designers. In the context of non-probabilistic verification, progress has been made in this direction. In particular, software model checking tools and techniques can now be applied directly to mainstream programming languages such as C and Java.

In this paper, we develop the underlying theory for quantitative verification of software with both probabilistic and real-time characteristics. We formalise the notion of *probabilistic timed programs*, whose semantics are defined in terms of infinite-state MDPs. The complexity of real software means that the use of *abstraction* is typically essential. Building on our previous work on tools and techniques for verifying a probabilistic extension of ANSI-C [8], we propose a *quantitative abstraction refinement* approach [5, 9, 10]. This builds successive, increasingly precise abstractions of an MDP, represented as two-player stochastic games [9]. At each step, the process is driven by *refinement* techniques which construct a new abstraction using information derived from quantitative verification of the stochastic game. This technique has already proven to be an efficient approach to the verification of probabilistic timed automata [5].

As a concrete illustration of the potential applicability of our verification framework, we discuss the challenges involved in applying it to the quantitative verification of SystemC, a C++-based system-level modelling language. SystemC is becoming increasingly prominent in the embedded systems domain, for example in the development of System-on-Chips (SoCs). Building formal verification techniques for SystemC has already been identified as an important but challenging direction of research [11]. Clearly, *quantitative* verification of SystemC will be even more demanding. We describe how some of the existing approaches and tools might be combined with our framework and identify some of the more important directions of future work.

An extended version of this paper, including additional details and proofs omitted from the text, can be found at [12].

2 Background Material

A distribution over Q is a function $\lambda : Q \to [0, 1]$ where the support $\{q \in Q \mid \lambda(q) > 0\}$ is countable and $\sum_{q \in Q} \lambda(q) = 1$, let $\mathsf{Dist}(Q)$ denote the set of such distributions.

2.1 MDPs and Stochastic Games

Markov decision processes (MDPs) are used to model systems that exhibit both nondeterministic and probabilistic behaviour.

Definition 1. *An MDP* M *is a tuple* $(S, \overline{S}, Act, Steps_{\mathsf{M}})$ *where* S *is a set of* states, $\overline{S} \subseteq S$ *is a set of* initial states, Act *is a set of* actions *and* $Steps_{\mathsf{M}} : S \times Act \to \mathsf{Dist}(S)$ *is a partial* probabilistic transition function.

For a state s of an MDP M, we write $Act(s)$ for the set of actions available in s, i.e., the actions $a \in Act$ for which $Steps_{\mathsf{M}}(s, a)$ is defined. The behaviour in state s is both probabilistic and nondeterministic: first an available action (i.e. an action in $Act(s)$) is selected nondeterministically, then a successor state is chosen according to the distribution $Steps_{\mathsf{M}}(s, a)$. A *path* is a sequence of such choices and a state is *reachable* if there is a path to it from an initial state. Under an *adversary* A, which resolves all nondeterminism, we can define a probability measure over paths [13]. The fundamental quantitative property for MDPs is that of *probabilistic reachability* which concerns the minimum or maximum probability of reaching a set of states F. Formally, we have:

$$p^{\min}_{\mathsf{M}}(F) \stackrel{\text{def}}{=} \inf_{s \in \overline{S}} \inf_A p^A_s(F) \quad \text{and} \quad p^{\max}_{\mathsf{M}}(F) \stackrel{\text{def}}{=} \sup_{s \in \overline{S}} \sup_A p^A_s(F)$$

where $p^A_s(F)$ denotes the probability of reaching target F, starting from s, when the MDP behaves according to adversary A.

Stochastic two-player games [14, 15] extend MDPs by allowing two types of nondeterministic choice, controlled by separate players.

Definition 2. *A* stochastic game G *is a tuple* $(S, \overline{S}, Act, Steps_{\mathsf{G}})$ *where* S *is a set of* states, $\overline{S} \subseteq S$ *is a set of* initial states Act *is a set of* actions *and* $Steps_{\mathsf{G}} : S \times Act \to 2^{\mathsf{Dist}(S)}$ *is a partial* probabilistic transition function.

The behaviour in a state s of a game G includes two successive nondeterministic choices: first player 1 selects an available action, then a distribution $\lambda \in Steps_{\mathsf{G}}(s, a)$ is selected by player 2. Finally, the successor state is then chosen according to the distribution λ. A pair of *strategies* (σ_1, σ_2) for players 1 and 2, resolve all the nondeterminism present in the game, and this induces a probability measure over the paths of the game.

2.2 Quantitative Abstraction Refinement

As proposed in [9], we use stochastic games to represent *abstractions* of MDPs. The key idea is to separate the two forms of nondeterminism: using player 1 choices to represent the nondeterminism caused by abstraction; and player 2 choices for the nondeterminism of the MDP. For an MDP M, the construction of an abstraction is based on a *partition* $\mathcal{P} = \{S_1, \ldots, S_n\}$ of its state space. For $\lambda \in \mathsf{Dist}(S)$, we let $\lambda_{\mathcal{P}} \in \mathsf{Dist}(\mathcal{P})$ denote the distribution where $\lambda_{\mathcal{P}}(S') = \sum_{s \in S'} \lambda(s)$ for all $S' \in \mathcal{P}$. Formally, we define the abstraction of an MDP as follows.

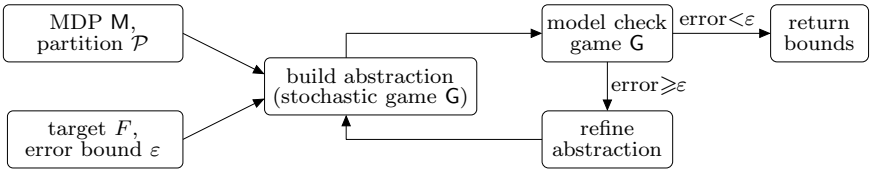


Fig. 1. Quantitative abstraction refinement for MDPs

Definition 3. *Let* $\mathsf{M}=(S,\overline{S},Act,Steps_{\mathsf{M}})$ *be an MDP and* $\mathcal{P}$ *a partition of* S. *The* abstraction of M with respect to $\mathcal{P}$ *is given by the stochastic game* $\mathsf{G}=(\mathcal{P},\overline{\mathcal{P}},2^{Act\times\mathsf{Dist}(\mathcal{P})},Steps_{\mathsf{G}})$ *where* $\overline{\mathcal{P}}=\{S'\in\mathcal{P}\mid S'\cap\overline{S}\neq\emptyset\}$ *and for* $S'\in\mathcal{P}$ *and* $\Theta\in 2^{Act\times\mathsf{Dist}(\mathcal{P})}$*:* $Steps_{\mathsf{G}}(S',\Theta)$ *is defined and equals* $\{\lambda\mid(a,\lambda)\in\Theta\wedge a\in Act\}$ *if and only if there exists* $s\in S'$ *such that* $\Theta=\{(a,Steps_{\mathsf{M}}(s,a)_{\mathcal{P}})\mid a\in Act(s)\}$.

If G is an abstraction of M, then the reachability probabilities on G yield lower and upper bounds on the reachability probabilities of M:

$$p_{\mathsf{G}}^{lb,\min}(F)\leqslant p_{\mathsf{M}}^{\min}(F)\leqslant p_{\mathsf{G}}^{ub,\min}(F)\quad\text{and}\quad p_{\mathsf{G}}^{lb,\max}(F)\leqslant p_{\mathsf{M}}^{\max}(F)\leqslant p_{\mathsf{G}}^{ub,\max}(F)\qquad(1)$$

where, for example, in the stochastic game G:

$$p_{\mathsf{G}}^{lb,\max}(F)\stackrel{\text{def}}{=}\sup_{S'\in\overline{\mathcal{P}}}\inf_{\sigma_1}\sup_{\sigma_2}p_{S'}^{\sigma_1,\sigma_2}(F)$$
$$p_{\mathsf{G}}^{ub,\max}(F)\stackrel{\text{def}}{=}\sup_{S'\in\overline{\mathcal{P}}}\sup_{\sigma_1}\sup_{\sigma_2}p_{S'}^{\sigma_1,\sigma_2}(F)$$

and $p_{S'}^{\sigma_1,\sigma_2}(F)$ denotes the probability of reaching the target $\alpha(F)$ under the pair of strategies (σ_1,σ_2) when starting in the state S'. These reachability values can be determined efficiently using *value iteration* [16] together with the corresponding adversary or strategy-pair which achieves the value. In [17], the game-based abstraction of Definition 3 above is phrased in terms of abstract interpretation [18] and the resulting bounds obtained are shown to coincide with the "best" values obtainable for a fixed abstraction of the MDP.

Quantitative abstraction refinement [5, 8, 10] is an approach for automatically constructing abstractions of probabilistic models. Using the notion of abstracting MDPs with stochastic games describe above, it has been successfully applied to the verification of a probabilistic extension of ANSI-C [8], probabilistic timed automata [5] and concurrent probabilistic systems [17].

Illustrated in Figure 1, the technique starts with an MDP M (or, in practice a high-level model with MDP semantics) and coarse partition $\mathcal{P}$ of its state space. It then constructs and analyses the resulting abstraction of M, yielding lower and upper bounds on a property of interest (e.g. the probability of reaching a set of target states F as in (1) above). If the difference between these bounds (the "error") is below a pre-specified bound ε, the process terminates, producing suitably tight lower and upper bounds. If not, the abstraction is refined, based on information (strategies) from the analysis of the game. The abstraction, analysis and refinement loop is repeated until the error drops below ε.

2.3 Clocks, Zones, Variables and Predicates

Clocks. Let $\mathcal{X}$ be a finite set of clocks. A function $v : \mathcal{X} \to \mathbb{R}$ is referred to as a *clock valuation* and the set of all clock valuations is denoted by $\mathbb{R}^{\mathcal{X}}$. For $v \in \mathbb{R}^{\mathcal{X}}$, $t \in \mathbb{R}$ and $X \subseteq \mathcal{X}$, we use $v{+}t$ to denote the valuation which increments all clocks by t and $v[X{:=}0]$ for the valuation in which clocks in X are reset to 0.

Zones. The set of zones of $\mathcal{X}$, written $\mathit{Zones}(\mathcal{X})$, is defined by the syntax:

$$\zeta ::= \texttt{true} \mid x \leqslant d \mid c \leqslant x \mid x{+}c \leqslant y{+}d \mid \neg\zeta \mid \zeta \vee \zeta$$

where $x, y \in \mathcal{X}$ and $c, d \in \mathbb{N}$. A zone ζ represents the set of clock valuations v which *satisfy* ζ, denoted $v \triangleleft \zeta$, i.e. those where ζ resolves to `true` by substituting each clock x with $v(x)$. We will use several classical operations on zones [19, 20]:

- $\nearrow\zeta$ contains all valuations that can be reached from ζ by letting time pass;
- $\swarrow\zeta$ contains all valuations that can reach ζ by letting time pass;
- $[X{:=}0]\zeta$ contains the valuations which are in ζ after resetting the clocks X;
- $\zeta[X{:=}0]$ contains the valuations obtained from ζ by resetting the clocks X.

In standard fashion, we restrict our attention to *c-closed* zones, in which constraints with bounds greater than c are removed, and where c is the largest such bound appearing in the description of the model under study.

Data Variables and Predicates. Let $\mathcal{D}$ be a finite set of data variables. We denote by $\mathit{Val}(\mathcal{D})$ the set of *data valuations* over $\mathcal{D}$ and by $\mathit{Up}(\mathcal{D})$ the set of *updates*, i.e. the set of functions $\mathit{up} : \mathit{Val}(\mathcal{D}) \to \mathit{Val}(\mathcal{D})$. Let $\mathit{Pred}(\mathcal{D})$ be the set of predicates over the data variables $\mathcal{D}$. For a data predicate ϕ and valuation u, we say $u \triangleleft \phi$ if ϕ holds after substituting each variable d with the value $u(d)$.

3 Probabilistic Timed Programs

We now introduce the formal model of *probabilistic timed programs* (PTPs), on which the techniques in this paper are based. These combine:

- timed behaviour, through real-valued clocks in the style of timed automata;
- stochastic behaviour, through discrete probabilistic choice;
- nondeterminism and concurrency through parallel composition;
- control flow and discrete data variables to capture program behaviour.

PTPs are essentially probabilistic timed automata (PTAs) [21–23] with the addition of discrete-valued variables. For timed automata formalisms, discrete variables are typically considered to be a straightforward syntactic extension since their values can simply be encoded into locations. Our focus in this paper is how to use abstraction when such an encoding is not feasible in practice.

Definition 4. *A* probabilistic timed program *(PTP) is a tuple of the form* $\mathsf{P}{=}(L, \overline{l}, \mathcal{D}, \overline{u}, \mathcal{X}, \mathit{Act}, \mathit{inv}, \mathit{enab}, \mathit{prob})$ *where:*

- L *is a finite set of* locations *and* $\overline{l} \in L$ *is an* initial location*;*
- $\mathcal{D}$ *is a finite set of* data variables *and* $\overline{u} \in \mathit{Val}(\mathcal{D})$ *is an initial data valuation;*

- $\mathcal{X}$ *is a finite set of* clocks*;*
- Act *is a finite set of actions;*
- $inv : L \to Zones(\mathcal{X})$ *is an* invariant condition*;*
- $enab : L \times Act \to Pred(\mathcal{D}) \times Zones(\mathcal{X})$ *is an* enabling condition*;*
- $prob : L \times Act \to \mathsf{Dist}(Up(\mathcal{D}) \times 2^{\mathcal{X}} \times L)$ *is a* probabilistic transition function.

We use $enab_{\mathcal{D}}$ and $enab_{\mathcal{X}}$ to denote the data and time components of the enabling function, i.e. $enab(l,a) = (enab_{\mathcal{D}}(l,a), enab_{\mathcal{X}}(l,a))$. A *state* of a PTP is a tuple $(l,u,v) \in L \times Val(\mathcal{D}) \times \mathbb{R}^{\mathcal{X}}$ such that $v \triangleleft inv(l)$. In a state (l,u,v), a certain amount of time $t \in \mathbb{R}$ can elapse, after which an action $a \in Act$ is performed. The choice of t requires that, while time passes, the invariant $inv(l)$ remains continuously satisfied. An action a can be chosen only if it is *enabled*, that is, the predicate-zone pair $enab(l,a)$ is satisfied by $(u, v+t)$. Once an action a is chosen, the update of the data variables, clocks to reset and successor location are selected at random, according to the distribution $prob(l,a)$. We call each element (l,a,up,X,l') such that $(up,X,l') \in Up(\mathcal{D}) \times 2^{\mathcal{X}} \times L$ is in the support of $prob(l,a)$ an *edge* and, for convenience, assume that the set of such edges, denoted $edges(l,a)$, is an ordered list $\langle e_1, \ldots, e_n \rangle$.

Definition 5. *Let* $\mathsf{P} = (L, \overline{l}, \mathcal{D}, \overline{u}, \mathcal{X}, Act, inv, enab, prob)$ *be a PTP. The semantics of* P *is an (infinite-state) MDP* $[\![\mathsf{P}]\!] = (S, \overline{S}, \mathbb{R} \times Act, Steps_{\mathsf{P}})$ *where:*

- $S = \{(l,u,v) \in L \times Val(\mathcal{D}) \times \mathbb{R}^{\mathcal{X}} \mid v \triangleleft inv(l)\}$ *and* $\overline{S} = \{(\overline{l}, \overline{u}, \mathbf{0})\}$*;*
- $Steps_{\mathsf{P}}((l,u,v),(t,a)) = \lambda$ *if and only if*
  - $v+t' \triangleleft inv(l)$ *for all* $0 \leqslant t' \leqslant t$*;*
  - $(u, v+t) \triangleleft enab(l,a)$*;*
  - *for any* $(l',u',v') \in S$*:*

$$\lambda(l',u',v') = \textstyle\sum \{\!|\, prob(l,a)(up,X,l') \mid (up,X) \in Up_{u \mapsto u'} \times \mathcal{X}_{v+t \mapsto v'} \,|\!\}$$

  where the set of updates $Up_{u \mapsto u'}$ *equals* $\{up \in Up(U) \mid up(u) = u'\}$ *and the set of clock resets* $\mathcal{X}_{v+t \mapsto v'}$ *is given by* $\{X \subseteq \mathcal{X} \mid (v+t)[X:=0] = v'\}$.

Each transition of the semantics of a PTP is a time-action pair (t,a), representing t time units elapsing, followed by a discrete a-labelled transition. For any state (l,u,v) and time-action pair (t,a), if $Steps_{\mathsf{P}}((l,u,v),(t,a))$ is defined and $edges(l,a) = \langle (l,a,up_1,X_1,l_1), \ldots, (l,a,up_n,X_n,l_n) \rangle$, then we write $(l,u,v) \xrightarrow{t,a} \langle (l_1, up_1(u), (v+t)[X_1:=0]), \ldots, (l_n, up_n(u), (v+t)[X_n:=0]) \rangle$.

The definition of parallel composition for PTPs is a straightforward extension of that for probabilistic timed automata [24], see [12] for details.

4 Abstraction of PTPs

We now consider the problem of constructing *abstractions* of PTPs, that is to say building a stochastic game abstraction [9] for its underlying MDP semantics. For this, we combine several different techniques. For the data part of PTPs, we use *predicate abstraction* [25]. For the time aspect, we consider two possibilities: first,

we again use predicates (over clock, rather than data, variables); secondly we use *zones*, which can be efficiently stored and manipulated using difference-bound matrices (DBMs) [19, 20]. The latter is better suited to the *forwards reachability* based techniques commonly used for timed automata (and proposed for the construction of abstractions of PTAs in [5]).

We thus have two abstractions to consider: (i) using predicates for both data and time; (ii) using predicates for data and zones for time. A crucial difference between the two is that (i) induces a *partition* of the PTP states S (in which case, Definition 3 applies directly), whereas (ii) gives instead a *covering* of S.

4.1 Abstract Domains for PTPs

We formally define the two different abstractions discussed above using the notion of *abstract domains* from the abstract interpretation framework of [18].

Definition 6. *For a given set of concrete states S, an* abstract domain $\mathtt{A}$ *is a tuple* $((\mathtt{Z}, \sqcup, \sqcap, \sqsubseteq), \alpha, \gamma)$ *where:*

- $(\mathtt{Z}, \sqcup, \sqcap, \sqsubseteq)$ *is a lattice of* abstract states*;*
- $\alpha : 2^S \rightarrow \mathtt{Z}$ *and* $\gamma : \mathtt{Z} \rightarrow 2^S$ *are* abstraction *and* concretisation *functions;*

such that (α, γ) form a Galois connection.

From this point on, we assume sets of data and clock predicates $\Phi = \{\phi_1, \ldots, \phi_n\} \subseteq \mathit{Pred}(\mathcal{D})$ and $\Psi = \{\psi_1, \ldots, \psi_m\} \subseteq \mathit{Pred}(\mathcal{X})$. For a predicate φ and valuation w (over $\mathcal{D}$ or $\mathcal{X}$), let $\varphi(w)$ denote the value of φ evaluated against w. For predicates $\Upsilon = \{\varphi_1, \ldots, \varphi_k\}$, let $\Upsilon(w)$ denote the predicate valuation $(\varphi_1(w), \ldots, \varphi_{|\Upsilon|}(w)) \in \mathbb{B}^\Upsilon$ and, for $b \in \mathbb{B}^\Upsilon$, let $\Upsilon[b]$ denote the predicate $\tilde{\varphi}_1 \wedge \ldots \wedge \tilde{\varphi}_k$ where $\tilde{\varphi}_i = \varphi_i$ if $b_i = \mathtt{true}$ and $\neg\varphi_i$ otherwise. Note that $\Psi[b]$ can be considered as a zone.

The Abstract Domain $\mathtt{A}^{\Phi,\Psi}$. The atoms of the lattice $(\mathtt{Z}^{\Phi,\Psi}, \sqcup, \sqcap, \sqsubseteq)$ are the tuples $\mathtt{z} = (l, b_1, b_2) \in L \times \mathbb{B}^\Phi \times \mathbb{B}^\Psi$, comprising a location l and predicate valuations b_1 and b_2. Since $L \times \mathbb{B}^\Phi \times \mathbb{B}^\Psi$ form the atoms of the lattice, the operations $\sqcup$, $\sqcap$ and $\sqsubseteq$ can be considered as the standard set operators over $2^{L \times \mathbb{B}^\Phi \times \mathbb{B}^\Psi}$. For any $S' \subseteq S$ and $(l, b_1, b_2) \in L \times \mathbb{B}^\Phi \times \mathbb{B}^\Psi$ the abstraction and concretisation functions are defined as follows:

$$\alpha(S') = \sqcup_{(l,u,v) \in S'} (l, \Phi(u), \Psi(v)) \text{ and } \gamma(l, b_1, b_2) = \{(l, u, v) \mid \Phi(u) = b_1 \wedge \Psi(v) = b_2\}.$$

The Abstract Domain $\mathtt{A}^{\Phi,\mathcal{X}}$. A basis for the lattice $(\mathtt{Z}^{\Phi,\mathcal{X}}, \sqcup, \sqcap, \sqsubseteq)$ are the tuples $\mathtt{z} = (l, b, \zeta) \in L \times \mathbb{B}^\Phi \times \mathit{Zones}(\mathcal{X})$ comprising a location l, predicate valuation b and zone ζ. For $(l, b, \zeta), (l', b', \zeta') \in L \times \mathbb{B}^\Phi \times \mathit{Zones}(\mathcal{X})$:

$$(l, b, \zeta) \sqcup (l', b', \zeta') = \textbf{if } (l, b) = (l, b') \textbf{ then } \{(l, b, \zeta \vee \zeta')\} \textbf{ else } \{(l, b, \zeta), (l', b', \zeta')\}$$
$$(l, b, \zeta) \sqcap (l', b', \zeta') = \textbf{if } (l, b) = (l, b') \textbf{ then } \{(l, b, \zeta \wedge \zeta')\} \textbf{ else } \emptyset$$

and $(l, b, \zeta) \sqsubseteq (l', b', \zeta')$ if and only if $(l, b) = (l', b')$ and $\zeta \Rightarrow \zeta' \equiv \mathtt{true}$, while $\mathtt{Z}' \sqsubseteq \mathtt{Z}''$ if and only if for all $\mathtt{z}' \in \mathtt{Z}'$ there exists $\mathtt{z}'' \in \mathtt{Z}''$ such that $\mathtt{z}' \sqsubseteq \mathtt{z}''$. We illustrate these operations with the following examples:

$$\{(l,b,x\leqslant 4)\} \sqcup \{(l,b,y\geqslant 1),(l',b,x\geqslant 4)\} = \{(l,b,(x\leqslant 4)\vee(y\geqslant 1)),(l',b,x\geqslant 4)\}$$
$$\{(l,b,x\leqslant 4)\} \sqcap \{(l,b,x\geqslant 2)\} = \{(l,b,2\leqslant x\leqslant 4)\}$$
$$\{(l,b,x\leqslant 1),(l',b',y\geqslant 3)\} \sqsubseteq \{(l,b,(x\leqslant 4),(l',b',y\geqslant 2)),(l'',b'',z\leqslant 1)\}\,.$$

With regards to the abstraction and concretisation functions, we have:

$$\alpha(S') = \sqcup_{(l,u,v)\in S'}\ (l,\Phi(u),[v]) \text{ and } \gamma(l,b,\zeta) = \{(l,u,v) \mid \Phi(u)=b \ \wedge\ v \triangleleft \zeta\}\,.$$

where $[v]$ is the *clock equivalence class* (or clock region) of v [26].

For both abstract domains we use $loc(\mathtt{z})$, $data(\mathtt{z})$ and $time(\mathtt{z})$ to denote the different components of the tuple $\mathtt{z}$ representing an abstract state.

4.2 Abstract Post Operators

We now describe how to construct an *abstract post operator* for each abstract domain, i.e. the "best" abstraction [18] of the concrete PTP transition semantics. For convenience, we split the *post* operator into two parts, representing the elapse of time in the current location l and the subsequent discrete transition along edge $e=(l,a,up,X,l')$. For any set $S'\subseteq S$ of concrete states, we have:

$$\mathsf{tpost}[l](S') = \{(l,u,v+t) \mid (l,u,v)\in S' \ \wedge\ t\in\mathbb{R} \ \wedge\ \forall t'\leqslant t.\ (v+t')\triangleleft inv(l)\}$$
$$\mathsf{dpost}[e](S') = \{(l,up(u),v[X:=0]) \mid (l,u,v)\in S' \wedge (u,v)\triangleleft enab(l,a)\}$$

For the abstract domain $\mathtt{A}=((\mathtt{Z},\sqcup,\sqcap,\sqsubseteq),\alpha,\gamma)$, the corresponding "best" *abstract time-post* and *discrete-post* operators are given by:

$$\mathsf{tpost}^{\mathtt{A}}[l](\mathtt{z}) = \alpha(\mathsf{tpost}[l](\gamma(\mathtt{z}))) \quad \text{and} \quad \mathsf{dpost}^{\mathtt{A}}[e](\mathtt{z}) = \alpha(\mathsf{dpost}[e](\gamma(\mathtt{z})))\,.$$

For both of the abstract domains introduced in the previous section, these operators can be efficiently computed, using DBMs to manipulate zones [19, 20] and SAT or SMT based techniques for predicates over data variables [27].

When representing clock valuations as zones, the elapse of time in l is captured by the function $\mathsf{tpost}_{\mathcal{X}}[l] : Zones(\mathcal{X})\rightarrow Zones(\mathcal{X})$ and the effect of edge e by $\mathsf{dpost}_{\mathcal{X}}[e](\zeta) : Zones(\mathcal{X})\rightarrow Zones(\mathcal{X})$. Both can computed with simple and efficient zone operations that can be implemented with DBMs. For $\zeta\in Zones(\mathcal{X})$:

$$\mathsf{tpost}_{\mathcal{X}}[l](\zeta) = inv(l)\wedge \nearrow\zeta$$
$$\mathsf{dpost}_{\mathcal{X}}[e](\zeta) = (\zeta\wedge enab_{\mathcal{X}}(l,a))[X:=0]\wedge inv(l')\,.$$

For the representation of data using predicates, we define the function $\mathsf{dpost}^{\Phi}_{\mathcal{D}}[e] : \mathbb{B}^{\Phi}\rightarrow 2^{\mathbb{B}^{\Phi}}$ giving the set of possible predicate valuations of the variables in successor states when taking edge e. Let $p_1,\ldots,p_{|\Phi|}$ be Boolean variables corresponding to the predicates $\phi_1,\ldots,\phi_{|\Phi|}$ and $b\in\mathbb{B}^{\Phi}$ be a predicate valuation. Then $\mathsf{dpost}^{\Phi}_{\mathcal{D}}[e](b)$ contains all satisfying instances of $p_1,\ldots,p_{|\Phi|}$ such that:

$$\exists u,u'\in Val(\mathcal{D}).\ up(u)=u'\wedge\Phi(u)=b\wedge\left(p_1\Leftrightarrow\phi_1(u')\wedge\cdots\wedge p_{|\Phi|}\Leftrightarrow\phi_{|\Phi|}(u')\right)$$

which we use an SMT solver to enumerate. This assumes that the types of variables in $\mathcal{D}$ and the operations occurring in *up* and predicates ϕ_i match the underlying theory of the SMT solver used. Alternatively, a bit-level encoding of data variables can be employed, and a SAT solver used for enumeration [28].

On the other hand, $\mathsf{tpost}^{\Psi}_{\mathcal{X}}[l] : \mathbb{B}^{\Psi} \to 2^{\mathbb{B}^{\Psi}}$ and $\mathsf{dpost}^{\Psi}_{\mathcal{X}}[e] : \mathbb{B}^{\Psi} \to 2^{\mathbb{B}^{\Psi}}$, given b_2 as input, return the satisfiable instances b_2' of the predicates:

$$\Psi[b_2'] \wedge \mathsf{tpost}_{\mathcal{X}}[l](\Psi[b_2]) \quad \text{and} \quad \Psi[b_2'] \wedge \mathsf{dpost}_{\mathcal{X}}[e](\Psi[b_2])\,.$$

Using these functions, we are able to efficiently compute the abstract post operators for both domains. More precisely, for any $(l, b_1, b_2) \in \mathsf{Z}^{\Phi,\Psi}$:

$$\begin{aligned}\mathsf{tpost}^{\Phi,\Psi}[l](l, b_1, b_2) &= \{(l, b_1, b_2') \mid b_2' \in \mathsf{tpost}^{\Psi}_{\mathcal{X}}[l](b_2)\}\\ \mathsf{dpost}^{\Phi,\Psi}[e](l, b_1, b_2) &= \{(l', b_1', b_2') \mid b_1' \in \mathsf{dpost}^{\Phi}_{\mathcal{D}}[e](b) \wedge b_2' \in \mathsf{dpost}^{\Psi}_{\mathcal{X}}[e](b_2)\}\end{aligned}$$

and for any $(l, b, \zeta) \in \mathsf{Z}^{\Phi,\mathcal{X}}$:

$$\begin{aligned}\mathsf{tpost}^{\Phi,\mathcal{X}}[l](l, b, \zeta) &= \{(l, b, \mathsf{tpost}_{\mathcal{X}}[l](\zeta))\}\\ \mathsf{dpost}^{\Phi,\mathcal{X}}[e](l, b, \zeta) &= \{(l', b', \mathsf{dpost}_{\mathcal{X}}[e](\zeta)) \mid b' \in \mathsf{dpost}^{\Phi}_{\mathcal{D}}[e](b)\}\,.\end{aligned}$$

4.3 Abstract Reachability Graphs

The abstract post operators of the previous section can be used to construct an *abstract reachability graph*. This generalises the approach taken in [5] for PTAs. In this section, for a given abstract domain, we formally define the concept of an abstract reachability graph, describe how it can be used to construct a stochastic game abstraction of a PTP, and then how to build such a graph. We fix a PTP P and abstract domain $\mathtt{A}=((\mathsf{Z}, \sqcup, \sqcap, \sqsubseteq), \alpha, \gamma)$ over P.

We begin by introducing the concept of *abstract transitions*. An abstract transition of P with respect to the abstract domain $\mathtt{A}$ takes the form:

$$\theta = \big(\mathsf{z}, a, \langle \mathsf{z}_1, \ldots, \mathsf{z}_n \rangle\big) \in \mathsf{Z} \times \mathit{Act} \times \mathsf{Z}^+$$

where $n = |\mathit{edges}(\mathit{loc}(\mathsf{z}), a)|$. Intuitively, θ represents the possibility of, from a PTP state in $\gamma(\mathsf{z})$, letting time pass, then taking action a and, for each edge $e_i = (l, a, up_i, X_i, l_i) \in \mathit{edges}(\mathit{loc}(\mathsf{z}), a)$, reaching a state in $\gamma(\mathsf{z}_i)$. The notion of *validity* for an abstract transition expresses the fact that a corresponding concrete transition actually exists. More precisely, we define:

$$\mathit{valid}(\theta) \stackrel{\text{def}}{=} \left\{ s \in \gamma(\mathsf{z}) \mid \exists t \in \mathbb{R}.\ \left(s \xrightarrow{t,a} \langle s_1, \ldots, s_n \rangle \wedge \forall 1 \leqslant i \leqslant n.\ s_i \in \gamma(\mathsf{z}_i) \right) \right\}$$

as the set of PTP states from which such a transition is possible, and we say that θ is *valid* if the set $\mathit{valid}(\theta)$ is non-empty.

We now explain how validity can be checked for the abstract domains $\mathtt{A}^{\Phi,\Psi}$ and $\mathtt{A}^{\Phi,\mathcal{X}}$. For simplicity, we will split the computation by considering validity with respect to the data and time components of an abstract transition $\theta =$

$(\mathtt{z}, a, \langle \mathtt{z}_1, \ldots, \mathtt{z}_n \rangle)$. More precisely, we let $valid_{\mathcal{D}}(\theta) \subseteq Val(\mathcal{D})$ and $valid_{\mathcal{X}}(\theta) \subseteq \mathbb{R}^{\mathcal{X}}$ denote the sets of data valuations satisfying $data(\mathtt{z})$ and clock valuations satisfying $time(\mathtt{z})$ from which it is possible to let time pass and perform the action a such that taking the ith edge e_i gives a state in $\mathtt{z}_i$. An abstract transition is then *valid* if the valid sets for both data and time are nonempty and moreover:

$$valid(\theta) = \{(l, v, x) \in \gamma(\mathtt{z}) \mid l = loc(\mathtt{z}) \wedge v \in valid_{\mathcal{D}}(\theta) \wedge x \in valid_{\mathcal{X}}(\theta)\}\,.$$

For the data component, $valid_{\mathcal{D}}(\theta)$ is characterised by the formula:

$$\Phi[data(\mathtt{z})] \wedge \big(enab_{\mathcal{D}}(loc(\mathtt{z}), a) \wedge (\wedge_{i=1}^{n} (\mathsf{wp}[up_i](\Phi[data(\mathtt{z}_i)]))) \big)$$

where $\mathsf{wp}[up_i]$ denotes the weakest precondition for update up_i. Thus, like in the previous section, we can check emptiness of $valid_{\mathcal{D}}(\theta)$ via a satisfiability check of the above formula using an SMT/SAT solver.

For the time component, we can compute $valid_{\mathcal{X}}(\theta)$ as a zone. For abstract domain $\mathtt{A}^{\Phi,\mathcal{X}}$, where abstract states contain zones, $valid_{\mathcal{X}}(\theta)$ is the zone:

$$time(\mathtt{z}) \wedge \swarrow \big(enab_{\mathcal{X}}(loc(\mathtt{z}), a) \wedge (\wedge_{i=1}^{n} ([X_i{:=}0] time(\mathtt{z}_i))) \big)\,.$$

Checking its emptiness is a simple DBM operation. For abstract domain $\mathtt{A}^{\Phi,\Psi}$, we can perform the same computation after first converting predicate valuations to zones, i.e. replacing $time(\cdot)$ with $\Psi[time(\cdot)]$ in the above.

We are now in a position to define abstract reachability graphs for PTPs.

Definition 7. *An* abstract reachability graph *for PTP* P *with respect to target locations* F *and abstract domain* $\mathtt{A}{=}((\mathtt{Z}, \sqcup, \sqcap, \sqsubseteq), \alpha, \gamma)$*, is a tuple* $(\mathtt{Y}, \mathtt{R})$ *where:*

- $\mathtt{Y} \subseteq \mathtt{Z}$ *is a covering multiset of abstract states, i.e.* $S \subseteq \cup_{\mathtt{z} \in \mathtt{Y}} \gamma(\mathtt{z})$*;*
- $\mathtt{R} \subseteq \mathtt{Y} \times Act \times \mathtt{Y}^{+}$ *is a set of* valid *abstract transitions;*

such that, if $\mathtt{z} \in \mathtt{Y}$*,* $loc(\mathtt{z}) \notin F$*,* $s \in \gamma(\mathtt{z})$ *and* $s \xrightarrow{t,a} \langle s_1, \ldots, s_n \rangle$*, then* $\mathtt{R}$ *contains an abstract transition* $(\mathtt{z}, a, \langle \mathtt{z}_1, \ldots, \mathtt{z}_n \rangle)$ *such that* $s_i \in \gamma(\mathtt{z}_i)$ *for all* $1 \leqslant i \leqslant n$*.*

An abstract transition $\theta = (\mathtt{z}, a, \langle \mathtt{z}_1, \ldots, \mathtt{z}_n \rangle)$ induces the probability distribution λ_θ over the abstract states $\mathtt{Z}$ where for any $\mathtt{z}' \in \mathtt{Z}$:

$$\lambda_\theta(\mathtt{z}') \stackrel{\text{def}}{=} \textstyle\sum_{i=1}^{n} \{\!|\, prob(l, a)(e_i) \mid \mathtt{z}_i{=}\mathtt{z}' \,|\!\}\,.$$

We now extend the notion of *validity* for the data and time components of abstract transitions to sets of abstract transitions with the same source. For any abstract state $\mathtt{z} \in \mathtt{Z}$ and set of abstract transitions $\Theta \subseteq \mathtt{R}(\mathtt{z})$, let:

$$valid_{\mathcal{D}}(\Theta) \stackrel{\text{def}}{=} (\cap_{\theta \in \Theta} valid_{\mathcal{D}}(\theta)) \setminus \big(\cup_{\theta \in \mathtt{R}(\mathtt{z}) \setminus \Theta} valid_{\mathcal{D}}(\theta)\big) \tag{2}$$

$$valid_{\mathcal{X}}(\Theta) \stackrel{\text{def}}{=} (\cap_{\theta \in \Theta} valid_{\mathcal{X}}(\theta)) \setminus \big(\cup_{\theta \in \mathtt{R}(\mathtt{z}) \setminus \Theta} valid_{\mathcal{X}}(\theta)\big)\,. \tag{3}$$

By construction, $valid_{\mathcal{D}}(\Theta)$ and $valid_{\mathcal{X}}(\Theta)$ identify precisely the data valuations u satisfying $data(\mathtt{z})$ and the clock valuations v satisfying $time(\mathtt{z})$, such that, from $(loc(\mathtt{z}), u, v)$, it is possible to perform the transition encoded by any abstract

BuildGame(P, $(\mathtt{Y}, \mathtt{R})$, $\mathtt{A}$)

1 $\overline{\mathtt{Y}} = \{\mathtt{y} \mid (\overline{l}, \overline{u}, \mathbf{0}) \in \gamma_{\mathtt{A}}(\mathtt{y})\}$
2 **for** $\mathtt{y} \in \mathtt{Y}$
3 **for** $\Theta \subseteq \mathtt{R}(\mathtt{y})$ **such that** Θ is valid
4 $Steps_{\mathsf{G}}(\mathtt{y}, \Theta) := \{\lambda_\theta \mid \theta \in \Theta\}$
5 **return** $\mathsf{G} = (\mathtt{Y}, \overline{\mathtt{Y}}, 2^{\mathtt{R}}, Steps_{\mathsf{G}})$

Fig. 2. Algorithm for building abstraction from reachability graph

transition $\theta \in \Theta$, but it is not possible to perform a transition encoded by any other abstract transition of $\mathtt{R}(\mathtt{z})$. We say a set of abstract transitions $\Theta \subseteq \mathtt{R}(\mathtt{z})$ is valid if either $valid_{\mathcal{D}}(\Theta)$ or $valid_{\mathcal{X}}(\Theta)$ hold.

We use the approach of [9] to represent an abstraction of an MDP as a stochastic two-player game. The basic idea is that the two players in the game represent nondeterminism introduced by the abstraction and nondeterminism from the original model. In an abstract state $\mathtt{z}$ of the game abstraction of a PTP, player 1 first picks a PTP state $(l, u, v) \in \gamma(\mathtt{z})$ and then player 2 makes a choice over the actions that become enabled after letting time pass from (l, u, v).

The algorithm BuildGame in Figure 2 describes how to construct for a PTP P, from a reachability graph $(\mathtt{Y}, \mathtt{R})$ over an abstract domain $\mathtt{A}$, a stochastic game. In a state $\mathtt{y}$ of the game, player 1 chooses between any *valid* set of abstract transitions $\Theta \subseteq \mathtt{R}(\mathtt{y})$. Player 2 then selects an abstract transition $\theta \in \Theta$. As the following result demonstrates, this game yields lower and upper bounds on both the minimum and maximum reachability probabilities of the PTP.

Theorem 1. *Let* P *be a PTP with target locations* F *and* $\mathtt{A}$ *an abstract domain over* P*. If* $(\mathtt{Y}, \mathtt{R})$ *is a reachability graph for* P *with respect to* F *and* $\mathtt{A}$*, then*

$$p_{\mathsf{G}}^{lb,*}\{\mathtt{y} \in \mathtt{Y} \mid loc(\mathtt{y}) \in F\} \leqslant p_{\mathsf{P}}^{*}(F) \leqslant p_{\mathsf{G}}^{ub,*}\{\mathtt{y} \in \mathtt{Y} \mid loc(\mathtt{y}) \in F\}$$

where G *is returned by* BuildGame(P,$(\mathtt{Y},\mathtt{R})$,$\mathtt{A}$) *(see Figure 2) and* $* \in \{\min, \max\}$.

Finally, we describe the process of constructing the reachability graph for a PTP. A generic reachability graph generation algorithm is given in Figure 3 which takes as input a PTP, target set of locations and abstract domain. The following theorem states the correctness of this algorithm.

Theorem 2. *Let* P *be a PTP with target locations* F *and* $\mathtt{A}$ *an abstract domain over* P*. If* $(\mathtt{Y}, \mathtt{R})$ *is returned by* BuildReachGraph($\mathsf{P}, F, \mathtt{A}$)*, then* $(\mathtt{Y}, \mathtt{R})$ *is a reachability graph of* P *with respect to* F *and* $\mathtt{A}$*.*

The following proposition demonstrates that, for the abstract domain $\mathtt{A}^{\Phi,\Psi}$, the resulting abstraction corresponds to the one generated using the approach of [9], applied to the (infinite-state) MDP semantics of a PTP based on the partition of the state space induced from the predicates Φ and Ψ.

Proposition 1. *Let* P *be a PTP with target locations* F *and abstract domain* $\mathtt{A}^{\Phi,\Psi}$*. If* $(\mathtt{Y}, \mathtt{R})$ *is the reachability graph returned by* BuildReachGraph($\mathsf{P}, F, \mathtt{A}^{\Phi,\Psi}$)

BuildReachGraph($\mathsf{P}, F, \mathtt{A}$)

```
1  Y := ∅
2  X := α(l̄, ū, 0)
3  while X ≠ ∅
4    choose x ∈ X
5    l := loc(x)
6    X := X \ {x}
7    Y := Y ∪ {x}
8    for z ∈ tpost^A(x)
9      for a ∈ Act such that enab(l, a) ≠ false
10       for e_i = (l, a, up_i, X_i, l_i) ∈ edges(l, a) = ⟨e_1, ..., e_n⟩
11         Z_i := dpost^A[e_i](z)
12         if l_i ∉ F then X := X ∪ (Z_i \ Y)
13       for ⟨z_1, ..., z_n⟩ ∈ Z_1 × ··· × Z_n
14         R := R ∪ {(x, a, ⟨z_1, ..., z_n⟩)}
15 return (Y, R)
```

Fig. 3. Algorithm for reachability graph construction

(see Figure 3), then the game $\mathsf{BuildGame}(\mathsf{P}, (\mathtt{Z}, \mathtt{R}), \mathtt{A}^{\Phi,\Psi})$ *equals that constructed by Definition 3 for the MDP* $[\![\mathsf{P}]\!]$ *(after the states with locations in* F *are made absorbing) when using the partition* $\mathcal{P} = \{\gamma(\mathtt{z}) \mid \mathtt{z} \in \mathtt{Z}^{\Phi,\Psi}\}$.

5 Abstraction Refinement for PTPs

The previous section described how to compute the abstraction of a PTP for two types of abstract domains, $\mathtt{A}^{\Phi,\Psi}$ and $\mathtt{A}^{\Phi,\mathcal{X}}$. To implement a quantitative abstraction refinement scheme similar to that described in Section 2.2, we also require *refinement* techniques for automatically constructing more precise abstractions. In practice, this means splitting one or more abstract states. There are two forms of refinement, either in terms of zones or predicates. In both cases, the refinement process works by modifying the abstract reachability graph. However, in the former, we will split abstract states by breaking up the zone component of an abstract state, while for the latter, we modify the abstract domain by adding a new data or clock predicate.

Choosing a State to Refine. Given an abstraction for a PTP P with target locations F, i.e. a reachability graph $(\mathtt{Y}, \mathtt{R})$ with respect to some abstract domain $\mathtt{A}$, the refinement approach is guided by the analysis of the corresponding stochastic game, i.e. that generated by $\mathsf{BuildGame}(\mathsf{P}, (\mathtt{Y}, \mathtt{R}), \mathtt{A})$. More precisely, given the bounds for the probability of reaching F and player 1 strategies that attain these bounds, we look at a single abstract state $\mathtt{z}$ for which the bounds differ and for which distinct player 1 strategies yield each bound.[1] In state $\mathtt{z}$, a player 1 strategy chooses an action available in $\mathtt{z}$, which, by construction, is a

[1] From the results of [9] such a state exists when the bounds differ in some state.

RefineZone$(\mathsf{P}, (\mathtt{Y}, \mathtt{R}), \mathtt{A}^{\Phi,\mathcal{X}}, \mathtt{z}, (\zeta_1, \ldots, \zeta_k))$

1 $\mathtt{Y}^{new} := \{(loc(\mathtt{z}), data(\mathtt{z}), \zeta_1), \ldots, (loc(\mathtt{z}), data(\mathtt{z}), \zeta_k))\}$
2 $\mathtt{Y}^{ref} := (\mathtt{Y} \setminus \{\mathtt{z}\}) \uplus \mathtt{Y}^{new}$
3 $\mathtt{R}^{ref} := \emptyset$
4 **for** $\theta = (\mathtt{z}_0, a, \langle \mathtt{z}_1, \ldots, \mathtt{z}_n \rangle) \in \mathtt{R}$
5 **if** $\mathtt{z} \notin \{\mathtt{z}_0, \mathtt{z}_1, \ldots, \mathtt{z}_n\}$ **then**
6 $\mathtt{R}^{ref} := \mathtt{R}^{ref} \cup \{\theta\}$
7 **else**
8 $\Theta^{new} := \{(\mathtt{z}'_0, a, \langle \mathtt{z}'_1, \ldots, \mathtt{z}'_n \rangle) \mid \mathtt{z}'_i \in \mathtt{Y}^{new}$ if $\mathtt{z}_i = \mathtt{z}$ and $\mathtt{z}'_i = \mathtt{z}_i$ o/wise$\}$
9 **for** $\theta^{new} \in \Theta^{new}$ **such that** $valid(\theta^{new}) \neq \emptyset$
10 $\mathtt{R}^{ref} := \mathtt{R}^{ref} \cup \{\theta^{new}\}$
11 **return** $(\mathtt{Y}^{ref}, \mathtt{R}^{ref})$

Fig. 4. Algorithm to perform zone refinement in abstract state $\mathtt{z}$

valid set of abstract transitions from $\mathtt{R}(\mathtt{z})$. Therefore, let $\Theta_{lb}, \Theta_{ub} \subseteq \mathtt{R}(\mathtt{z})$ denote the distinct player 1 strategy choices for the lower and upper bound respectively. Since both sets are valid, we have that either $valid_{\mathcal{D}}(\Theta_{lb})$ or $valid_{\mathcal{X}}(\Theta_{lb})$ is nonempty and either $valid_{\mathcal{D}}(\Theta_{ub})$ or $valid_{\mathcal{X}}(\Theta_{ub})$ is nonempty.

In the next section we show how to refine the zones when the abstract domain is of the form $\mathtt{A}^{\Phi,\mathcal{X}}$ and either $valid_{\mathcal{X}}(\Theta_{lb})$ or $valid_{\mathcal{X}}(\Theta_{ub})$ holds. Following this, we consider how to refine the predicates over $\mathcal{D}$ assuming either $valid_{\mathcal{D}}(\Theta_{lb})$ or $valid_{\mathcal{D}}(\Theta_{ub})$ hold. The remaining case (refining clock predicates when the abstract domain is of the form $\mathtt{A}^{\Phi,\Psi}$ and either $valid_{\mathcal{X}}(\Theta_{lb})$ or $valid_{\mathcal{X}}(\Theta_{ub})$ holds) follows similarly, see [12] for details.

Zone Refinement. In this case, the abstract domain is of the form $\mathtt{A}^{\Phi,\mathcal{X}}$ and either $valid_{\mathcal{X}}(\Theta_{lb})$ or $valid_{\mathcal{X}}(\Theta_{ub})$ holds. Since the time validity conditions of Θ_{lb} and Θ_{ub} identify precisely the clock valuations in the zone component $time(\mathtt{z})$ of $\mathtt{z}$ for which the corresponding transitions of $[\![\mathsf{P}]\!]$ are possible, we split the zone component of $\mathtt{z}$ into:

$$valid_{\mathcal{X}}(\Theta_{lb}), \quad valid_{\mathcal{X}}(\Theta_{ub}) \quad \text{and} \quad time(\mathtt{z}) \wedge \neg(valid_{\mathcal{X}}(\Theta_{lb}) \vee valid_{\mathcal{X}}(\Theta_{ub}))\,. \tag{4}$$

Since at least one of $valid(\Theta_{lb})$ and $valid(\Theta_{ub})$ is non-empty and by construction $\Theta_{lb} \neq \Theta_{ub}$, using the definition of validity, it follows that the split of the zone $time(\mathtt{z})$ given in (4) produces a strict refinement.

The refinement algorithm is shown in Figure 4. It takes as input a PTP, abstract domain and set of zones and returns a new reachability graph for the PTP with respect to the same abstract domain. When using the algorithm the set $\{\zeta_1, \ldots, \zeta_k\}$ is given by the non-empty zones in (4). Lines 1–2 split the abstract state $\mathtt{z}$ based on the set of zones given as input, then, based on this splitting, lines 3–10 update the set of abstract transitions $\mathtt{R}$ resulting in a new reachability graph, for which the corresponding stochastic game is a refined abstraction of the PTP. The following result states the correctness of the zone refinement scheme.

RefineDataPredicate$(\mathsf{P}, (\mathtt{Y}, \mathtt{R}), \mathtt{A}^{\Phi,\star}, \phi')$

1 $\Phi^{ref} := \Phi \cup \{\phi'\}$
2 $\mathtt{A}^{ref} := \mathtt{A}^{\Phi^{ref},\star}$
3 $\mathtt{Y}^{ref} := \{\mathtt{y}^{ref} \in \mathtt{Z}^{\Phi^{ref},\star} \mid \exists \mathtt{y} \in \mathtt{Y}.\ \gamma_{\mathtt{A}^{ref}}(\mathtt{y}^{ref}) \subseteq \gamma_{\mathtt{A}}(\mathtt{y})\}$
4 $\mathtt{R}^{ref} := \emptyset$
5 **for** $\theta = (\mathtt{z}_0, a, \langle \mathtt{z}_1, \ldots, \mathtt{z}_n \rangle) \in \mathtt{R}$
6 $\quad \Theta^{new} := \{(\mathtt{z}'_0, a, \langle \mathtt{z}'_1, \ldots, \mathtt{z}'_n \rangle) \mid \gamma_{\mathtt{A}^{ref}}(\mathtt{z}'_i) \subseteq \gamma_{\mathtt{A}}(\mathtt{z}_i)$ for all $0 \leqslant i \leqslant n\}$
7 $\quad$ **for** $\theta^{new} \in \Theta^{new}$ **such that** $valid(\theta^{new}) \neq \emptyset$
8 $\quad\quad \mathtt{R}^{ref} := \mathtt{R}^{ref} \cup \{\theta^{new}\}$
9 **return** $(\mathtt{Z}^{ref}, \mathtt{R}^{ref}, \mathtt{A}^{ref})$

Fig. 5. Algorithm for refinement in terms of data predicates

Proposition 2. *Let* P *be a PTP with target locations* F, $\mathtt{A}^{\Phi,\mathcal{X}}$ *an abstract domain for* P, $(\mathtt{Y}, \mathtt{R})$ *a reachability graph for* P *with respect to* F *and* $\mathtt{A}^{\Phi,\mathcal{X}}$ *and* G *the game. If* $(\mathtt{Y}^{ref}, \mathtt{R}^{ref})$ *is returned by* RefineZone$(\mathsf{P}, (\mathtt{Y}, \mathtt{R}), \mathtt{A}^{\Phi,\mathcal{X}}, \mathtt{z}, \{\zeta_1, \ldots, \zeta_k\})$ *for* $\mathtt{z} \in \mathtt{Y}$ *and* $\zeta_1, \ldots, \zeta_k \in Zones(\mathcal{X})$ *and* G^{ref}=BuildGame$(\mathsf{P}, (\mathtt{Z}^{ref}, \mathtt{R}^{ref}), A^{\Phi,\mathcal{X}})$, *then:*

(i) $(\mathtt{Z}^{ref}, \mathtt{R}^{ref})$ *is a reachability graph for* P *with respect to* F *and* $\mathtt{A}^{\Phi,\mathcal{X}}$*;*
(ii) $p^{lb,*}_{\mathsf{G}}(\mathtt{Y}_F) \leqslant p^{lb,*}_{\mathsf{G}^{ref}}(\mathtt{Y}^{ref}_F)$ *and* $p^{ub,*}_{\mathsf{G}^{ref}}(\mathtt{Y}^{ref}_F) \leqslant p^{ub,*}_{\mathsf{G}}(\mathtt{Y}_F)$ *for* $* \in \{\min, \max\}$

where $\mathtt{Y}_F = \{\mathtt{y} \in \mathtt{Y} \mid loc(\mathtt{y}) \in F\}$ *and* $\mathtt{Y}^{ref}_F = \{\mathtt{y} \in \mathtt{Y}^{ref} \mid loc(\mathtt{y}) \in F\}$.

Data Predicate Refinement. In this case we suppose either $valid_{\mathcal{D}}(\Theta_{lb})$ or $valid_{\mathcal{D}}(\Theta_{ub})$ holds. The aim of the refinement is to remove the player 1 choices between Θ_{lb} and Θ_{ub}, so the simplest approach would be to add predicates representing either or both of the sets of data valuations $valid_{\mathcal{D}}(\Theta_{lb})$ and $valid_{\mathcal{D}}(\Theta_{ub})$. However, as can be seen from the definition of $valid_{\mathcal{D}}$ (see (2)), the corresponding predicates will be complex and therefore make the abstraction construction prohibitively expensive. Hence, we take a different approach outlined below.

By construction $\Theta_{lb} \neq \Theta_{ub}$, and hence, there exists an abstract transition $\theta' \in \mathtt{R}(\mathtt{z})$ such that either $\theta' \in \Theta_{lb} \backslash \Theta_{ub}$ or $\Theta_{ub} \backslash \Theta_{lb}$. We will show that if we refine by adding a predicate $\phi_{\theta'}$ representing $valid_{\mathcal{D}}(\theta')$, then we eliminate the player 1 choice between Θ_{lb} and Θ_{ub}. Without loss of generality we suppose $\theta' \in \Theta_{lb} \backslash \Theta_{ub}$, and therefore from (2) it follows that:

$$valid_{\mathcal{D}}(\Theta_{lb}) \subseteq valid_{\mathcal{D}}(\theta') \quad \text{and} \quad valid_{\mathcal{D}}(\Theta_{ub}) \cap valid_{\mathcal{D}}(\theta') = \emptyset\ .$$

By adding the predicate $\phi_{\theta'}$ it then follows that, if $\phi_{\theta'}$ is true in some abstract state, Θ_{ub} cannot be valid, while, if the predicate is false, Θ_{lb} cannot be valid. Thus, in any abstract state at most one of these choices can be valid and the choice between Θ_{lb} and Θ_{ub} has been eliminated.

The data predicate refinement algorithm is given in Figure 5 where ϕ' is a predicate over $\mathcal{D}$ (which in practice would correspond to the predicate $\phi_{\theta'}$). The lines 1–3 create the new abstract domain, while lines 4–8 create an abstract

reachability graph over the new abstract domain, for which the corresponding stochastic game is a refined abstraction of the PTP. The following demonstrates the correctness of the predicate refinement approach.

Proposition 3. *Let* P *be a PTP with target locations* F, $\mathtt{A}^{\Phi,\star}$ *an abstract domain for* P, $(\mathtt{Y},\mathtt{R})$ *a reachability graph for* P *with respect to* F *and* $\mathtt{A}^{\Phi,\star}$, *and* G *the game. If* $(\mathtt{Y}^{ref},\mathtt{R}^{ref},\mathtt{A}^{ref})$ *is returned by* $\mathsf{RefineDataPredicate}(\mathsf{P},(\mathtt{Y},\mathtt{R}),\mathtt{A}^{\Phi,\star},\varphi')$ *for any data predicate* φ', *and* G^{ref} *is the resulting game, then:*

(i) $(\mathtt{Z}^{ref},\mathtt{R}^{ref})$ *is a reachability graph for* P *with respect to* F *and* $\mathtt{A}^{ref}$;
(ii) $p^{lb,*}_{\mathsf{G}}(\mathtt{Y}_F) \leqslant p^{lb,*}_{\mathsf{G}^{ref}}(\mathtt{Y}^{ref}_F)$ *and* $p^{ub,*}_{\mathsf{G}^{ref}}(\mathtt{Y}^{ref}_F) \leqslant p^{ub,*}_{\mathsf{G}}(\mathtt{Y}_F)$ *for* $* \in \{\min,\max\}$

where $\mathtt{Y}_F = \{\mathtt{y} \in \mathtt{Y} \mid loc(\mathtt{y}) \in F\}$ *and* $\mathtt{Y}^{ref}_F = \{\mathtt{y} \in \mathtt{Y}^{ref} \mid loc(\mathtt{y}) \in F\}$.

Quantitative Abstraction Refinement. The refinement schemes presented above, combined with the techniques for abstraction given in the previous section, can be combined into a quantitative abstraction refinement loop. This provides fully automatic construction of abstractions for PTPs. We omit the details of the loop implementation, which are similar to [5]. See [12] for details.

6 Quantitative Verification of SystemC

We now describe a specific potential instantiation of the verification framework we have presented. In particular, we discuss its applicability to the quantitative verification of SystemC, a system-level modelling language that is becoming increasingly prominent in the development of embedded systems, e.g. for System-on-Chip (SoC) designs. Currently, analysis of SystemC designs is primarily performed using simulation; however, there is growing interest in applying verification techniques [29–34].

SystemC is appealing to designers because it is close enough to the hardware level to support synthesis to RTL (register transfer level) descriptions, but allows modelling of complex designs at a higher level of abstraction. Based on C++, it combines an imperative programming style, low-level data-types for hardware, an object-oriented approach to design and convenient high-level abstractions of concurrent communicating processes. Furthermore, systems can be efficiently simulated at the design stage.

System-on-Chip designs typically include many different components, including for example support for radio communications. Furthermore, these devices may then become integrated into larger, networked embedded systems. In these instances, reasoning about the behaviour of a SystemC design may need to take account of the inherently stochastic characteristics of the unreliable or unpredictable components that it interacts with. Another source of probabilistic behaviour is the use of randomisation. This is a key feature of, for example communication technologies like ZigBee, which are increasingly found in today's embedded devices.

Considered in its entirety, a quantitative analysis of SystemC requires all of the basic ingredients that we have proposed for probabilistic timed programs:

- **software:** basic process behaviour is defined in terms of C++ code, using a rich array of data types;
- **concurrency:** designs comprise multiple concurrent processes, communicating through message-passing primitives;
- **timing:** processes can be subjected to precisely timed delays, through interaction with the SystemC scheduler;
- **probability:** SystemC components may link to unpredictable devices, due to communication failures, unreliable components or randomisation.

The development of (non-quantitative) verification techniques for SystemC has already been identified as an important, but difficult, challenge [11]. Applying quantitative verification offers more powerful analysis techniques but promises to be even more demanding. In the remainder of this section, we outline how some of the existing approaches and tools for SystemC verification might be built upon to implement our framework. We then conclude by identifying some useful directions and challenges for future work.

Translating SystemC to PTPs. A SystemC design is decomposed into *modules*, representing the separate components within a design. Modules are connected, through *ports*, to *channels*, which model interactions between components. Built-in "primitive" channels such as signals, FIFOs and mutexes are provided. The behaviour of each module is described by a set of *threads* or *processes*, specified as C++ class methods.

In [34], a translation from SystemC to the timed automata based input language of UPPAAL is proposed and implemented. Our probabilistic timed program formalism is a superset of timed automata so the basic ideas can be used directly.[2] The approach of [34] is to translate each C++ method, representing a process or thread into a timed automaton. This is based on an extraction of the control flow graph: control vertices becomes locations, control flow edges become transitions and branching conditions (e.g. on `if` statements or `while` loops) are incorporated into the enabling conditions of transitions. The process of generating the control flow graph is facilitated by model extraction tools for SystemC like Scoot [35] and PINAPA [36]. These have been designed with a variety of applications in mind, including verification.

In order to ensure that the predicate abstraction techniques described in this paper can be applied to SystemC C++ code, the underlying SMT/SAT solvers used need to support the data-types and operations allowed in the language. Since SystemC offers low-level datatypes aimed at hardware designs, a SAT-based approach using a bit-level semantics is likely to be needed [28]. This approach was already applied to abstraction-based software model checking of SystemC in [30] and to a probabilistic extension of ANSI-C in [8].

Scheduling and Timed Behaviour. Concurrency between SystemC threads is controlled by the *scheduler*, whose behaviour is precisely defined in the language standard [37]. The SystemC scheduler is *co-operative* and *non-preemptive*:

[2] In practice, we need to add various syntactic niceties such as urgent and committed locations, and communication over channels, but this is relatively straightforward.

threads suspend themselves explicitly by calling a `wait()` or `wait(t)` function. The latter is an example of *timed* behaviour: the calling thread is suspended until it receives a timed notification from the scheduler after delay `t`.

The translation scheme of [34] captures the behaviour of the SystemC scheduler as a network of timed automata. This keeps track of which processes should be run in each part of each scheduler cycle. Uncertainty between the order in which multiple ready processes are executed is modelled as a nondeterministic choice. Delays in each process are handled by local clocks in each automaton.

Probabilistic Behaviour. As outlined above, it is often desirable to incorporate either randomisation or failures into SystemC models. For randomisation, this is likely to appear as calls to a C/C++ `rand()` function. Alternatively, as is done in [8], custom randomised functions could be added. In either case, these can be intercepted and converted to a probabilistic branch in the PTP. For failures, as in [8], SystemC code that corresponds to connections or communications with unreliable components can be replaced by a stub that captures the stochastic behaviour (e.g. using `rand()` as above). Probabilistic timed automata (PTAs) have already been applied to a large number of realistic case studies in which randomisation or failures are modelled in this fashion [22, 24, 38, 39].

Directions and Challenges. Implementing the verification techniques sketched in this section represents a considerable challenge. As ever, the most immediate difficulty is scalability: extending existing tools and techniques to handle the size and complexity of real SystemC designs. In this respect, it may be beneficial to consider state-of-the-art techniques for software model checking, which are not currently applied to the probabilistic case because they are non-trivial to adapt. These include, for example, the use of approximate abstractions and "lazy" construction of abstractions. One particular source of complexity in SystemC models is concurrency between processes. Development of software model checking techniques for concurrent programs is another very active field of research, that may yield gains in this area.

In a different direction, we may also aim to improve the expressivity of the PTP formalism proposed in this paper. In the current version, probability and time are largely orthogonal which, in many cases, is not a serious restriction for system modelling. However, it would be interesting to explore to what extent this can generalised. An extension with costs and rewards would be relatively straightforward. Looking further ahead, the use of languages like SystemC in embedded systems means that the digital components of the design will often interact with analogue devices. This would necessitate the use of more general, but less tractable, probabilistic models such as stochastic hybrid automata.

7 Related Work

For quantitative verification of probabilistic timed automata (PTAs), a variety of techniques [5, 21, 22, 24, 38, 39] and tools [4, 6, 7] have been produced. In [6], an extension of PTAs with discrete-valued variables, called VPTAs, is handled

via a translation to the basic case. Another interesting extension is *priced* PTAs [24, 40], which add a notion of prices (or weights) to locations and actions.

The development of abstraction and refinement techniques for probabilistic models such as MDPs is also an active area of research. This was first proposed in [41], using MDPs as abstractions, rather than stochastic games as in [5, 8, 10] (and this paper). In [42], MDP abstractions based on predicates are used to form a probabilistic CEGAR (counterexample-guided abstraction refinement) technique. Later work [17] adapted this to stochastic games. Other abstraction refinement frameworks for MDPs are put forward in [43] and [44].

Abstraction-refinement approaches have been proposed for non-probabilistic timed automata, e.g. [45], which uses bounded model checking and SAT-based techniques, [46], which is based on the region graph construction, and [47], for verifying PLC automata using UPPAAL [1]. Also related is [48], which applies SAT-based techniques to timed automata with data.

Finally, as highlighted in the previous section, there is an increasing amount of interest in developing verification techniques for SystemC [11]. A variety of existing verification techniques have been explored, including BDD-based model exploration [29], bounded model checking [31] and abstraction-refinement [30]. Another approach is to translate SystemC into other formalisms and languages for which tool support exists. This includes translations to Petri nets [32], Promela [33] and UPPAAL [34]. With the exception of [34], which models timing information, none of the above consider quantitative properties of SystemC.

8 Conclusions

We have outlined a theoretical framework for the verification of programs that exhibit both probabilistic and timed behaviour, based on quantitative abstraction refinement techniques. This represents the first steps towards quantitative verification of complex software systems, such as those found in the domain of embedded systems. We discussed some of the ongoing work and the challenges in this direction of research, using the SystemC language as an illustrative example.

Acknowledgements. The authors are supported in part by EPSRC grants EP/D07956X, EP/D076625 and EP/F001096, EU FP7 project CONNECT and ERC Advanced Grant VERIWARE.

References

1. Larsen, K., Pettersson, P., Yi, W.: UPPAAL in a nutshell. Int. Journal on Software Tools for Technology Transfer 1, 134–152 (1997)
2. Hinton, A., Kwiatkowska, M., Norman, G., Parker, D.: PRISM: A tool for automatic verification of probabilistic systems. In: Hermanns, H., Palsberg, J. (eds.) TACAS 2006. LNCS, vol. 3920, pp. 441–444. Springer, Heidelberg (2006)
3. Katoen, J.-P., Hahn, E., Hermanns, H., Jansen, D., Zapreev, I.: The ins and outs of the probabilistic model checker MRMC. In: Proc. QEST'09, IEEE Press, Los Alamitos (2009)

4. Uppaal-PRO, http://www.cs.aau.dk/~arild/uppaal-probabilistic/
5. Kwiatkowska, M., Norman, G., Parker, D.: Stochastic games for verification of probabilistic timed automata. In: Ouaknine, J., Vaandrager, F.W. (eds.) FORMATS 2009. LNCS, vol. 5813, pp. 212–227. Springer, Heidelberg (2009)
6. Hartmanns, A., Hermanns, H.: A Modest approach to checking probabilistic timed automata. In: Proc. QEST'09, pp. 187–196 (2009)
7. Berendsen, J., Jansen, D., Vaandrager, F.: Fortuna: Model checking priced probabilistic timed automata. In: Proc. QEST'10, IEEE Press, Los Alamitos (2010)
8. Kattenbelt, M., Kwiatkowska, M., Norman, G., Parker, D.: Abstraction refinement for probabilistic software. In: Jones, N.D., Müller-Olm, M. (eds.) VMCAI 2009. LNCS, vol. 5403, pp. 182–197. Springer, Heidelberg (2009)
9. Kwiatkowska, M., Norman, G., Parker, D.: Game-based abstraction for Markov decision processes. In: Proc. QEST'06, pp. 157–166. IEEE Press, Los Alamitos (2006)
10. Kattenbelt, M., Kwiatkowska, M., Norman, G., Parker, D.: A game-based abstraction-refinement framework for Markov decision processes. Technical Report RR-08-06, Oxford University Computing Laboratory (2008)
11. Vardi, M.: Formal techniques for SystemC verification. In: Proc. DAC'07, pp. 188–192 (2007); Position Paper
12. Kwiatkowska, M., Norman, G., Parker, D.: A framework for verification of software with time and probabilities (extended version), http://qav.comlab.ox.ac.uk/papers/formats10extended.pdf
13. Kemeny, J., Snell, J., Knapp, A.: Denumerable Markov Chains. Springer, Heidelberg (1976)
14. Shapley, L.: Stochastic games. Proc. National Academy of Science 39, 1095–1100 (1953)
15. Condon, A.: The complexity of stochastic games. Inf. and Comp. 96, 203–224 (1992)
16. Puterman, M.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. John Wiley and Sons, Chichester (1994)
17. Wachter, B., Zhang, L.: Best probabilistic transformers. In: Barthe, G., Hermenegildo, M. (eds.) VMCAI 2010. LNCS, vol. 5944, pp. 362–379. Springer, Heidelberg (2010)
18. Cousot, P., Cousot, R.: Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Proc. POPL'77, pp. 238–252. ACM Press, New York (1977)
19. Henzinger, T., Nicollin, X., Sifakis, J., Yovine, S.: Symbolic model checking for real-time systems. Information and Computation 111, 193–244 (1994)
20. Tripakis, S.: The formal analysis of timed systems in practice. PhD thesis, Université Joseph Fourier (1998)
21. Jensen, H.: Model checking probabilistic real time systems. In: Proc. 7th Nordic Workshop on Programming Theory, pp. 247–261 (1996)
22. Kwiatkowska, M., Norman, G., Segala, R., Sproston, J.: Automatic verification of real-time systems with discrete probability distributions. Theoretical Computer Science 282, 101–150 (2002)
23. Beauquier, D.: Probabilistic timed automata. Theoretical Computer Science 292, 65–84 (2003)
24. Kwiatkowska, M., Norman, G., Parker, D., Sproston, J.: Performance analysis of probabilistic timed automata using digital clocks. Formal Methods in System Design 29, 33–78 (2006)

25. Graf, S., Saidi, H.: Construction of abstract state graphs with PVS. In: Grumberg, O. (ed.) CAV 1997. LNCS, vol. 1254, pp. 72–83. Springer, Heidelberg (1997)
26. Alur, R., Courcoubetis, C., Dill, D.: Model-checking in dense real-time. Information and Computation 104, 2–34 (1993)
27. Lahiri, S., Ball, T., Cook, B.: Predicate abstraction via symbolic decision procedures. In: Etessami, K., Rajamani, S.K. (eds.) CAV 2005. LNCS, vol. 3576, pp. 24–38. Springer, Heidelberg (2005)
28. Clarke, E., Kroening, D., Sharygina, N., Yorav, K.: Predicate abstraction of ANSI-C programs using SAT. Formal Methods in System Design 25, 105–127 (2004)
29. Drechsler, R., Grosse, D.: Reachability analysis for formal verification of SystemC. In: Proc. DSD'02, p. 337. IEEE Press, Los Alamitos (2002)
30. Kroening, D., Sharygina, N.: Formal verification of SystemC by automatic hardware/software partitioning. In: Proc. MEMOCODE'05, pp. 101–110. IEEE Press, Los Alamitos (2005)
31. Grosse, D., Kühne, U., Drechsler, R.: HW/SW co-verification of embedded systems using bounded model checking. In: Proc. 16th ACM Great Lakes Symp. VLSI, pp. 43–48. ACM Press, New York (2006)
32. Karlsson, D., Eles, P., Peng, Z.: Formal verification of SystemC designs using a Petri-net based representation. In: Proc. DATE'06, EDAA, pp. 1228–1233 (2006)
33. Traulsen, C., Cornet, J., Moy, M., Maraninchi, F.: A SystemC/TLM semantics in Promela and its possible applications. In: Bošnački, D., Edelkamp, S. (eds.) SPIN 2007. LNCS, vol. 4595, pp. 204–222. Springer, Heidelberg (2007)
34. Herber, P., Fellmuth, J., Glesner, S.: Model checking SystemC designs using timed automata. In: Proc. CODES+ISSS'08, pp. 131–136 (2008)
35. Blanc, N., Kroening, D., Sharygina, N.: Scoot: A tool for the analysis of SystemC models. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 467–470. Springer, Heidelberg (2008)
36. Moy, M., Maraninchi, F., Maillet-Contoz, L.: Pinapa: An extraction tool for SystemC descriptions of Systems-on-a-Chip. In: Proc. EMSOFT'05, pp. 317–324 (2005)
37. IEEE Standards Association: IEEE standard 1666: SystemC language reference manual (2005), `http://www.systemc.org/`
38. Daws, C., Kwiatkowska, M., Norman, G.: Automatic verification of the IEEE 1394 root contention protocol with KRONOS and PRISM. Int. Journal on Software Tools for Technology Transfer 5, 221–236 (2004)
39. Kwiatkowska, M., Norman, G., Sproston, J., Wang, F.: Symbolic model checking for probabilistic timed automata. Inf. and Comp. 205, 1027–1077 (2007)
40. Berendsen, J., Jansen, D., Katoen, J.P.: Probably on time and within budget on reachability in priced probabilistic timed automata. In: Proc. QEST'06, pp. 311–322. IEEE Press, Los Alamitos (2006)
41. D'Argenio, P., Jeannet, B., Jensen, H., Larsen, K.: Reduction and refinement strategies for probabilistic analysis. In: Hermanns, H., Segala, R. (eds.) PROBMIV 2002, PAPM-PROBMIV 2002, and PAPM 2002. LNCS, vol. 2399, pp. 57–76. Springer, Heidelberg (2002)
42. Hermanns, H., Wachter, B., Zhang, L.: Probabilistic CEGAR. In: Gupta, A., Malik, S. (eds.) CAV 2008. LNCS, vol. 5123, pp. 162–175. Springer, Heidelberg (2008)

43. de Alfaro, L., Roy, P.: Magnifying-lens abstraction for Markov decision processes. In: Damm, W., Hermanns, H. (eds.) CAV 2007. LNCS, vol. 4590, pp. 325–338. Springer, Heidelberg (2007)
44. Chadha, R., Viswanathan, M.: A counterexample guided abstraction-refinement framework for Markov decision processes. ACM Transactions on Computational Logic (to appear, 2010)
45. Kemper, S., Platzer, A.: SAT-based abstraction refinement for real-time systems. In: STACS 1985. ENTCS, vol. 182, pp. 107–122 (2006)
46. Sorea, M.: Lazy approximation for dense real-time systems. In: Lakhnech, Y., Yovine, S. (eds.) FORMATS 2004 and FTRTFT 2004. LNCS, vol. 3253, pp. 363–378. Springer, Heidelberg (2004)
47. Dierks, H., Kupferschmid, S., Larsen, K.: Automatic abstraction refinement for timed automata. In: Raskin, J.-F., Thiagarajan, P.S. (eds.) FORMATS 2007. LNCS, vol. 4763, pp. 114–129. Springer, Heidelberg (2007)
48. Zbrzezny, A., Półrola: SAT-based reachability checking for timed automata with discrete data. Fundamenta Informaticae 79, 579–593 (2008)

Synchrony and Time in Fault-Tolerant Distributed Algorithms
(Invited Tutorial)

Ulrich Schmid

Technische Universität Wien
s@ecs.tuwien.ac.at

Abstract. In 1985, Fischer, Lynch and Paterson published their celebrated impossibility of solving distributed agreement (consensus) in purely asynchronous distributed systems with crash failures. Synchrony requirements, i.e., constraints on the occurrence of certain events in a distributed system, are hence mandatory for being able to solve interesting distributed computing problems. Timing requirements are the most obvious, though not the only, possibility to express synchrony conditions. We will survey existing partially synchronous distributed computing models and their ability to circumvent impossibility results, and explore the role of time and clocks in designing fault-tolerant distributed algorithms in such models.

K. Chatterjee and T.A. Henzinger (Eds.): FORMATS 2010, LNCS 6246, p. 46, 2010.

Reconciling Urgency and Variable Abstraction in a Hybrid Compositional Setting

D.A. van Beek, P.J.L. Cuijpers, J. Markovski,
D.E. Nadales Agut, and J.E. Rooda*

Eindhoven University of Technology (TU/e)
P.O. Box 513, NL-5600 MB Eindhoven, The Netherlands

Abstract. The extension of timed formalisms to a hybrid setting with urgency, has been carried out in a rather straightforward manner, seemingly without difficulty. However, in this paper, we show that the combination of urgency with abstraction from continuous variables leads to undesired behavior. Abstraction from continuous variables ultimately leads to a timed system again, but with a much richer set of possible branching behaviors than a timed system that comprises only clocks. As it turns out, the formal definition of urgency, as defined for timed systems with clocks, does not fit our intuition of urgency anymore when applied to a timed system that is an abstraction of a hybrid system. Therefore, we propose to extend the formal semantics of timed and hybrid systems with guard trajectories. In this way, the continuous branching behavior introduced by hybrid systems remains visible even after abstraction from continuous variables. The practical applicability of the introduction of guard trajectories is illustrated by our revision of the structured operational semantics of the CIF language. The interplay between urgency and abstraction now adheres to our intuition, while compositionality with respect to urgency, variable abstraction, and parallel composition, is retained. In the future, symbolic elimination of urgency can be used to ensure that guard trajectories do not need to be actually calculated.

1 Introduction

Urgent actions were introduced in timed formalisms to support easy modeling of greedy, or eager, behavior. As an example, we consider timed automata of [1]. These timed automata are extensions of standard automata with clocks that keep track of passage of time. The actions of an automaton are guarded by constraints on the clocks, which indicate when the action *may* be performed. In addition, the introduction of urgency of actions to this model allows the modeler to determine when an action *must* be performed. Using model checkers like UPPAAL, KRONOS, and IF [2–4], one can then check whether the urgent execution of actions will guarantee that these actions meet their deadlines.

* This work was partly done as part of the EU FP7 2007-2013 projects MULTIFORM and C4C, contract numbers FP7-ICT-224249 and FP7-ICT-223844.

K. Chatterjee and T.A. Henzinger (Eds.): FORMATS 2010, LNCS 6246, pp. 47–61, 2010.

The extension of these timed formalisms to a hybrid setting with urgency has been carried out in a rather straightforward manner. In hybrid automata [5, 6] clocks are generalized to differential equations over continuous variables. Thus, guards deal with continuous variables, so urgency allows the modeler to specify that an action will happen *as soon as* a guard becomes true. Tools like HyTech and HyVisual [7, 8] already support this idea, e.g., they can be used to verify that an action is executed before a certain continuous condition is met.

Admittedly, the formal definition of urgency has had its complications. One problem was that the earliest possible moment of an action may not be defined, e.g., when the guard on an action is a left-open time-interval. Another problem was that the unexpected blocking of urgent actions due to a failing synchronization may enable other actions that were not available before. In this way, systems that were considered equivalent before placing them in a parallel composition, may behave differently after placing them in a parallel composition, thus ruining compositionality. Still, most of these complications have been solved for timed systems. Thus, one often defines that the urgent execution of an action in a left-open time-interval leads to a deadlock, and one can strengthen the notion of equivalence to consider non-urgent actions, even if there are earlier urgent actions that prevent them. These solutions, even though not the only possible ones, are satisfactory and readily transferrable to a hybrid setting as well.

An additional complication in the definition of urgency arises only when one starts in a hybrid setting and subsequently obtains a timed system through abstraction of hybrid variables. In fact, the problem already manifests itself when only part of the continuous behavior is abstracted from, but for the sake of simplicity we abstract from all variables in the setting of this paper. To the best of our knowledge, the combination of urgency and variable abstraction has not been studied in detail before, hence the problem did not manifest itself sooner.

When abstracting from the value of continuous variables in a hybrid system, one would like to obtain a timed system in which the moments at which certain urgent actions are enabled or disabled are accurately preserved. The fact that the value of a continuous variable is not directly observable, should not change the fact that a certain guard that depends on this value is true or false at a certain time. Furthermore, we would expect in particular that if a system contains a deadlock *before* abstracting from the value of continuous variables, that it also contains this deadlock *after* abstracting from the value of these variables. After all, the abstraction is intended to give us a system that is easier to analyse, which means that the abstraction should preserve the properties that we are interested in. Finally, we expect that abstraction distributes over compositions and operators whenever reasonably possible, because we would like to employ abstraction to verify separate components. In the opposite case, the results of such a partial verification could not be used in the verification of the whole.

The definitions for abstraction from variables in current literature, e.g. [9, 10], aim towards obtaining a timed transition system when all variables are abstracted from. However, in a timed transition system the continuity of the moments of choice is abstracted from, which becomes even more prominent when

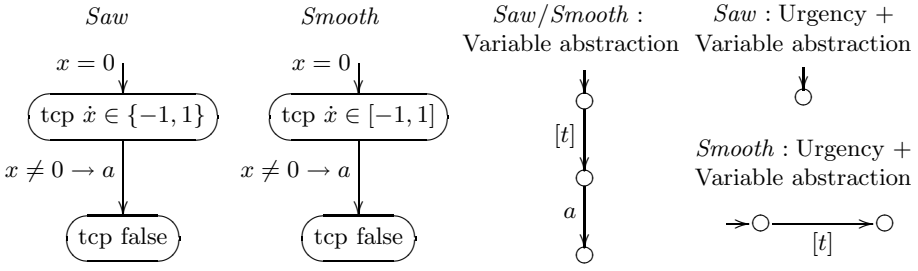


Fig. 1. *μCIF* automata *Saw* and *Smooth*, and the resulting infinite timed transitions systems parameterized with $t \in \mathbb{R}$ such that $t > 0$ after applying: (1) variable abstraction and (2) urgency followed by variable abstraction

urgency is introduced. We note that other typical compositions and operators in the automata- and process-theoretic approaches are not affected. As an illustration of how urgency depends on the continuity of the moments of choice, let us consider the hybrid systems depicted in Fig. 1. The systems differ only in the differential inclusions that define their continuous behavior. *Saw* has a differential inclusion ranging over two points, while *Smooth* has a differential inclusion ranging over the closed interval between these two points.

Using the existing definitions of urgency, we can show that any behavior of *Saw* can be mimicked by *Smooth*, but not vice versa. In particular, from the initial valuation $x \mapsto 0$, *Smooth* allows x to remain 0 for an arbitrary period of time (thus disallowing the action a), while *Saw* does not allow x to remain 0 and it is always able to execute an a after an arbitrarily small delay. Declaring the action a to be urgent, now shows the difference between *Saw* and *Smooth* even more prominently, because the action a must happen as soon as it can. The application of urgency to *Saw* leads to a deadlock, since the guard leads to a left-open time-interval in which a is enabled. The latter gives us a system in which $x(t) = 0$ is the only possible solution, so a will never occur, but time can progress.

The above described behavior supports the intuition, but when we abstract from the value of x before applying urgency, the semantics changes unexpectedly. The abstraction from x results in a isomorphic timed transition system for both *Saw* and *Smooth*, as each variable trajectory in *Smooth* has a related trajectory in *Saw* with the same duration and the same start- and end-valuation. Making a urgent leads to a system in which no transition labeled by a can occur, but time can progress. Apparently, by abstracting from x, we also abstracted from the intervals in which a is enabled, causing the deadlock to disappear. This is not desirable, since it allows for variable abstraction only on the 'top' level, after the system has been described completely and urgency has been applied. Therefore, we cannot employ variable abstraction to simplify the verification of the components of a system.

As a remedy, we propose to make the guard trajectories visible on the timed transitions, even if one has already abstracted from the values of the variables

that these guards comprise. To illustrate our approach, we adapt the structured operation semantics (SOS) [11] of a subset of the compositional interchange format (*CIF*) language [12, 13] that comprises urgency and variable abstraction. Another solution to the above problem would be to restrict to hybrid systems in which the above phenomenon does not occur. These systems, referred to as *finite set refutable*, were studied in [14]. All timed systems that comprise fixed clock-rates are finite-set refutable, as well as all hybrid systems that employ only differential equations, rather than differential inclusions. However, open systems with free input/output variables and systems with open differential inclusions are, in general, not finite-set refutable. So, while the extension to guard trajectories does not change anything for classical timed system, it provides a more applicable theory with a more robust notion of variable abstraction. Furthermore, ongoing research indicates that symbolic elimination of the urgency operator is possible, meaning that the introduced guard trajectories never need to be actually calculated when analyzing the behavior of a system.

The remainder of this paper is structured as follows. We discuss related work in section 2. Section 3 illustrates our approach by adapting the SOS of a relevant subset of the *CIF* language. We show that urgency and variable abstraction indeed distribute modulo bisimulation, solving the aforementioned problems. In section 4 we show compatibility with the common urgency concepts from the literature, and we end with concluding remarks in section 5.

2 Related Work

In this section we discuss the notions of urgency in the literature on timed and hybrid automata and relate them to our work.

Specifying urgency. Early approaches to specifying urgency employed state invariants to limit the progress of time at a given location [15]. Amongst others, this approach found its way into the toolset UPPAAL [2]. A variant of this approach, termed stopping conditions, has been employed in a timed restriction of the hybrid I/O automata [16]. When the stopping condition becomes valid, the progress of time is stopped and an activity must execute immediately. Another extension, termed timed automata with deadlines, was investigated in [17–20]. Deadlines are auxiliary clock constraints, which specify when the transition has become urgent. It is required that they imply the corresponding guard constraints to ensure that at least one transition has been enabled after stopping the progress of time. This property of progress of time is also known as time reactivity or time-lock freedom [19, 21]. Timed automata with deadlines are embedded in the specification languages IF and MODEST [4, 22]. Yet another approach employing urgency predicates has been proposed in [23] as an extension of timed I/O automata. In the same study the four approaches from above have been paralleled, concluding that stopping conditions, deadlines, and urgency predicates essentially have the same expressivity, whereas invariants can be additionally applied to right-open intervals as well. However, by construction only deadlines and urgency predicates guarantee time reactivity.

In the hybrid setting, urgency flags denote urgent actions in the framework of HyTech [7] and invariants are required to endorse urgent transitions. The same approach is used when coupling Hytech and UPPAAL [24]. The completely opposite approach is taken in HyVisual [8], where every action considered urgent.

In this paper, we consider two types of urgency: (1) global labeled transition urgency by means of urgent action labels and (2) local urgency by means of the time-can-progress construct. The former is implemented by means of an urgency mapping in a fashion similar to the urgency flags of [7], whereas as the latter is closest to the stopping conditions of [16]. As a design choice, we do not hard-code/enforce the property of time reactivity in the semantics, although we foresee that it can be supported by restricting the allowed syntax.

Synchronization of urgent actions. When considering synchronization of urgent actions, the literature provides three prominent manners: (1) synchronization with hard deadlines or impatient synchronization or AND-synchronization, where the urgency constraints of all synchronizing parties must be endorsed as soon as they are enabled, (2) synchronization with soft deadlines or patient synchronization or MAX-synchronization, where some urgency constraints can be contravened so that the synchronization can occur as long as the guards still hold, and (3) MIN-synchronization, which occurs when one of the synchronizing parties is ready to synchronize provided that the other will eventually be enabled. In the setting of this paper, we opt for both patient and impatient synchronization. Implementing MIN-synchronization would require substantial changes of the semantics, i.e., time look-ahead capability, whilst its practical use for modeling real-life systems is limited [25].

When using urgency flags, there are substantial restrictions on the guards of the synchronizing actions [7, 24]. In the current setting, we use an operator to specify synchronizing actions and urgency. The patient synchronization contravenes urgency constraints by assuming non-urgency of all synchronizing actions and, only after successful synchronization in accordance with the guards has succeeded, it re-imposes the urgency constraints. Our approach is similar to the drop bisimulation proposed in [20], where deadlines are dropped to enable patient synchronization and, afterwards, undropped to preserve compositionality. To support modeling with data we also introduced urgent channels in the extended framework of χ 2.0 [26], a topic beyond the scope of this paper.

3 *μCIF*

This section presents a concise syntax and formal semantics of a subset of the *CIF* language [12], denoted as *μCIF*. It illustrates how the urgency concept can be defined in a compositional way, robust against abstraction from local variables, by extending hybrid transition systems (HTSs) [27] with guard trajectories. Similar ideas are applied in the complete framework of χ 2.0 [26], and in the latest extension of CIF [12, 28]. *μCIF* is a modeling language for hybrid systems that adopts concepts from hybrid automata theory and process algebra. We briefly overview the features of the language.

The basic building blocks of *μCIF* are atomic automata, which consist of a set of locations $\mathcal{L}$ and edges that connect them. A location specifies a state of the system, and it can have equations associated to it in the form of *time can progress* predicates. These equations define the continuous behavior of the automaton: they determine when time can pass, and how the value of the variables changes as time elapses. The values of the variables belong to the set Λ that contains, among else, the sets $\mathbb{B}$, $\mathbb{R}$, and $\mathbb{C}$. Guarded labeled transitions specify the discrete behavior of the system. These transitions are labeled with actions originating from $\mathcal{A}$. Guards are incorporated as predicates over variables, given by $\mathcal{P}$. Continuous behavior is specified by timed transitions labeled by variable and guard trajectories as described below. The set $\mathcal{T}$ comprises all time points.

We distinguish between two types of variables: (1) variables, denoted by the set $\mathcal{V}$, and (2) the dotted versions of those variables, which belong to the set $\dot{\mathcal{V}} \triangleq \{\dot{x} \mid x \in \mathcal{V}\}$. The evolution of the value of a variable as time elapses is constrained by equations. Furthermore, we distinguish between *discrete variables*, which dotted versions are always 0, and *continuous variables*, which dotted versions represent the derivative. Variables are constrained by differential algebraic equations and we implement them as predicates, given by $\mathcal{P}$, over all variables $\mathcal{V} \cup \dot{\mathcal{V}}$. We also have initialization conditions, elements of $\mathcal{P}$, which allows to model steady state initialization.

We introduce several operators of *μCIF*. Parallel composition with synchronization composes two *μCIF* automata in parallel, synchronizing on a specified set of actions, while interleaving the rest. Actions are not synchronizing by default and they must be declared as such by placing them in the synchronization set. We introduce local urgency by means of a *time-can-progress* predicate and global urgency by means of *urgency composition*. The time-can-progress predicate is associated to each location of the automaton and specifies whether passage of time is allowed. Action transitions become urgent when time can no longer progress. The urgency operator declares action transitions as urgent, thus disabling the passage of time whenever an urgent action is enabled. Finally, variables in *μCIF* can be made local by means of *variable scopes*. When a variable is abstracted from, the changes made to the variable by the automaton are not visible outside the scope, and vice versa, external automata cannot change the value of the abstracted variable.

The semantics of the *μCIF* is given in the form of a structured operational semantics (SOS) [11] on a hybrid transition system(HTS) [27]. This is significantly different with the typical way of giving semantics to hybrid automata [5]. In the SOS approach, the semantics of the operators is defined by rules that give the semantics of the composition on the basis of the semantics of the constituent components, whereas in traditional approaches, the semantics of the composition is given by syntactic transformations of the constituent components to a basic automaton. In [12, 28], the SOS approach was chosen since it is better suited for guiding the implementation process [29]. Additionally, it makes use of standard SOS formats to show compositionality [30], thus enabling symbolic reasoning. Since *μCIF* is a subset of CIF, we inherit the SOS approach.

Syntax. The basic building block of *μCIF* are atomic automata. An atomic automaton resembles a hybrid automaton: for each location in an automaton there is a predicate 'init' that specifies the conditions under which execution can begin on that location, and a predicate 'tcp' that specifies local urgency by giving the conditions under which time can progress in that location and that determines how the value of variables change over time, defining the dynamic behavior. The edges of the automaton describe the actions that the automaton can perform.

Definition 1. *An atomic automaton is a tuple* $(L, \text{init}, \text{tcp}, E)$ *with a set of locations* $L \subseteq \mathcal{L}$*; initial and time-can-progress predicates* $\text{init}, \text{tcp}\colon L \to \mathcal{P}$*; and a set of edges* $E \subseteq L \times \mathcal{P} \times \mathcal{A} \times L$.

Starting with atomic automata, *μCIF* automata are built using a set of compositions $C \in \mathcal{C}$, given by $C ::= \alpha \mid C \parallel_S C \mid \upsilon_U(C) \mid [\![x, \dot{x} \mapsto c, d :: C]\!]$, where α is an atomic automaton, $S \subseteq \mathcal{A}$, $U \subseteq \mathcal{A}$, $x \in \mathcal{V}$, $\dot{x} \in \dot{\mathcal{V}}$, and $c, d \in \Lambda$.

The semantics of *μCIF* automata is in terms of a hybrid transition system (HTS). Each state of this HTS is a pair $\langle p, \sigma \rangle$ comprising a composition of automata p and a *valuation* σ that associates a value to each variable. The set of all valuations is denoted by $\Sigma \triangleq \mathcal{V} \cup \dot{\mathcal{V}} \to \Lambda$. We keep track of the evolution of the values of variables using the concept of *variable trajectories*, denoted as $\rho : \mathcal{T} \to \Sigma$. A variable trajectory ρ holds the values of the variables at each time point $s \in \mathcal{T}$ in the delay. Since the values of variables change over time, the truth values of the guards dependent on the variable change as well. Thus, the set of enabled action transitions at each point in time can be determined using the concept of *guard trajectories*. A guard trajectory, denoted as $\theta : \mathcal{T} \to 2^{\mathcal{A}}$, keeps track of the set of enabled actions for each point in time.

The discrete and continuous behavior of HTSs is defined in terms of action transitions, given by $\_ \xrightarrow{\cdot} \_ \subseteq (\mathcal{C} \times \Sigma) \times \mathcal{A} \times (\mathcal{C} \times \Sigma)$, and timed transitions, given by $\_ \xmapsto{\cdot} \_ \subseteq (\mathcal{C} \times \Sigma) \times ((\mathcal{T} \to \Sigma) \times (\mathcal{T} \to 2^{\mathcal{A}})) \times (\mathcal{C} \times \Sigma)$, respectively. The intuition of an action transition $\langle p, \sigma \rangle \xrightarrow{a} \langle p', \sigma' \rangle$ is that $\langle p, \sigma \rangle$ executes the discrete action $a \in \mathcal{A}$ and thereby transforms into $\langle p', \sigma' \rangle$, where p' and σ' denote the resulting automaton and variable valuation, respectively. The intuition behind a timed transition $\langle p, \sigma \rangle \xmapsto{\rho,\theta} \langle p', \sigma' \rangle$ is that during the passage of time, the valuation that defines the values of the visible variables at each time-point $s \in [0, t] = \text{dom}(\rho) = \text{dom}(\theta)$ is given by the variable trajectory $\rho(s)$. The novelty in this paper is the set of enabled actions at each time-point $s \in [0, t]$, given by the guard trajectory $\theta(s)$. At the end-time point $t \in \mathcal{T}$, the resulting state is $\langle p', \sigma' \rangle$.

Structural Operational Semantics. We use $\sigma(x)$, with $\sigma \in \Sigma$, to denote the value of variable x in valuation σ. By $\sigma \models u$ we denote that the predicate $u \in \mathcal{P}$ is satisfied in the valuation σ.

An atomic automaton $(L, \text{init}, \text{tcp}, E)$ can execute actions or time delays. Given an active location $\ell \in L$, i.e., a location for which $\text{init}(\ell)$ holds, an action a can be executed only if there is an edge (ℓ, g, a, ℓ') such that the guard g is satisfied. Rule 1 formalizes as follows:

$$\frac{(\ell, g, a, \ell') \in E,\ \sigma \models \text{init}(\ell),\ \sigma \models g}{\langle (L, \text{init}, \text{tcp}, E), \sigma \rangle \xrightarrow{a} \langle (L, \text{id}_{\ell'}, \text{tcp}, E), \sigma \rangle}\ 1$$

where $\text{id}_\ell \in L \to \mathcal{P}$ for $\ell \in \mathcal{L}$ such that $\text{id}_\ell(\ell'') \triangleq\ \ell = \ell''$.

During a time delay $[0, t]$ for $t \in \mathcal{T}$, the values of variables can change. In a timed transition $\langle p, \sigma \rangle \xmapsto{\rho,\theta} \langle p', \sigma' \rangle$, the variable trajectory ρ contains the values of variables at each point in the time interval $[0, t]$. The value of x can be defined as a function $\rho \downarrow x : [0, t] \to \Lambda$, in terms of ρ, such that $(\rho \downarrow x)(s) = \rho(s)(x)$.

In the CIF tooling, the variables x and $\dot{x}$ are, in principle, different variables. This makes it easier to implement an algorithm that checks whether a certain variable trajectory is a solution of an equation. The coupling between x and $\dot{x}$ is performed through the definition of a *dynamic type*. Most importantly, this dynamic type ensures that discrete variables remain constant over time, and that continuous variables have the expected derivatives. Formally, the dynamic type of a variable is a set $G \subseteq 2^{(\mathcal{T} \to \Lambda \times \mathcal{T} \to \Lambda)}$. A variable trajectory ρ is said to satisfy a dynamic type constraint G if $(\rho \downarrow x, \rho \downarrow \dot{x}) \in G$. For each variable x we assume the existence of a dynamic type G_x associated to it.

Now, timed transitions are enabled in an active location ℓ if there exists a *valid variable trajectory*. A variable trajectory ρ is valid if it has a positive duration, i.e. $\text{dom}(\rho) = [0, t]$ and $0 < t$; tcp holds during $[0, t)$, with $[0, 0) \triangleq \emptyset$, and all variables satisfy the *dynamic type constraints*. The guard trajectory is constructed accordingly based on the variable trajectory, as given by rule 2:

$$\frac{\rho(0) \models \text{init}(\ell), \forall_{s \in [0,t)}\, \rho(s) \models \text{tcp}(\ell), \forall_{x \in \mathcal{V}}\, (\rho \downarrow x, \rho \downarrow \dot{x}) \in G_x}{\langle (L, \text{init}, \text{tcp}, E), \rho(0) \rangle \xmapsto{\rho,\theta} \langle (L, \text{id}_\ell, \text{tcp}, E), \rho(t) \rangle}\ 2$$

where the duration of the delay is positive, $0 < t$, and the trajectories are defined exactly on the delay interval $\text{dom}(\rho) = [0, t]$, $\text{dom}(\theta) = [0, t]$. The guard trajectory is defined as $\forall_{s \in [0,t]}\, \theta(s) = \{a \mid (\ell, g, a, \ell') \in E \wedge \rho(s) \models g\}$.

Parallel composition. As a result of composing two *μCIF* automata in parallel, the equally labeled action transitions of both components that are specified in the synchronization set $S \subseteq \mathcal{A}$ must be taken together. This is given by rule 3. The other action transitions are interleaved, as stated by rules 4 and 5.

$$\frac{\langle p, \sigma \rangle \xrightarrow{a} \langle p', \sigma' \rangle, \langle q, \sigma \rangle \xrightarrow{a} \langle q', \sigma' \rangle, a \in S}{\langle p \parallel_S q, \sigma \rangle \xrightarrow{a} \langle p' \parallel_S q', \sigma' \rangle}\ 3$$

$$\frac{\langle p, \sigma \rangle \xrightarrow{a} \langle p', \sigma' \rangle, a \notin S}{\langle p \parallel_S q, \sigma \rangle \xrightarrow{a} \langle p' \parallel_S q, \sigma' \rangle}\ 4 \qquad \frac{\langle q, \sigma \rangle \xrightarrow{a} \langle q', \sigma' \rangle, a \notin S}{\langle p \parallel_S q, \sigma \rangle \xrightarrow{a} \langle p \parallel_S q', \sigma' \rangle}\ 5$$

For timed transitions, the two components must agree in their variable trajectories (and hence in the duration of the time delay). The guard trajectory is constructed from the guard trajectories of the components of the parallel composition: at a given time point s, an action is enabled in the parallel composition $p \parallel_S q$ if is enabled in p and q (regardless of whether the action is in S), or if it is enabled in p or q and it is not synchronizing. Rule 6 formalizes this:

$$\frac{\langle p,\sigma\rangle \overset{\rho,\theta_p}{\longmapsto} \langle p',\sigma'\rangle, \langle q,\sigma\rangle \overset{\rho,\theta_q}{\longmapsto} \langle q',\sigma'\rangle}{\langle p \parallel_S q,\sigma\rangle \overset{\rho,\theta_p \oplus_S \theta_q}{\longmapsto} \langle p' \parallel_S q',\sigma'\rangle}\ 6$$

where $(\theta_p \oplus_S \theta_q)(t) \triangleq (\theta_p(t) \cap \theta_q(t)) \cup (\theta_p(t) \setminus S) \cup (\theta_q(t) \setminus S)$.

Urgency operator. The urgency operator $\upsilon_U(p)$ gives actions from the set $U \subseteq \mathcal{A}$ a higher priority than timed transitions. Timed transitions are allowed only if at the current state, and at each intermediate state while delaying, there is no urgent action enabled. This is given by rules 7 and 8 as follows:

$$\frac{\langle p,\sigma\rangle \overset{a}{\longrightarrow} \langle p',\sigma'\rangle}{\langle \upsilon_U(p),\sigma\rangle \overset{a}{\longrightarrow} \langle \upsilon_U(p'),\sigma'\rangle}\ 7 \qquad \frac{\langle p,\sigma\rangle \overset{\rho,\theta}{\longmapsto} \langle p',\sigma'\rangle,\ \forall_{s\in[0,t)}\, U \cap \theta(s) = \emptyset}{\langle \upsilon_U(p),\sigma\rangle \overset{\rho,\theta}{\longmapsto} \langle \upsilon_U(p'),\sigma'\rangle}\ 8$$

Variable Scope. We introduce local variables by means of the variable scope operator. The variable scope $[\![\, x, \dot{x} \mapsto c, d :: p \,]\!]$, for $x \in \mathcal{V}$, $\dot{x} \in \dot{\mathcal{V}}$, $c, d \in \Lambda$, and $p \in \mathcal{C}$, behaves as p but all the changes made to the local variables x and $\dot{x}$ are invisible. The initial values of x and $\dot{x}$ are c and d, respectively.

To define the semantics of this operator we use the function overwriting operator $\succ$, which, given two functions f and g, is defined as $f \succ g = f \cup g \restriction_{\mathrm{dom}(f)}$, where $g \restriction_X$ is the restriction of the domain of function g to $\mathrm{dom}(g) \setminus X$. In rules 9 and 10 the local value of the variables at the end of the transition is kept in the scope operator, whereas changes in the global valuation are overridden.

$$\frac{\langle p,\sigma_{xcd}\rangle \overset{a}{\longrightarrow} \langle p',\sigma'\rangle}{\langle [\![\, x, \dot{x} \mapsto c, d :: p \,]\!],\sigma\rangle \overset{a}{\longrightarrow} \langle [\![\, x, \dot{x} \mapsto \sigma'(x), \sigma'(\dot{x}) :: p' \,]\!],\sigma'_{xef}\rangle}\ 9$$

$$\frac{\langle p,\sigma_{xcd}\rangle \overset{\rho,\theta}{\longmapsto} \langle p',\sigma'\rangle}{\langle [\![\, x, \dot{x} \mapsto c, d :: p \,]\!],\rho_x(0)\rangle \overset{\rho_x,\theta}{\longmapsto} \langle [\![\, x, \dot{x} \mapsto \sigma'(x), \sigma'(\dot{x}) :: p' \,]\!],\rho_x(t)\rangle}\ 10$$

where $\forall_{\sigma\in\Sigma,\, x\in\mathcal{V},\, c,d\in\Lambda}\ \sigma_{xcd} \triangleq \{x \mapsto c, \dot{x} \mapsto d\} \succ \sigma$ and $\mathrm{dom}(\rho_x) = \mathrm{dom}(\rho)$ with $\forall_{s\in\mathrm{dom}(\rho)}\, \exists_{c,d\in\Lambda}\ \rho_x(s) \triangleq \rho(s) \restriction_{\{x,\dot{x}\}} \cup \{x \mapsto c, \dot{x} \mapsto d\}$.

Stateless bisimilarity. Two *μCIF* components are equivalent if they have the same behavior (in the bisimulation sense) given the same valuation of variables.

Definition 2. *A symmetric relation $R \subseteq \mathcal{C} \times \mathcal{C}$ is a stateless bisimulation relation if and only if for all $(p,q) \in R$ it holds that (1) $\forall_{\sigma,\sigma'\in\Sigma,\, a\in\mathcal{A},\, p'\in\mathcal{C}}\ \langle p,\sigma\rangle \overset{a}{\longrightarrow} \langle p',\sigma'\rangle \Rightarrow \exists_{q'\in\mathcal{C}}\, \langle q,\sigma\rangle \overset{a}{\longrightarrow} \langle q',\sigma'\rangle \wedge (p',q') \in R$, and (2) $\forall_{\sigma,\sigma'\in\Sigma,\, \rho\in\mathcal{T}\to\Sigma,\, \theta\in\mathcal{T}\to 2^{\mathcal{A}},\, p'\in\mathcal{C}}\ \langle p,\sigma\rangle \overset{\rho,\theta}{\longmapsto} \langle p',\sigma'\rangle \Rightarrow \exists_{q'\in\mathcal{C}}\, \langle q,\sigma\rangle \overset{\rho,\theta}{\longmapsto} \langle q',\sigma'\rangle \wedge (p',q') \in R$. Two components p and q are stateless bisimilar, denoted by $p \underline{\leftrightarrow} q$, if there exists a stateless bisimulation relation R such that $(p,q) \in R$.*

Stateless bisimilarity is a congruence for all *μCIF* operators. This facilitates symbolic reasoning as we can replace equivalent automata in any context.

Theorem 1. *Stateless bisimilarity is a congruence over all μCIF operators.*

Proof. The SOS rules 1–10 satisfy the *process-tyft* format, which guarantees congruence for stateless bisimilarity [30]. □

Urgency and variable abstraction. The core of the problem in the interaction between urgency and variable abstraction is in the use of timed transition systems. In the previous subsections, we have given a semantics that does not abstract to a timed transition system, but rather to a 'guard trajectory labeled' transition system. Next, we give the main theorem of this paper, and prove that this change in semantics indeed solves the problem. The theorem states that urgency and variable abstraction distribute, meaning that the order in which they are applied is of no consequence anymore.

Theorem 2. $\forall_{p\in\mathcal{C},\, U\subseteq\mathcal{A}}\ [\![\ x, \dot{x} \mapsto c, d :: \upsilon_U(p)\ ]\!] \ \underline{\leftrightarrow}\ \upsilon_U([\![\ x, \dot{x} \mapsto c, d :: p\ ]\!])$.

Proof. Let $\mathcal{R} = \{([\![\ x,\ \dot{x}\ \mapsto\ c,\ d\ ::\ \upsilon_U(r)\ ]\!],\ \upsilon_U([\![\ x, \dot{x} \mapsto c, d :: r\ ]\!]))\ |\ r \in \mathcal{C},\ U \subseteq \mathcal{A},\ c, d \in \Lambda\}$. We will show that R is a stateless bisimulation, i.e., it satisfies the conditions of Definition 2.

Let $(p,\, q) \in R$ be such that $p = [\![\ x,\ \dot{x}\ \mapsto\ c,\ d\ ::\ \upsilon_U(r)\ ]\!]$ and $q = \upsilon_U([\![\ x, \dot{x} \mapsto c, d :: r\ ]\!]))$. Assume $\langle [\![\ x, \dot{x} \mapsto c, d :: \upsilon_U(r)\ ]\!], \sigma\rangle \xrightarrow{a} \langle p', \sigma'_{xef}\rangle$. By rules 9 and 7, we know that $p' = [\![\ x,\ \dot{x}\ \mapsto\ \sigma'(x),\ \sigma'(\dot{x})\ ::\ \upsilon_U(r')\ ]\!]$, $\langle \upsilon_U(r), \sigma_{xcd}\rangle \xrightarrow{a} \langle \upsilon_U(r'), \sigma'\rangle$, and $\langle r, \sigma_{xcd}\rangle \xrightarrow{a} \langle r', \sigma'\rangle$. Thus applying rules 9 and 7, we have that $\langle \upsilon_U([\![\ x, \dot{x} \mapsto c, d :: r\ ]\!])), \sigma\rangle \xrightarrow{a} \langle \upsilon_U([\![\ x, \dot{x} \mapsto \sigma'(x), \sigma'(\dot{x}) :: r'\ ]\!]), \sigma'_{xef}\rangle$.

Next suppose $\langle p, \rho_x(0)\rangle \overset{\rho_x,\theta}{\longmapsto} \langle p', \rho_x(t)\rangle$. By rule 10, we have that $[4]\langle \upsilon_U(r), \sigma_{xcd}\rangle \overset{\rho,\theta}{\longmapsto} \langle \upsilon_U(r'), \sigma'\rangle$, and by rule 8 we have that $[4]\langle r, \sigma_{xcd}\rangle \overset{\rho,\theta}{\longmapsto} \langle r', \sigma'\rangle$ and $\forall_{s\in[0,t)}\, U \cap \theta(s) = \emptyset$. Applying rule 10 to the last transition we get $\langle [\![\ x,\ \dot{x}\ \mapsto\ c,\ d\ ::\ r\ ]\!]\ , \rho_x(0)\rangle \overset{\rho_x,\theta}{\longmapsto} \langle [\![\ x, \dot{x} \mapsto c, d :: r'\ ]\!], \rho_x(t)\rangle$, and since we know that $\forall_{s\in[0,t)}\, U \cap \theta(s) = \emptyset$, we can apply rule 8 to obtain $\langle \upsilon_U([\![\ x,\ \dot{x}\ \mapsto\ c,\ d\ ::\ r\ ]\!]), \rho_x(0)\rangle \overset{\rho_x,\theta}{\longmapsto} \langle \upsilon_U([\![\ x, \dot{x} \mapsto c, d :: r'\ ]\!]), \rho_x(t)\rangle$. □

We note that an essential part of the proof of Theorem 2 is that the guard trajectories remain the same after variable abstraction, allowing us to deduce that urgent transitions do not interrupt the passage of time. Only when a system is finite set refutable[14], these guard trajectories can be recovered from the duration of the time transitions and the guards on the intermediate states.

4 Compositionality and Synchronization

We discuss the compositionality features of μCIF that are prominent in the literature, as well as passage of time when synchronizing urgent action transitions.

Compositionality of the parallel composition. When synchronizing action transitions in timed and hybrid automata, one typically takes the conjunction of the guards involved in the synchronization as a guard of the synchronized transition. As noted in [20, 23] this is the most prominent way of synchronization.

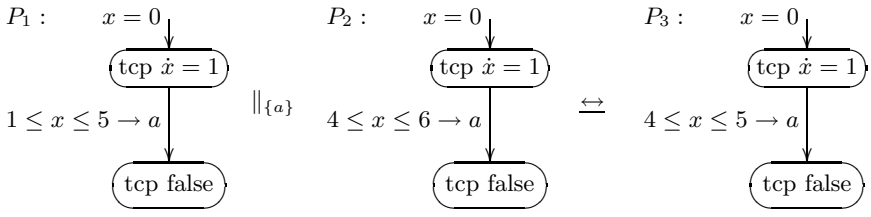


Fig. 2. Parallel composition with synchronization of μCIF automata

As an illustration, we depict the composition of two μCIF automata in Fig. 2, where $P_1 \parallel_{\{a\}} P_2 \leftrightarrow P_3$. Recall that initial states are depicted by incoming arrows on which we state the initial values of the variables. State invariants like tcp predicates are placed inside a state.

Taking the conjunction of the guards as a guard of the synchronized actions may lead to compositionality problems, depending on the way urgency is defined. As an example, consider the implementation of urgent transitions through deadlines, as in [20]. In Fig. 3 we have two timed automata with deadlines (TAD), T_1 and T_2. The initial value of the clock x is 0 and the guards $g : 1 \leq x \leq 5$ and $g : 4 \leq x \leq 6$ are associated with the actions a and c, respectively, denoting when the transitions may be taken. The deadlines $d : x \geq 3$ and $d : x \geq 6$ express when the transitions labeled by a and c must be taken, respectively. Thus, the action a is enabled in the interval $[1, 5]$ and must be taken at 3, whereas c is enabled in the interval $[4, 6]$ and must be taken at 6. The TAD T_1 and T_2 are considered bisimilar, since due to the urgency of the action a, the transition labeled by c will never be taken. However, when synchronizing on the action a, with a component that suppresses it, e.g., a component 'stop' that only idles, we see that the parallel composition behaves differently. Namely, the previously preempted action c is now observable in $T_2 \parallel_{\{a\}}$ stop, whereas $T_1 \parallel_{\{a\}}$ stop. Consequently, standard timed bisimulation [1] is not a congruence for TAD [20].

In μCIF, we solve the compositionality issue by defining the semantics of urgency and synchronization differently. We specify urgency using the tcp predicate, which is basically a state timed invariant, different from deadlines, which are on transition level. Additionally, we use SOS to define synchronization directly on the semantic level of transition systems, rather than defining it on the symbolic level of automata. In this way, we obtain compositionality for free by adhering to the process-tyft format from [30].

The closest mimic to the automata of Fig. 3 is given by the μCIF automata in Fig. 4, where the behavior of the automata T_1, T_2, and 'stop' is mimicked by P_4, P_5 and P_6, respectively. The crucial difference is that we use the tcp state predicate to stop the progress of time in order to induce urgency, which imposes a unique deadline for all outgoing transitions. This is observed in the initial state of P_5 as tcp $x \leq 3 \wedge x \leq 6$, which is equivalent to tcp $x \leq 3$. In our setting, P_4 is bisimilar to P_5, but the change in the semantics of urgency and the parallel composition ensures that $P_4 \parallel_{\{a\}} P_6$ is also bisimilar to $P_5 \parallel_{\{a\}} P_6$.

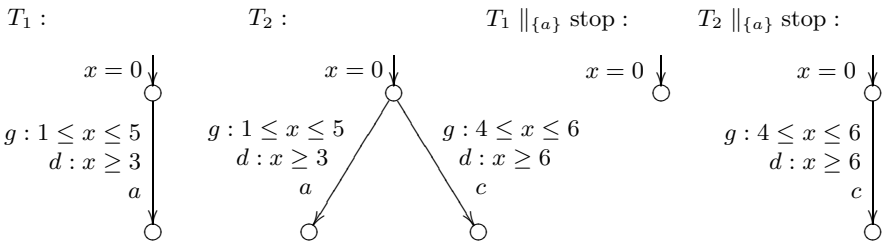


Fig. 3. Compositionality of timed automata with deadlines

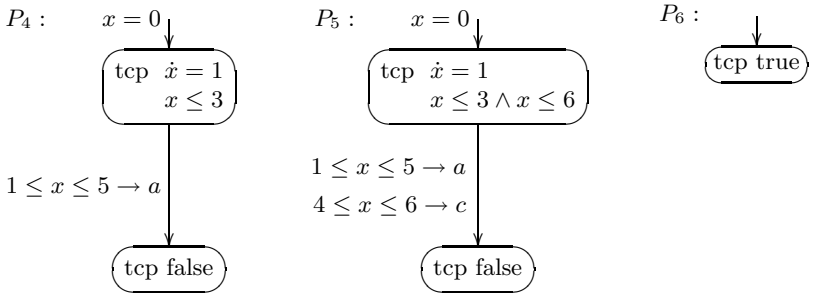


Fig. 4. μCIF automata P_4, P_5, and P_6, mimicking the behavior of TAD automata T_1, T_2, and stop, of Fig. 3, respectively

Impatient and patient synchronization. Above we showed how to impose hard deadlines by means of the tcp predicate. Alternatively, one could obtain the same behavior by means of the global urgency operator υ_U for $U \subseteq \mathcal{A}$. We put the automaton on which we want to impose urgency in parallel with an automaton that request urgency on synchronizing action transitions as depicted in Fig. 5. Here, we show how to alternatively specify the urgency of action a at time 3, obtaining μCIF automaton P_7 that is bisimilar to the automaton P_4 of Fig. 4.

An advantage of the specification of deadlines using the urgency operator, is that it provides for more flexibility. In particular, the scope of the urgency operator can be used to specify both impatient and patient synchronization of actions. In impatient synchronization, we have hard deadlines. Therefore, the urgency constraints of all synchronizing parties must be endorsed as soon as they are enabled. In patient synchronization we have soft deadlines. Some urgency constraints can be contravened so that the synchronization can occur as long as the guards still hold. To specify patient synchronization, we place the parallel composition *inside* the scope of the urgency operator. For example, $\upsilon_{\{a\}}(P_1 \parallel_{\{a\}} P_2)$, where P_1 and P_2 are given in Fig. 2 specifies the patient synchronization of P_1 and P_2. The resulting system has an outgoing guarded action transition $4 \leq x \leq 5 \rightarrow a$, which must be taken at time 4, restricted by tcp $x \leq 4$.

The urgency operator we defined, provides a reasonable amount of flexibility at specifying various types of synchronization. Ongoing research shows that the global urgency operator can be eliminated through symbolic reasoning at

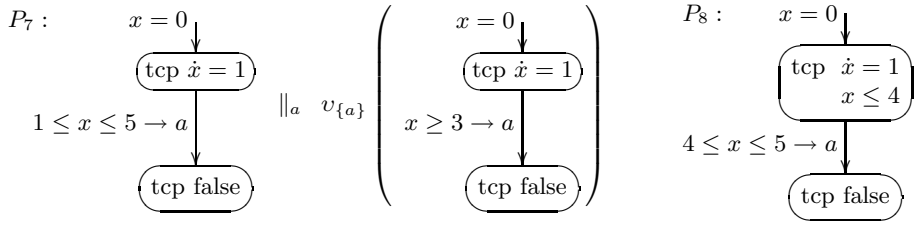


Fig. 5. The μCIF automaton P_7 shows an alternative definition of hard deadlines $P_7 \leftrightarrows P_4$; the μCIF automaton P_8 is the result of the patient synchronization of P_1 and P_2, i.e., $P_8 \leftrightarrows \upsilon_{\{a\}}(P_1 \|_{\{a\}} P_2)$.

the syntactic level, by transferring it to local urgency of the tcp predicate. We can achieve this by means of a linearization procedure that transforms μCIF automata by pushing the guards of the urgent transitions in the tcp predicate. As a consequence, guard trajectories never need to be actually calculated when proving properties of a system. Their only purpose is to retain compositionality when combining urgency with variable hiding. After a composition has been linearized, the need for guard trajectories disappears. The linearization approach can be illustrated by observing the μCIF automata P_4 of Fig. 4 and P_7 of Fig. 5. The guard of the urgent transitions a of P_7, given by $x \ge 3$, is pushed to the tcp predicate of the initial state of P_4, given by tcp $x \le 3$. Thus, the linearization procedure provides an efficient implementation of urgency in the CIF toolset [13].

5 Concluding Remarks

In this paper, we investigated the interplay between urgent actions and variable abstraction in timed and hybrid systems. We showed that this interaction is not distributive due to the use of timed transition systems as the basic semantic model, rendering component-wise verification inapplicable. We proposed to add guard trajectories in the labels of the timed transitions as a remedy. We illustrated the proposal by revising the semantics of the *CIF* language, as it is used in the MULTIFORM project. We proved that the identified problem indeed disappears, while retaining commonly desired properties of urgency and variable abstraction, such as compositionality and the possibility of specifying different kinds of synchronization. Our approach employs SOS techniques, guaranteeing compositionality with respect to stateless bisimilarity. Furthermore, it makes that the concepts introduced in this paper easily transferrable to other timed and hybrid formalisms. This has, e.g., already been done for the χ language [10]. Whether our results are useful in the verification of hybrid systems with urgency, remains as a topic for future research. Ongoing research based on linearization procedures that eliminate the urgency operator is promising and, moreover, the obtained results provide for local variable abstraction and verification based on this abstraction, an option not available in previous work.

Acknowledgements. We thank Albert Hofkamp for his original idea to introduce guards in the transition system, Ramon Schiffelers for his early involvement in this paper, and Mohammad Mousavi for his advise on the SOS.

References

1. Alur, R., Dill, D.: A theory of timed automata. Theoretical Computer Science 126, 183–235 (1994)
2. Larsen, K., Pettersson, P., Yi, W.: UPPAAL in a nutshell. International Journal on Software Tools for Technology Transfer 1(1-2), 134–152 (1997)
3. Bozga, M., Daws, C., Maler, O., Olivero, A., Tripakis, S., Yovine, S.: Kronos: A model-checking tool for real-time systems. In: Y. Vardi, M. (ed.) CAV 1998. LNCS, vol. 1427, pp. 546–550. Springer, Heidelberg (1998)
4. Bozga, M., Mounier, L., Graf, S.: If-2.0: A validation environment for component-based real-time systems. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, pp. 343–348. Springer, Heidelberg (2002)
5. Henzinger, T.: The theory of hybrid automata. In: Verification of Digital and Hybrid Systems. NATO ASI Series F: Computer and Systems Science, vol. 170, pp. 265–292. Springer, Heidelberg (2000)
6. Lynch, N., Segala, R., Vaandrager, F.: Hybrid I/O automata revisited. In: Di Benedetto, M.D., Sangiovanni-Vincentelli, A.L. (eds.) HSCC 2001. LNCS, vol. 2034, pp. 403–417. Springer, Heidelberg (2001)
7. Henzinger, T., Ho, P.H., Wong-toi, H.: HyTech: A model checker for hybrid systems. International Journal on Software Tools for Technology Transfer 1(1), 110–122 (1997)
8. Lee, E., Zheng, H.: Operational semantics of hybrid systems. In: Morari, M., Thiele, L. (eds.) HSCC 2005. LNCS, vol. 3414, pp. 25–53. Springer, Heidelberg (2005)
9. Reynolds, J.C.: Theories of programming languages. Cambridge University Press, New York (1999)
10. Beek, D.A.v., Man, K.L., Reniers, M.A., Rooda, J.E., Schiffelers, R.R.H.: Syntax and consistent equation semantics of hybrid Chi. Journal of Logic and Algebraic Programming 68(1-2), 129–210 (2006)
11. Plotkin, G.: A structural approach to operational semantics. Technical Report DAIMI FN-19, Computer Science Department, Aarhus University (1981)
12. Beek, D.A.v., Collins, P., Nadales, D.E., Rooda, J., Schiffelers, R.R.H.: New concepts in the abstract format of the compositional interchange format. In: ADHS 2009, pp. 250–255. IFAC (2009)
13. Systems Engineering Institute: The Compositional Interchange Format for Hybrid Systems (2010), `http://se.wtb.tue.nl/sewiki/cif/start`
14. Cuijpers, P., Reniers, M.: Lost in translation: Hybrid-time flows vs real-time transitions. In: Egerstedt, M., Mishra, B. (eds.) HSCC 2008. LNCS, vol. 4981, pp. 116–129. Springer, Heidelberg (2008)
15. Alur, R., Henzinger, T.: Real-time system = discrete system + clock variables. In: Theories and Experiences for Real-time System Development. AMAST Series in Computing, vol. 2, pp. 1–29. World Scientific, Singapore (1993)
16. Kaynar, D., Lynch, N., Segala, R., Vaandrager, F.: A framework for modelling timed systems with restricted hybrid automata. In: RTSS 2003, pp. 166–178. IEEE Computer Society Press, Los Alamitos (2003)

17. Bornot, S., Sifakis, J., Tripakis, S.: Modeling urgency in timed systems. In: de Roever, W.-P., Langmaack, H., Pnueli, A. (eds.) COMPOS 1997. LNCS, vol. 1536, pp. 103–129. Springer, Heidelberg (1998)
18. Sifakis, J.: The compositional specification of timed systems. In: Halbwachs, N., Peled, D.A. (eds.) CAV 1999. LNCS, vol. 1633, pp. 2–7. Springer, Heidelberg (1999)
19. Bornot, S., Sifakis, J.: An algebraic framework for urgency. Information and Computation 163, 172–202 (2000)
20. D'Argenio, P., Gebremichael, B.: The coarsest congruence for timed automata with deadlines contained in bisimulation. In: Abadi, M., de Alfaro, L. (eds.) CONCUR 2005. LNCS, vol. 3653, pp. 125–140. Springer, Heidelberg (2005)
21. Bowman, H.: Modeling timeouts without timelocks. In: Katoen, J.-P. (ed.) AMAST-ARTS 1999, ARTS 1999, and AMAST-WS 1999. LNCS, vol. 1601, pp. 20–35. Springer, Heidelberg (1999)
22. Bohnenkamp, H., D'Argenio, P., Hermanns, H., Katoen, J.P.: MODEST: A compositional modeling formalism for hard and softly timed systems. IEEE Transactions on Software Engineering 32(10), 812–830 (2006)
23. Gebremichael, B., Vaandrager, F.: Specifying urgency in timed I/O automata. In: SEFM 2005, pp. 64–74. IEEE Computer Society, Los Alamitos (2005)
24. Mufti, W., Tcherukine, D.: Integration of model-checking tools: from discrete to hybrid models. In: INMIC 2007, pp. 1–4. IEEE International, Los Alamitos (2007)
25. Bornot, S., Sifakis, J.: On the composition of hybrid systems. In: Henzinger, T.A., Sastry, S.S. (eds.) HSCC 1998. LNCS, vol. 1386, pp. 49–63. Springer, Heidelberg (1998)
26. van Beek, D., Hofkamp, A., Reniers, M., Rooda, J., Schiffelers, R.: Syntax and formal semantics of Chi 2.0. SE Report 2008-01, Eindhoven University of Technology (2008)
27. Cuijpers, P., Reniers, M., Heemels, W.: Hybrid transition systems. Technical Report CS-Report 02-12, TU/e, Eindhoven, Netherlands (2002)
28. van Beek, D.A., Reniers, M.A., Schiffelers, R.R.H., Rooda, J.E.: Foundations of a compositional interchange format for hybrid systems. In: Bemporad, A., Bicchi, A., Buttazzo, G. (eds.) HSCC 2007. LNCS, vol. 4416, pp. 587–600. Springer, Heidelberg (2007)
29. Scott, D.: Outline of a mathematical theory of computation. In: CISS 1970, Princeton, pp. 169–176 (1970)
30. Mousavi, M.R., Reniers, M.A., Groote, J.F.: Notions of bisimulation and congruence formats for SOS with data. Information and Computation 200(1), 107–147 (2005)

Computing Equilibria in Two-Player Timed Games *via* Turn-Based Finite Games*

Patricia Bouyer, Romain Brenguier, and Nicolas Markey

LSV, CNRS & ENS Cachan, France
{bouyer,brenguie,markey}@lsv.ens-cachan.fr

Abstract. We study two-player timed games where the objectives of the two players are not opposite. We focus on the standard notion of Nash equilibrium and propose a series of transformations that builds two finite turn-based games out of a timed game, with a precise correspondence between Nash equilibria in the original and in final games. This provides us with an algorithm to compute Nash equilibria in two-player timed games for large classes of properties.

1 Introduction

Timed games. Game theory (especially games played on graphs) has been used in computer science as a powerful framework for modelling interactions in embedded systems [16,13]. Over the last fifteen years, games have been extended with the ability to depend on timing informations, taking advantage of the large development of timed automata [1]. Adding timing constraints allows for a more faithful representation of reactive systems, while preserving decidability of several important properties, such as the existence of a winning strategy for one of the agents to achieve her goal, whatever the other agents do [3]. Efficient algorithms exist and have been implemented, *e.g.* in the tool Uppaal-Tiga [4].

Zero sum vs. non-zero sum games. In this purely antagonist view, games can be seen as two-player games, where one agent plays against another one. Moreover, the objectives of those two agents are opposite: the aim of the second player is simply to prevent the first player from winning her own objective. More generally, a (positive or negative) payoff can be associated with each outcome of the game, which can be seen as the amount the second player will have to pay to the first player. Those games are said to be zero-sum.

In many cases, however, games can be non-zero-sum: the objectives of the two players are then no more complementary, and the aim of one player is no more to prevent the other player from winning. Such games appear *e.g.* in various problems in telecommunications, where the agents try to send data on a network [12]. Focusing only on surely-winning strategies in this setting may then be too narrow: surely-winning strategies must be winning against any behaviour of the other player, and do not consider the fact that the other player also tries to achieve her own objective.

* This work is partly supported by projects DOTS (ANR-06-SETI-003), QUASIMODO (FP7-ICT-STREP-214755) and GASICS (ESF-EUROCORES LogiCCC).

K. Chatterjee and T.A. Henzinger (Eds.): FORMATS 2010, LNCS 6246, pp. 62–76, 2010.

Nash equilibria. In the non-zero-sum game setting, it is then more interesting to look for equilibria. One of the most-famous and most-studied notion of equilibrium is that proposed by Nash in 1950 [14]: a *Nash equilibrium* is a behaviour of the players in which they act rationally, in the sense that no player can get a better payoff if she, alone, modifies her strategy [14]. Notice that a Nash equilibrium needs not exist in general, and may not be optimal, in the sense that several equilibria can coexist, and may have very different payoffs.

Our contribution. We extend the standard notion of Nash equilibria to timed games, where non-determinism naturally arises and has to be taken into account. We propose a whole chain of transformations that builds, given a two-player timed game, two turn-based finite games which, in some sense that we will make precise, preserve Nash equilibria. The first transformation consists in building a finite concurrent game with non-determinism based on the classical region abstraction; the second transformation decouples this concurrent game into two concurrent games, one per player: in each game, the preference relation of one of the players is simply dropped, but we have to consider "joint" equilibria. The last two transformations work on each of the two copies of the concurrent game: the first one solves the non-determinism by giving an advantage to the associated player, and the last one makes use of this advantage to build a turn-based game equivalent to the original concurrent game. This chain of transformations is valid for the whole class of two-player timed games, and Nash equilibria are preserved for a large class of objectives, for instance ω-regular objectives[1]. These transformations allow to recover some known results about zero-sum games, but also to get new decidability results for Nash equilibria in two-player timed games.

Related work. Nash equilibria (and other related solution concepts such as *subgame-perfect equilibria*, *secure equilibria*, ...) have recently been studied in the setting of (untimed) games played on a graph [8,9,10,15,17,18,19,20]. None of them, however, focuses on timed games. In the setting of concurrent games, mixed strategies (*i.e.*, strategies involving probabilistic choices) are arguably more relevant than pure (*i.e.*, non-randomized) strategies. However, adding probabilities to timed strategies involves several important technical issues (even in zero-sum non-probabilistic timed games), and we defer the study of mixed-strategy Nash equilibria in two-player timed games to future works.

For lack of space, proofs are omitted and can be found in [5].

2 Preliminaries

2.1 Timed Games

A *valuation* over a finite set of clocks Cl is a mapping $v\colon \mathsf{Cl} \to \mathbb{R}_+$. If v is a valuation and $t \in \mathbb{R}_+$, then $v + t$ is the valuation that assigns to each $x \in \mathsf{Cl}$ the value $v(x)+t$. If v is a valuation and $Y \subseteq \mathsf{Cl}$, then $[Y \leftarrow 0]v$ is the valuation that assigns 0 to each $y \in Y$ and $v(x)$ to each $x \in \mathsf{Cl} \setminus Y$. A *clock constraint* over Cl is a formula built on the

[1] In the general case, the undecidability results on (zero-sum) priced timed games entail undecidability of the existence of Nash equilibria.

grammar: $\mathfrak{C}(\mathsf{Cl}) \ni g ::= x \sim c \mid g \wedge g$, where x ranges over Cl, $\sim \in \{<, \leq, =, \geq, >\}$, and c is an integer. The semantics of clock constraints over valuations is natural, and we omit it.

Definition 1. *A* timed automaton *is a tuple* $\langle \mathit{Loc}, \mathit{Cl}, \mathit{Inv}, \mathit{Trans} \rangle$ *such that:*

- Loc *is a finite set of locations;*
- Cl *is a finite set of clocks;*
- $\mathit{Inv}\colon \mathit{Loc} \to \mathfrak{C}(\mathit{Cl})$ *assigns an* invariant *to each location;*
- $\mathit{Trans} \subseteq \mathit{Loc} \times \mathfrak{C}(\mathit{Cl}) \times 2^{\mathit{Cl}} \times \mathit{Loc}$ *is the set of transitions.*

We assume the reader is familiar with timed automata [1], and in particular with states (pairs $(\ell, v) \in \mathsf{Loc} \times \mathbb{R}_+^X$ such that $v \models \mathsf{Inv}(\ell)$), runs (seen as infinite sequences of states for our purpose), *etc*. We now define the notion of two-player timed games. The two players will be called player 1 and player 2. Our definition follows that of [11].

Definition 2. *A* (two-player) timed game *is a tuple* $\mathcal{G} = \langle \mathit{Loc}, \mathit{Cl}, \mathit{Inv}, \mathit{Trans}, \mathit{Owner}, (\preccurlyeq_1, \preccurlyeq_2) \rangle$ *where:*

- $\langle \mathit{Loc}, \mathit{Cl}, \mathit{Inv}, \mathit{Trans} \rangle$ *is a timed automaton;*
- $\mathit{Owner}\colon \mathit{Trans} \to \{1, 2\}$ *assigns a player to each transition;*
- *for each* $i \in \{1, 2\}$, $\preccurlyeq_i \subseteq \left(\mathit{Loc} \times \mathbb{R}_+^{\mathit{Cl}}\right)^\omega \times \left(\mathit{Loc} \times \mathbb{R}_+^{\mathit{Cl}}\right)^\omega$ *is a quasi-order on runs of the timed automaton, called the* preference relation *for player* i.

A timed game is played as follows: from each state of the underlying timed automaton (starting from an initial state $s_0 = (\ell, \mathbf{0})$, where $\mathbf{0}$ maps each clock to zero), each player chooses a nonnegative real number d and a transition δ, with the intended meaning that she wants to delay for d time units and then fire transition δ. There are several (natural) restrictions on these choices:

- spending d time units in ℓ must be allowed[2] *i.e.*, $v + d \models \mathsf{Inv}(\ell)$;
- $\delta = (\ell, g, z, \ell')$ belongs to the current player (given by function Owner);
- the transition is firable after d time units (*i.e.*, $v + d \models g$), and the invariant is satisfied when entering ℓ' (*i.e.*, $[z \leftarrow 0](v + d) \models \mathsf{Inv}(\ell')$).

When there is no such possible choice for a player (for instance if there is no transition from ℓ belonging to that player), she chooses a special move, denoted by $\perp$.

From a state (ℓ, v) and given a choice (m_1, m_2) for the two players, with $m_i \in (\mathbb{R}_+ \times \mathsf{Trans}) \cup \{\perp\}$, an index i_0 such that $d_{i_0} = \min\{d_i \mid m_i = (d_i, \delta_i) \text{ and } i \in \{1, 2\}\}$ is selected (non-deterministically if both delays are identical), and the corresponding transition $\delta_{i_0} = (\ell, g, z, \ell')$ is applied, leading to a new state $(\ell', [z \leftarrow 0](v + d_{i_0}))$. To ensure well-definedness of the above semantics we assume in the sequel that timed games are *non-blocking*, that is, for any reachable state (ℓ, v), at least one player has an allowed transition (this avoids that both players play the special action $\perp$).

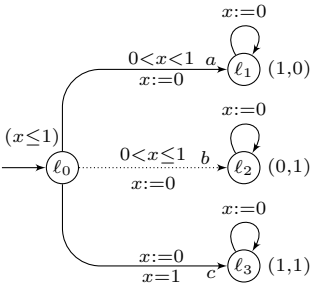
The outcome of such a game when players have fixed their various choices is a run of the underlying timed automaton, that is an element of $\left(\mathsf{Loc} \times \mathbb{R}_+^{\mathsf{Cl}}\right)^\omega$, and possible

[2] Formally, this should be written $v + d' \models \mathsf{Inv}(\ell)$ for all $0 \leq d' \leq d$, but this is equivalent to having only $v \models \mathsf{Inv}(\ell)$ and $v + d \models \mathsf{Inv}(\ell)$ since invariants are convex.

outcomes are compared by each player using their preference relations. In the examples, we will define the preference relation of a player by assigning a value (called a *payoff*) to each possible outcome of the game, and the higher the payoff, the better the run in the preference relation.

This semantics can naturally be formalized in terms of an infinite-state non-deterministic concurrent game and strategies, that we will detail in the next section.

Example 1. We give an example of a timed game, that we will use as a running example: consider the timed game $\mathcal{G}$ on the right. When relevant the name of a transition is printed on the corresponding edge. Owners of the transitions are specified as follows: player 1 plays with plain edges, whereas player 2 plays with dotted edges. On the right of these locations we indicate payoffs for the two players (if a play ends up in ℓ_1, player 1 gets payoff 1, whereas player 2 gets payoff 0). Hence player 1 will prefer runs ending in ℓ_1 or ℓ_3 than runs ending in ℓ_2.



2.2 Concurrent Games

In this section we define two-player concurrent games, which we then use to encode the formal semantics of timed games. A *transition system* is a 2-tuple $\mathcal{S} = \langle \mathsf{States}, \mathsf{Edg} \rangle$ where States is a (possibly uncountable) set of states, and $\mathsf{Edg} \subseteq \mathsf{States} \times \mathsf{States}$ is the set of transitions. A *path* π in $\mathcal{S}$ is a non-empty sequence $(s_i)_{0 \leq i < n}$ (where $n \in \mathbb{N} \cup \{+\infty\}$) of states of $\mathcal{S}$ such that $(s_i, s_{i+1}) \in \mathsf{Edg}$ for all $i < n-1$. The *length* of π, denoted by $|\pi|$ is $n-1$. The set of finite paths (also called *histories* in the sequel) of $\mathcal{S}$ is denoted by[3] $\mathsf{Hist}_{\mathcal{S}}$, the set of infinite paths (also called *plays*) of $\mathcal{S}$ is denoted by $\mathsf{Play}_{\mathcal{S}}$, and $\mathsf{Path}_{\mathcal{S}} = \mathsf{Hist}_{\mathcal{S}} \cup \mathsf{Play}_{\mathcal{S}}$ is the set of paths of $\mathcal{S}$. Given a path $\pi = (s_i)_{0 \leq i < n}$ and an integer $j \leq |\pi|$, the *j-th prefix* of π, denoted by $\pi_{\leq j}$, is the finite path $(s_i)_{0 \leq i < j+1}$. If $\pi = (s_i)_{0 \leq i < n}$ is a history, we write $\mathsf{last}(\pi) = s_{|\pi|}$.

We extend the definition of concurrent games given *e.g.* in [2] with non-determinism:

Definition 3. *A* (two-player non-deterministic) concurrent game *is a tuple* $\mathcal{G} = \langle \mathsf{States}, \mathsf{Edg}, \mathsf{Act}, \mathsf{Mov}, \mathsf{Tab}, (\preccurlyeq_1, \preccurlyeq_2) \rangle$ *in which:*

- $\langle \mathsf{States}, \mathsf{Edg} \rangle$ *is a transition system;*
- Act *is a (possibly uncountable) set of actions;*
- $\mathsf{Mov} \colon \mathsf{States} \times \{1,2\} \to 2^{\mathsf{Act}} \setminus \{\varnothing\}$ *is a mapping indicating the actions available to each player in a given state;*
- $\mathsf{Tab} \colon \mathsf{States} \times \mathsf{Act}^2 \to 2^{\mathsf{Edg}} \setminus \{\varnothing\}$ *associates to each state and each pair of actions the set of resulting edges. It is required that if* $(s', s'') \in \mathsf{Tab}(s, (m_1, m_2))$*, then* $s' = s$*.*
- *for each* $i \in \{1,2\}$*,* $\preccurlyeq_i \subseteq \mathsf{States}^\omega \times \mathsf{States}^\omega$ *is a quasi-order called the* preference relation *for player* i*.*

[3] For this and the following definitions, we explicitly mention the underlying transition system as a subscript. In the sequel, we may omit this subscript when the transition system is clear from the context.

A *deterministic* concurrent game is a concurrent game where $\mathsf{Tab}(s,(m_1,m_2))$ is a singleton for every $s \in \mathsf{States}$ and $(m_1,m_2) \in \mathsf{Mov}(s,1) \times \mathsf{Mov}(s,2)$. A *turn-based* game is a concurrent game for which there exists a mapping $\mathsf{Owner}\colon \mathsf{States} \to \{1,2\}$ such that, for every state $s \in \mathsf{States}$, the set $\mathsf{Mov}(s,i)$ is a singleton unless $\mathsf{Owner}(s) = i$.

In a concurrent game, from some state s, each player i selects one action m_i among its set $\mathsf{Mov}(s,i)$ of allowed actions (the resulting pair (m_1,m_2) is called a *move*). This results in a set of edges $\mathsf{Tab}(s,(m_1,m_2))$, one of which is applied and gives the next state of the game. In the sequel, we abusively write $\mathsf{Hist}_{\mathcal{G}}$, $\mathsf{Play}_{\mathcal{G}}$ and $\mathsf{Path}_{\mathcal{G}}$ for the corresponding set of paths in the underlying transition system of $\mathcal{G}$. We also write $\mathsf{Hist}_{\mathcal{G}}(s)$, $\mathsf{Play}_{\mathcal{G}}(s)$ and $\mathsf{Path}_{\mathcal{G}}(s)$ for the respective subsets of paths starting in state s.

Definition 4. *Let $\mathcal{G}$ be a concurrent game, and $i \in \{1,2\}$. A* strategy *for player i is a mapping $\sigma_i\colon \mathsf{Hist}_{\mathcal{G}} \to \mathsf{Act}$ such that $\sigma_i(\pi) \in \mathsf{Mov}(\mathsf{last}(\pi),i)$ for all $\pi \in \mathsf{Hist}_{\mathcal{G}}$. A* strategy profile *is a pair (σ_1,σ_2) where σ_i is a player-i strategy. We write $\mathsf{Strat}^i_{\mathcal{G}}$ for the set of strategies of player i in $\mathcal{G}$, and $\mathsf{Prof}_{\mathcal{G}}$ for the set of strategy profiles in $\mathcal{G}$.*

Notice that we only consider non-randomized (*pure*) strategies in this paper.

Let $\mathcal{G}$ be a concurrent game, $i \in \{1,2\}$, and σ_i be a player i-strategy. A path $\pi = (s_j)_{0\le j\le|\pi|}$ is *compatible* with the strategy σ_i if, for all $k \le |\pi| - 1$, there exists a pair of actions $(m_1,m_2) \in \mathsf{Act}^2$ such that $m_j \in \mathsf{Mov}(s_k,j)$ for all $j \in \{1,2\}$, $m_i = \sigma_i(\pi_{\le k})$, and $(s_k,s_{k+1}) \in \mathsf{Tab}(s_k,(m_1,m_2))$. A path π is compatible with a strategy profile (σ_1,σ_2) whenever it is compatible with both strategies σ_1 and σ_2. We write $\mathsf{Out}_{\mathcal{G},s}(\sigma_i)$ (resp. $\mathsf{Out}_{\mathcal{G},s}(\sigma_1,\sigma_2)$) for the set of paths from s (also called *outcomes*) in $\mathcal{G}$ that are compatible with strategy σ_i (resp. strategy profile (σ_1,σ_2)). Notice that, in the case of deterministic concurrent games, a strategy profile has a single infinite outcome. This might not be the case for non-deterministic concurrent games.

Given a move (m_1,m_2) and a new action m' for player i, we write $(m_1,m_2)_{[i\mapsto m']}$ for the move (n_1,n_2) with $n_i = m'$ and $n_{3-i} = m_{3-i}$. This notation is extended to strategies in a natural way.

In the context of non-zero-sum games, several notions of equilibria have been defined. We present a refinement of *Nash equilibria* towards non-deterministic concurrent games.

Definition 5. *Let $\mathcal{G}$ be a concurrent game, and s be a state of $\mathcal{G}$. A* pseudo Nash equilibrium *in $\mathcal{G}$ from s is a tuple $((\sigma_1,\sigma_2),\pi)$ where $(\sigma_1,\sigma_2) \in \mathsf{Prof}_{\mathcal{G}}$, and $\pi \in \mathsf{Out}_{(\mathcal{G},s)}(\sigma_1,\sigma_2)$ is such that for all $i \in \{1,2\}$ and all $\sigma'_i \in \mathsf{Strat}^i_{\mathcal{G}}$, it holds:*

$$\forall \pi' \in \mathsf{Out}_{(\mathcal{G},s)}((\sigma_1,\sigma_2)_{[i\mapsto\sigma'_i]}).\ \pi' \preccurlyeq_i \pi.$$

Such an outcome π is called a best play *for the strategy profile (σ_1,σ_2).*

In the case of deterministic games, π is uniquely determined by (σ_1,σ_2), and pseudo Nash equilibria coincide with *Nash equilibria* as defined in [14]: they are strategy profiles where no player has an incentive to unilaterally deviate from her strategy.

In the case of non-deterministic games, a strategy profile for an equilibrium may give rise to several outcomes. The choice of playing the best play π is then made cooperatively by both players: once both strategies are fixed, it is the interest of both players to cooperate and play "optimally".

2.3 Back to Timed Games

Two comments are in order here: (i) non-determinism in timed games could be dropped by giving priority to one of the players, in case both of them play the same delay. Our algorithm could of course be adapted in this case; (ii) even if the timed game were deterministic, our transformation to region games involves some extra non-determinism. As will be seen in the sequel, the above notion of *pseudo Nash equilibria* is the notion we need for our construction to preserve equilibria.

It is easy to see the semantics of a timed game as the semantics of an infinite-state concurrent game (see the research report [5]). Using that point-of-view, timed games inherit the notions of history, play, path, strategy, profile, outcome and pseudo Nash equilibrium. We illustrate some of these notions on the running example.

Example 1 (Cont'd). This game starts in configuration $(\ell_0, 0)$ (clock x is set to 0). A strategy profile is then determined by an initial choice for the first transition. If one of the players choose some delay smaller than 1, she will have payoff 1 but the other player will have payoff 0, hence the other player will be able to preempt this choice and choose a smaller delay that will improve her own payoff. Hence there will be no such pseudo Nash equilibrium. There is a single pseudo Nash equilibrium, where player 1 chooses $(1, c)$ (delay for 1 t.u. and take transition c) and player 2 chooses $(1, b)$. The best play for that strategy profile is the run taking transition c.

In this paper we will be interested in the computation of pseudo Nash equilibria in timed games. To do so we propose a sequence of transformations that will preserve equilibria (in some sense), yielding the construction of two turn-based finite games in which the initial problem will be reduced to the computation of *twin* Nash equilibria. All these transformations are presented in the next section. These transformations will also give a new point-of-view on timed games, which we will use in Section 4.2 to recover some decidability results. Many more results are expected.

3 From Timed Games to Turn-Based Finite Games

In this section we propose a chain of transformations of the timed game $\mathcal{G}$ into two turn-based finite games, and reduce the computation of pseudo Nash equilibria in $\mathcal{G}$ to the computation of 'twin' Nash equilibria in the two turn-based games. Notice that we will have to impose restrictions on the preference relations: indeed, price-optimal reachability is undecidable in two-player priced timed games, and these quantitative objectives can be encoded as a payoff function, see [7] for details.

3.1 From Timed Games to Concurrent Games...

We assume the reader is familiar with the region automaton abstraction for timed automata [1]. Let $\mathcal{G} = \langle \mathsf{Loc}, \mathsf{Cl}, \mathsf{Inv}, \mathsf{Trans}, \mathsf{Owner}, (\preccurlyeq_1, \preccurlyeq_2) \rangle$ be a timed game. Let $\mathfrak{R}$ be the set of regions for the timed automaton underlying $\mathcal{G}$, and $\pi_{\mathfrak{R}}$ be the projection over the regions $\mathfrak{R}$ (for configurations, runs, *etc.*) We define the *region game* $\mathcal{R} = \langle \mathsf{States}, \mathsf{Edg}, \mathsf{Act}, \mathsf{Mov}, \mathsf{Tab}, (\preccurlyeq_1^{\mathcal{R}}, \preccurlyeq_2^{\mathcal{R}}) \rangle$ as follows:

- $\mathsf{States} = \{(\ell, r) \in \mathsf{Loc} \times \mathfrak{R} \mid r \models \mathsf{Inv}(\ell)\}$;
- $\mathsf{Edg} = \{((\ell, r), (\ell', r')) \mid (\ell, r) \to (\ell', r')$ in the region automaton of $\mathcal{G}\}$;
- $\mathsf{Act} = \{\bot\} \cup \{(r, \delta) \mid r \in \mathfrak{R}$ and $\delta \in \mathsf{Trans}\}$;
- $\mathsf{Mov}\colon \mathsf{States} \times \{1, 2\} \to 2^{\mathsf{Act}} \setminus \{\varnothing\}$ such that:

$$\mathsf{Mov}((\ell, r), i) = \{(r', \delta) \mid r' \in \mathsf{Succ}(r),\ r' \models \mathsf{Inv}(\ell), \delta = (\ell, g, Y, \ell') \text{ is s.t. } r' \models g \text{ and } [Y \leftarrow 0]r' \models \mathsf{Inv}(\ell') \text{ and } \mathsf{Owner}(\delta) = i\}$$

  if this set is non-empty, and $\mathsf{Mov}((\ell, r), i) = \{\bot\}$ otherwise.
- $\mathsf{Tab}\colon \mathsf{States} \times \mathsf{Act}^2 \to 2^{\mathsf{Edg}} \setminus \{\varnothing\}$ such that for every $(\ell, r) \in \mathsf{States}$ and every $(m_1, m_2) \in \mathsf{Mov}((\ell, r), 1) \times \mathsf{Mov}((\ell, r), 2)$, if we write r' for $\min\{r_j \mid j \in \{1, 2\}$ and $m_j = (r_j, \delta_j)\}$,[4] then we have:

$$\mathsf{Tab}((\ell, r), (m_1, m_2)) = \{((\ell, r), (\ell_j, [Y_j \leftarrow 0]r_j)) \mid j \in \{1, 2\} \text{ and } m_j = (r_j, \delta_j) \text{ with } r_j = r',\ (\ell, g_j, Y_j, \ell_i) = \delta_j \text{ and } r_j \models g_j\}$$

- The preference relation $\preccurlyeq_i^{\mathcal{R}}$ for player i is defined by saying that $\gamma \preccurlyeq_i^{\mathcal{R}} \gamma'$ iff there exists ρ and ρ' such that $\pi_{\mathfrak{R}}(\rho) = \gamma$, $\pi_{\mathfrak{R}}(\rho') = \gamma'$ and $\rho \preccurlyeq_i \rho'$.

Note that the game $\mathcal{R}$ is non-deterministic, even if the original timed game is not. Indeed, non-determinism appears when players want to play delays leading to the same region. The (relative) order of the choices for the delays chosen by the two players cannot be distinguished by the region abstraction.

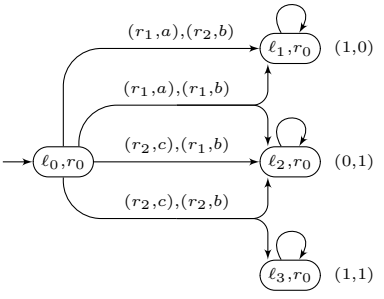
Definition 6. *A preference relation $\preccurlyeq_i$ is said to be* region-uniform *when for all plays ρ and ρ', if the sequence of regions seen in both paths are the same, then they are equivalent, i.e. $\rho \preccurlyeq_i \rho'$ and $\rho' \preccurlyeq_i \rho$.*

Proposition 7. *Let $\mathcal{G}$ be a timed game, and assume that the two preference relations of $\mathcal{G}$ are region-uniform. Let $\mathcal{R}$ be its associated region game. Then there is a pseudo Nash equilibrium in $\mathcal{G}$ from $(\ell_0, \mathbf{0})$ with best play ρ iff there is a pseudo Nash equilibrium in $\mathcal{R}$ from $(\ell_0, [\mathbf{0}]_{\mathfrak{R}})$ with best play $\pi_{\mathfrak{R}}(\rho)$. Furthermore, this equivalence is constructive.*

Example 1 (Cont'd). We illustrate the construction and the previous notions on the running example. We write r_0 (resp. r_1, r_2) for the region $x = 0$ (resp. $0 < x < 1$, $x = 1$). The region game $\mathcal{R}$ is as depicted on Fig. 1. In this region game, there are two non-deterministic transitions. First when the two players choose to wait until region r_2, in which case the game can turn to either ℓ_2 or to ℓ_3. Then when both players choose to move within the region r_1 (there is an uncertainty on whether player 1 or player 2 was faster), and depending on who was faster, the game will move to either ℓ_1 or ℓ_2. The first non-determinism is inherent to the game (and could be removed by construction assuming one player is more powerful, see Subsection 2.3 for explanations), whereas the second non-determinism is (somehow) artificial and comes from the region abstraction.

In $\mathcal{G}$, there is a single pseudo Nash equilibrium, where both players wait until $x = 1$ (region r_2), and propose to move respectively to ℓ_3 (resp. ℓ_2). The best play is then $(\ell_0, 0)(\ell_3, 0)^*$. This corresponds to the unique pseudo Nash equilibrium that we find in the region game.

[4] This is well-defined because both r_j's are time-successors of r.



The transition table from (ℓ_0, r_0) (*i.e.*, $\mathsf{Tab}((\ell_0, r_0), (m_1, m_2))$):

	$m_2 = (r_1, b)$	$m_2 = (r_2, b)$
$m_1 = (r_1, a)$	$(\ell_1, r_0), (\ell_2, r_0)$	(ℓ_1, r_0)
$m_1 = (r_2, c)$	(ℓ_2, r_0)	$(\ell_2, r_0), (\ell_3, r_0)$

Fig. 1. The region game from our original automaton

3.2 ... Next to Two Twin Concurrent Games...

Given a concurrent non-deterministic finite game $\mathcal{R} = \langle \mathsf{States}, \mathsf{Edg}, \mathsf{Act}, \mathsf{Mov}, \mathsf{Tab}, (\preccurlyeq_1^{\mathcal{R}}, \preccurlyeq_2^{\mathcal{R}}) \rangle$, we construct two concurrent games $\mathcal{R}_1$ and $\mathcal{R}_2$ where we simply forget the preferences of one player. Formally for $i \in \{1, 2\}$, we define the game $\mathcal{R}_i = \langle \mathsf{States}, \mathsf{Edg}, \mathsf{Act}, \mathsf{Mov}, \mathsf{Tab}, (\preccurlyeq_1^i, \preccurlyeq_2^i) \rangle$, where $\preccurlyeq_i^i$ is the quasi-order $\preccurlyeq_i$, and $\preccurlyeq_{3-i}^i$ is the trivial quasi-order where all runs are equivalent.

Definition 8. *A* twin pseudo Nash equilibrium *for the two games $\mathcal{R}_1$ and $\mathcal{R}_2$ is a tuple $((\sigma_1^{\mathcal{R}_1}, \sigma_2^{\mathcal{R}_1}), (\sigma_1^{\mathcal{R}_2}, \sigma_2^{\mathcal{R}_2}), \rho)$ such that $((\sigma_1^{\mathcal{R}_1}, \sigma_2^{\mathcal{R}_1}), \rho)$ is a pseudo Nash equilibrium in the game $\mathcal{R}_1$ and $((\sigma_1^{\mathcal{R}_2}, \sigma_2^{\mathcal{R}_2}), \rho)$ is a pseudo Nash equilibrium in the game $\mathcal{R}_2$. We furthermore say that ρ is a* best play *for the twin pseudo equilibrium.*

We relate pseudo Nash equilibria in $\mathcal{R}$ with twin pseudo Nash equilibria in $\mathcal{R}_1$ and $\mathcal{R}_2$. Note that we require best plays be the same, but not strategies.

Proposition 9. *Let $\mathcal{R}$ be the region game associated with some timed game $\mathcal{G}$. Then there is a pseudo Nash equilibrium in $\mathcal{R}$ from s with best play γ if and only if there is a twin pseudo equilibrium for the corresponding games $\mathcal{R}_1$ and $\mathcal{R}_2$ from s with best play γ. Furthermore this equivalence is constructive.*

3.3 ... Next to Concurrent Deterministic Games...

We transform each game $\mathcal{R}_i$ into a concurrent deterministic game $\mathcal{C}_i$. Game $\mathcal{C}_i$ will give priority to player i, in that it will be the role of player i to solve non-determinism. The game $\mathcal{C}_i = \langle \mathsf{States}, \mathsf{Edg}, \mathsf{Act}', \mathsf{Mov}_i, \mathsf{Tab}_i, (\preccurlyeq_1^i, \preccurlyeq_2^i) \rangle$ is defined as follows:

- $\mathsf{Act}' = \mathsf{Act} \cup ((\mathsf{Act} \setminus \{\bot\}) \times \{\bullet, \circ\})$;
- $\mathsf{Mov}_i \colon \mathsf{States} \times \{1, 2\} \to 2^{\mathsf{Act}'} \setminus \{\varnothing\}$ such that:

$$\mathsf{Mov}_i(s, i) = \begin{cases} \{\bot\} & \text{if } \mathsf{Mov}(s, i) = \{\bot\} \\ \mathsf{Mov}(s, i) \times \{\bullet, \circ\} & \text{otherwise} \end{cases}$$

$$\mathsf{Mov}_i(s, 3 - i) = \mathsf{Mov}(s, 3 - i)$$

– Given $(m_1, m_2) \in \mathsf{Mov}(s,1) \times \mathsf{Mov}(s,2)$ we have that $\mathsf{Tab}(s,(m_1,m_2))$ has at least one element, and at most two elements.[5]
  - In case it has only one element, then setting $m'_{3-i} = m_{3-i}$ and picking $m'_i \in \{(m_i,\bullet),(m_i,\circ)\}$, we define: $\mathsf{Tab}_i(s,(m'_1,m'_2)) = \mathsf{Tab}(s,(m_1,m_2))$;
  - In case it has two elements, say $(s,s_\bullet)$ and $(s,s_\circ)$, one of them comes from a transition of player i in $\mathcal{G}$ and the other comes from a transition of player $3-i$ in $\mathcal{G}$. Hence w.l.o.g. we can assume that $(s,s_\bullet)$ belongs to player i. We now define $m'_{3-i} = m_{3-i}$ and for any $m'_i \in \{(m_i,\bullet),(m_i,\circ)\}$, we define:

$$\mathsf{Tab}_i(s,(m'_1,m'_2)) = \begin{cases} \{(s,s_\bullet)\} & \text{if } m'_i = (m_i,\bullet) \\ \{(s,s_\circ)\} & \text{if } m'_i = (m_i,\circ) \end{cases}$$

By construction, the two games $\mathcal{C}_1$ and $\mathcal{C}_2$ are deterministic, and they share the same structure. Only decisions on how to solve non-determinism are made by different players. Our aim will be to compute equilibria in these two similar games.

Proposition 10. *Assume $\mathcal{C}_i$ (with $i \in \{1,2\}$) is the deterministic concurrent game defined from the concurrent game $\mathcal{R}_i$. Then there is a pseudo Nash equilibrium in $\mathcal{R}_i$ from s with best play γ iff there is a Nash equilibrium in $\mathcal{C}_i$ from s with best play γ.*[6] *Furthermore this equivalence is constructive.*

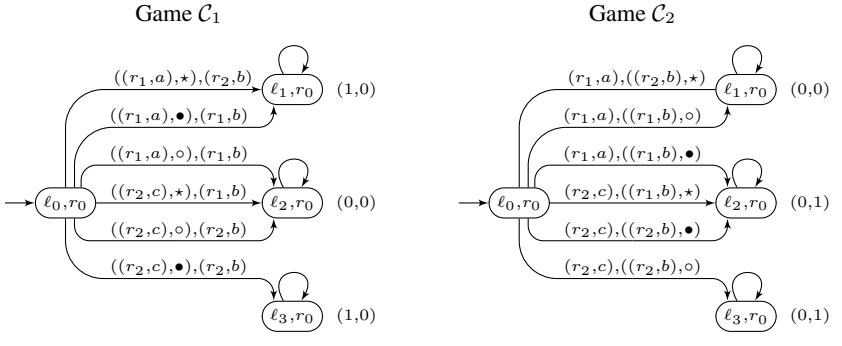


Fig. 2. Two concurrent games $\mathcal{C}_1$ and $\mathcal{C}_2$ from our original automaton

Example 1 (Cont'd). We build on the previous example, and give the two games $\mathcal{C}_1$ and $\mathcal{C}_2$ in Fig. 2. An action $(m,\star)$ denotes either $(m,\bullet)$ or $(m,\circ)$. There are several Nash equilibria in game $\mathcal{C}_1$: one where the first player chooses $((r_2,c),\bullet)$ and the second player chooses (r_2,b), which leads to (ℓ_3,r_0) with payoff $(1,0)$; and one where both players play a pair of actions leading to (ℓ_1,r_0), in which case the payoff is also $(1,0)$.

Similarly there are several Nash equilibria in game $\mathcal{C}_2$: one where the second player chooses $((r_2,b),\circ)$ and the first player chooses (r_2,c), which leads to (ℓ_3,r_0) with

[5] This is because the game $\mathcal{G}$ is non-blocking, and in this game, each player proposes her choice for a transition, and one of these two transitions will be chosen.

[6] Remember that $\mathcal{C}_i$'s and $\mathcal{R}_i$'s share the same structure and have the same runs.

payoff $(0,1)$; the second one where both players play a pair of actions leading to (ℓ_2, r_0), in which case the payoff is also $(0,1)$.

There is a single twin equilibrium in $\mathcal{C}_1$ and $\mathcal{C}_2$, namely the one leading to state (ℓ_3, r_0), which coincides with those equilibria already found in $\mathcal{G}$ and $\mathcal{R}$.

3.4 ... And Finally to Two Turn-Based Games

In the (deterministic) concurrent game $\mathcal{C}_i$, the advantage is given to player i, who has the ability to solve non-determinism. We can give a slightly different interpretation to that mechanism, which takes into account an interpretation of the new actions. Indeed, actions have a timed interpretation in the original timed game, and can be ordered w.r.t. their delay. Taking advantage of this order on actions, we build a turn-based game $\mathcal{T}_i$.

Let $\mathcal{C}_i = \langle \mathsf{States}, s_0, \mathsf{Edg}, \mathsf{Act}', \mathsf{Mov}_i, \mathsf{Tab}_i, (\preccurlyeq_1^i, \preccurlyeq_2^i)\rangle$ be the games obtained from the previous construction. Let $s \in \mathsf{States}$. We naturally order the set $\mathsf{Mov}_i(s,1) \cup \mathsf{Mov}_i(s,2)$ with a relation $<_s$ so that:

- (i) if $\bot \in \mathsf{Mov}_i(s,1) \cup \mathsf{Mov}_i(s,2)$ then $\bot$ is maximal w.r.t. $<_s$;
- (ii) for every $m \in \mathsf{Mov}_i(s,j)$, there exists $s' \in \mathsf{States}$ such that for every $m' \in \mathsf{Mov}_i(s,3-j)$, $m <_s m'$ implies $\mathsf{Tab}_i(s,(m,m')) = \{(s,s')\}$.

This is possible due to the definition of game $\mathcal{C}_i$: when (r, δ_{3-i}) is allowed to player $3-i$ from s, and $((r,\delta_i),\bullet)$ and $((r,\delta_i),\circ)$ are allowed to player i from s, then the three actions are totally ordered by $<_s$ as follows:[7] $((r,\delta_i),\bullet) <_s (r,\delta_{3-i}) <_s ((r,\delta_i),\circ)$. Intuitively an action with marker $\bullet$ means that player i can play her own transition faster than player $3-i$ can play her own transition, but also that she can decide to play more slowly (role of action with marker $\circ$).

We can also define an equivalence relation $=_s$ compatible with this order, by saying $m =_s m' \Leftrightarrow m, m' \in \mathsf{Mov}_i(s,1) \cup \mathsf{Mov}_i(s,2),\ m \not<_s m'$ and $m' \not<_s m$. It is worth noticing that $m =_s m'$ implies that they belong to the same player. This can be the case if two transitions are available to a player from the same region, and also if a player can only play action $\bot$. We will write $[m]_s$ for the equivalence class associated to m. We next say that $[m]_s$ belongs to player j whenever all actions in $[m]_s$ belong to player j.

Example 1 (Cont'd). Consider games $\mathcal{C}_1$ and $\mathcal{C}_2$ depicted in Fig. 2. In game $\mathcal{C}_1$, the order on actions (written simply as $<$) from (ℓ_0, r_0) is given by:

$$\begin{array}{ccccccccc} ((r_1,a),\bullet) & < & (r_1,b) & < & ((r_1,a),\circ) & < & ((r_2,c),\bullet) & < & (r_2,b) & < & ((r_2,c),\circ) \\ \wr & & \wr & & \wr & & \wr & & \wr & & \\ (\ell_1,r_0) & & (\ell_2,r_0) & & (\ell_1,r_0) & & (\ell_3,r_0) & & (\ell_2,r_0) & & \end{array}$$

Below each action we write the target state when this action is played, provided an action smaller (for the order $<$) is not played by the other player. There is no target with action $((r_2,c),\circ)$ because it is always preempted by some 'faster' action (no $\bot$ action is available in our example).

In game $\mathcal{C}_2$, the order on actions (also written $<$) from (ℓ_0, r_0) is given by:

[7] This is due to the fact that we have assumed edge $(s, s_\bullet)$ belong to player i, see the construction of game $\mathcal{C}_i$.

$$\begin{array}{ccccccccccc} ((r_1,b),\bullet) & < & (r_1,a) & < & ((r_1,b),\circ) & < & ((r_2,b),\bullet) & < & (r_2,c) & < & ((r_2,b),\circ) \\ \wr & & \wr & & \wr & & \wr & & \wr & & \\ (\ell_2,r_0) & & (\ell_1,r_0) & & (\ell_2,r_0) & & (\ell_2,r_0) & & (\ell_3,r_0) & & \end{array}$$

We will take advantage of this order on actions to build turn-based games that will in some sense be equivalent with the previous concurrent (deterministic) games. The idea will be to take the smallest action(s) in the order, and ask the corresponding player whether or not she wants to play that action; if yes, we proceed with this action in the game, otherwise we do the same with the second action in the order until one of the players plays her action; The meaning in the context of timed games is actually also the following: we see that if the two players want to play in the same region, then in game $\mathcal{C}_i$ the advantage of player i is that we first ask her whether she wishes to play her action (role of action labelled with $\bullet$), then if not, the other player will be asked to decide whether she wants to play her own action, and finally, if not, we ask a last time player i whether she wants to play her action (now she has the additional knowledge that the other player didn't choose her own action).

Formally we define the turn-based game $\mathcal{T}_i$ as follows: $\mathcal{T}_i = \langle \mathsf{States}_i, \mathsf{Edg}_i, \mathsf{Act}' \cup \{\mathsf{del}\}, \mathsf{Mov}'_i, \mathsf{Tab}'_i, (\preccurlyeq'^i_1, \preccurlyeq'^i_2)\rangle$ where:

- $\mathsf{States}_i = \{(s,[m]_s) \mid s \in \mathsf{States} \text{ and } m \in (\mathsf{Mov}_i(s,1) \cup \mathsf{Mov}_i(s,2)) \setminus \{\bot\}\}$;
- The set Edg_i is defined as follows:

$$\begin{aligned} \mathsf{Edg}_i = \quad & \{((s,[m]_s),(s,[m']_s)) \mid m' \neq \bot \text{ is next after } m \text{ w.r.t. } <_s\} \\ & \cup \{((s,[m]_s),(s',[m']_{s'})) \mid \{(s,s')\} = \mathsf{Tab}_i(s,(m,m'')) \\ & \qquad \text{for every } m <_s m'' \text{ and } m' \text{ is minimal w.r.t. } <_{s'} \text{ from } s'\}; \end{aligned}$$

- The set of available actions is defined as follows:
  - if $[m]_s$ belongs to player j, then we use the new action del (for *delay*):

$$\mathsf{Mov}'_i((s,[m]_s),j) = \begin{cases} \mathsf{Mov}_i(s,j) \cap [m]_s & \text{if } m \text{ is maximal w.r.t.} \\ & <_s \text{ in } \mathsf{Mov}_i(s,j) \\ (\mathsf{Mov}_i(s,j) \cap [m]_s) \cup \{\mathsf{del}\} & \text{otherwise} \end{cases}$$

  - if $[m]_s$ belongs to player $3-j$, then $\mathsf{Mov}'_i((s,[m]_s),j) = \{\bot\}$.
- The transition table is defined as follows:
  - if $[m]_s$ belongs to player 1:

$$\begin{cases} \mathsf{Tab}'_i((s,[m]_s),(m,\bot)) = \{((s,[m]_s),(s',[m']_s)) \mid \{(s,s')\} = \mathsf{Tab}_i(s,(m,m'')) \\ \qquad\qquad \text{for every } m <_s m'' \text{ and } m' \text{ is minimal w.r.t. } <_{s'} \text{ from } s'\} \\ \mathsf{Tab}'_i((s,[m]_s),(\mathsf{del},\bot)) = \{((s,[m]_s),(s,[m']_s)) \mid m' \text{ is next after } m \text{ w.r.t. } <_s\} \end{cases}$$

  - the second case ($[m]_s$ belongs to player 2) is similar, just swap m or del with $\bot$.
- In order to define the preference relations we first define a projection from plays in the turn-based game $\mathcal{T}_i$ onto plays in the concurrent game $\mathcal{C}_i$. Pick a run ν in $\mathcal{T}_i$, and define its projection $\psi_i(\nu)$ in $\mathcal{C}_i$ as follows: if

$$\nu = (s_1,\mathfrak{m}_1^1)(s_1,\mathfrak{m}_1^2)...(s_1,\mathfrak{m}_1^{k_1})(s_2,\mathfrak{m}_2^1)...(s_2,\mathfrak{m}_2^{k_2})...(s_p,\mathfrak{m}_p^1)...(s_p,\mathfrak{m}_p^{k_p})...$$

with $\mathfrak{m}_i^1$ minimal w.r.t. $<_{s_i}$ for every $1 \leq i$, then $\psi(\nu) = s_1 s_2 \dots s_p \dots$. The preference relations are then defined according to this projection:

$$\nu \preccurlyeq'^{i}_{j} \nu' \Leftrightarrow \psi_i(\nu) \preccurlyeq^{i}_{j} \psi_i(\nu')$$

Note that the game $\mathcal{T}_i$ is turn-based[8] and that a state $(s, [m]_s)$ *belongs* to player j such that $m \in \mathsf{Mov}_i(s, j)$ (as already mentioned this is independent of the choice of m in $[m]_s$). The structure of the turn-based games $\mathcal{T}_1$ and $\mathcal{T}_2$ are now slightly different from that of the previous concurrent deterministic games $\mathcal{C}_1$ and $\mathcal{C}_2$.

Proposition 11. *Let $\mathcal{C}_i$ (with $i \in \{1, 2\}$) be the previous deterministic concurrent game, and let $\mathcal{T}_i$ be the associated turn-based game. There is a Nash equilibrium in $\mathcal{C}_i$ from s with best play $\psi_i(\nu)$ iff there is a Nash equilibrium in $\mathcal{T}_i$ from $(s, [m]_s)$ with best play ν, where m is a minimal action w.r.t. $<_s$. Furthermore this equivalence is constructive.*

Example 1 (Cont'd). We build on our running example, and compute the corresponding games $\mathcal{T}_1$ and $\mathcal{T}_2$. They are displayed on Fig. 3. Plain states and plain edges belong

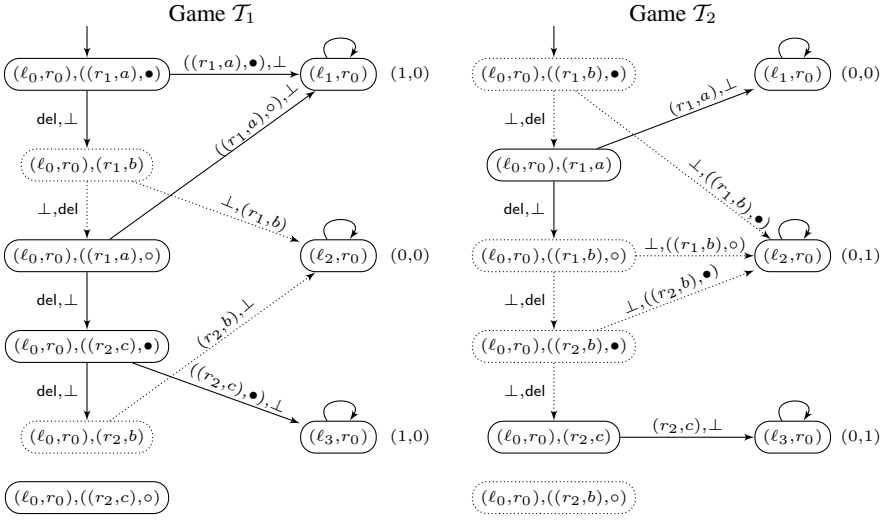


Fig. 3. Final turn-based games from our original timed game

to player 1 whereas dotted states and dotted edges belong to player 2. We do recognize here the various Nash equilibria that we described in the concurrent deterministic games, and only one is "common" to both games, namely the one leading to (ℓ_3, r_0).

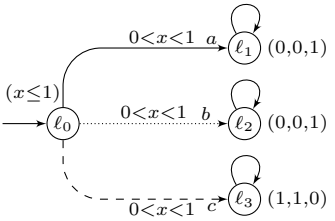
3.5 Summary of the Construction

The following theorem summarizes our construction:

Theorem 12. *Let $\mathcal{G}$ be a timed game with region-uniform preference relations. Assume $\mathcal{T}_1$ and $\mathcal{T}_2$ are the two turn-based (deterministic) games constructed in this section. Then, there is a pseudo Nash equilibrium in $\mathcal{G}$ from $(\ell_0, \mathbf{0})$ with best play ρ iff there are two Nash equilibria in $\mathcal{T}_1$ and $\mathcal{T}_2$ from $((\ell_0, \mathbf{0}), [m]_{(\ell_0,\mathbf{0})})$ with best plays ν_1 and ν_2*

[8] By construction, in any state $(s, [m]_s)$, one of $\mathsf{Mov}'_i((s, [m]_s), j)$ with $j \in \{1, 2\}$ equals $\{\bot\}$.

respectively, where m is a minimal action w.r.t. $<_{(\ell_0,\mathbf{0})}$, such that $\psi_1(\nu_1) = \psi_2(\nu_2) = \pi_{\mathfrak{R}}(\rho)$. Furthermore this equivalence is constructive.



Remark 1. The three-player game on the right has several Nash equilibria, for instance player 1 (plain arrows) chooses her transition at time 0.6, player 2 (dotted arrows) chooses her transition at time 0.7, and player 3 (dashed arrows) chooses her transition at time 0.8. If we build the region abstraction, each player will have a single possible move (play her transition in the region $0 < x < 1$), and the game will proceed by selecting non-deterministically one of them. There would be several ways to extend the method developed in this paper to three players: have a copy of the game for each player, assuming she plays against a coalition of the other players, or have a copy of the game for each priority order given to the players. It is not hard to be convinced that none of these choices will be correct on this example.

4 Decidability Results

4.1 Some General Decidability Results

We first need a representation for the preference relations (which must be region-uniform) of both players. Let $\mathcal{G} = \langle \mathsf{Loc}, \mathsf{Cl}, \mathsf{Inv}, \mathsf{Trans}, \mathsf{Owner}, (\preccurlyeq_1, \preccurlyeq_2)\rangle$ be a two-player timed game. We assume the preference relation for player i is given by a (possibly infinite) sequence of linear-time objectives $(\Omega_j^i)_{j\geq 1}$ where it is better for a run to satisfy Ω_j^i than Ω_k^i as soon as $k > j$ (w.l.o.g. we assume that Ω_{j+1}^i implies $\neg\Omega_l^i$ for all $l \leq j$). In other terms, the aim of player i is to minimize the index j for which the play belongs to Ω_j^i. ω-regular or LTL-definable objectives, and also more *quantitative* objectives (for instance, given a distinguished goal state $\mathsf{Goal}_i \in \mathsf{Loc}$ for player i, by defining Ω_j^i to be the set of traces visiting Goal_i in less than j steps.

We first need to (be able to) transfer objectives (and preference relations) to the two turn-based games $\mathcal{T}_1$ and $\mathcal{T}_2$: a linear-time objective Ω in $\mathcal{G}$ is said to be *transferable to game* $\mathcal{T}_i$ whenever we can construct an objective $\widehat{\Omega}$ such that for every run ν in $\mathcal{T}_i$, $\nu \models \widehat{\Omega}$ iff for all ρ with $\pi_{\mathfrak{R}}(\rho) = \psi_i(\nu)$, $\rho \models \Omega$. It is said *transferable* whenever it is transferable to both $\mathcal{T}_1$ and $\mathcal{T}_2$. For example, notice that (sequences of) stutter-free region-uniform objectives are transferable.

Nash equilibria in game $\mathcal{T}_i$ will be rather easy to characterize since player $3-i$ will never be enclined to deviate from her strategy (all runs are equivalent for her preference relation). We assume all objectives Ω_j^i are transferable, and we write $W_i^{3-i}(j)$ for the set of winning states in game $\mathcal{T}_i$ for player $3-i$ with the objective $\bigwedge_{1\leq k<j}\big(\neg\widehat{\Omega}_k^i\big)$. Those sets are computable for many classes of objectives. Then:

Theorem 13. *Let $\mathcal{G}$ be a timed game with preference relations given as transferable, region-uniform, prefix-independent sequences $(\Omega_j^i)_j$ of objectives. There is a pseudo Nash equilibrium in $\mathcal{G}$ with payoff (Ω_1^j, Ω_2^k) iff there are two runs ν_1 in $\mathcal{T}_1$ and ν_2 in $\mathcal{T}_2$ s.t.*[9] *(i) $\nu_1 \models \big(\mathbf{G}\, W_1^2(j)\big) \wedge \widehat{\Omega}_j^1$, (ii) $\nu_2 \models \big(\mathbf{G}\, W_2^1(k)\big) \wedge \widehat{\Omega}_k^2$, and (iii) $\psi_1(\nu_1) = \psi_2(\nu_2)$.*

[9] $\mathbf{G}$ is the LTL modality for "*always*".

Notice that this allows to handle ω-regular (and LTL-definable) objectives by considering the product of the game with a suitable (deterministic) automaton.

The sequence of states $(W_i^{3-i}(j))_{j\geq 1}$ in game $\mathcal{T}_i$ is non-increasing and hence stationary (because $\mathcal{T}_i$ is finite-state). Hence there exist indices $h_0 = 1 < h_1 < h_2 < \cdots < h_l$ such that the function $j \mapsto W_i^{3-i}(j)$ is constant on all intervals $[h_p, h_{p+1})$ and on $[h_l, +\infty)$. Those indices can be computed together with the corresponding sets of winning states. Then the only possible equilibria in $\mathcal{T}_i$ are those such that there is a run satisfying $\widehat{\Omega}_j^i$ that stays furthermore within the set $W_i^{3-i}(h_p)$ if $h_p \leq j < h_{p+1}$, or within $W_i^{3-i}(h_l)$ if $j \geq h_l$. This can be done for instance if each player is given a goal state Goal_i, and Ω_j^i is "reach Goal_i in j steps". In that case, $W_i^{3-i}(h_l)$ is the set of states from which player $3-i$ can avoid Goal_i. Hence we can compute Nash equilibria in two-player timed games where each player tries to minimize the number of steps to the goal state. This allows to recover part of the results of [8] for two-player games.

4.2 Zero-Sum Games

Our chain of transformations also yields a new point-of-view on classical two-player timed games with zero-sum objectives. In that case the preference relation of player 1 is characterized by the sequence $(\Omega, \neg\Omega)$ whereas that of player 2 is characterized by the sequence $(\neg\Omega, \Omega)$. In that case we say that the objective of player 1 is Ω.

Theorem 14. *Let $\mathcal{G}$ be a zero-sum timed game where player 1's objective is Ω, and is assumed to be transferable. Then player 1 has a winning strategy in $\mathcal{G}$ from $(\ell, \mathbf{0})$ iff player 1 has a winning strategy in game $\mathcal{T}_2$ from $(\ell, [\mathbf{0}])$ for the objective $\widehat{\Omega}$.*

5 Conclusion

We have proposed a series of transformations of two-player timed games into two turn-based finite games. These transformations reduce the computation of Nash equilibria in timed games for a large class of objectives (the so-called region-uniform objectives) to the computation of "twin" equilibria for related objectives in the two turn-based finite games. We give an example on how this can be used to compute Nash equilibria in timed games. In turn our transformations give a nice and new point-of-view on zero-sum timed games, which can then be interpreted as a turn-based finite game.

Our method does not extend to n players. In [6], we have developed a completely new approach that allows to compute Nash equilibria in timed games with an arbitrary number of players but only for reachability objectives. We plan to continue working on the computation of Nash equilibria in timed games with an arbitrary number of players.

Another interesting research direction is the computation of other kinds of equilibria in timed games (secure equilibria, subgame-perfect equilibria, *etc*). We believe that the transformations that we have made in this paper are correct also for these other notions, and that we can for instance reduce the computation of subgame-perfect equilibria to the computation of subgame-perfect equilibria in the two turn-based finite games. A major difference is that Theorem 13 has to be refined. Tree automata could be the adequate tool for this problem [17].

References

1. Alur, R., Dill, D.L.: A theory of timed automata. TCS 126(2), 183–235 (1994)
2. Alur, R., Henzinger, T.A., Kupferman, O.: Alternating-time temporal logic. J. ACM 49, 672–713 (2002)
3. Asarin, E., Maler, O., Pnueli, A., Sifakis, J.: Controller synthesis for timed automata. In: SSSC'98, pp. 469–474. Elsevier, Amsterdam (1998)
4. Behrmann, G., Cougnard, A., David, A., Fleury, E., Larsen, K.G., Lime, D.: UPPAAL-Tiga: Time for playing games! In: Damm, W., Hermanns, H. (eds.) CAV 2007. LNCS, vol. 4590, pp. 121–125. Springer, Heidelberg (2007)
5. Bouyer, P., Brenguier, R., Markey, N.: Computing equilibria in two-player timed games *via* turn-based finite games. Research Report LSV-10-11, Lab. Spécification & Vérification, ENS Cachan, France (2010)
6. Bouyer, P., Brenguier, R., Markey, N.: Nash equilibria for reachability objectives in multi-player timed games. In: CONCUR'10. LNCS. Springer, Heidelberg (2010)
7. Brenguier, R.: Calcul des équilibres de Nash dans les jeux temporisés. Master's thesis, ENS Cachan (2009)
8. Brihaye, T., Bruyère, V., De Pril, J.: Equilibria in quantitative reachability games. In: Ablayev, F., Mayr, E.W. (eds.) Computer Science – Theory and Applications. LNCS, vol. 6072, pp. 72–83. Springer, Heidelberg (2010)
9. Chatterjee, K., Henzinger, T.A., Jurdziński, M.: Games with secure equilibria. In: LICS'06, pp. 160–169. IEEE Comp. Soc. Press, Los Alamitos (2006)
10. Chatterjee, K., Majumdar, R., Jurdziński, M.: On Nash equilibria in stochastic games. In: Marcinkowski, J., Tarlecki, A. (eds.) CSL 2004. LNCS, vol. 3210, pp. 26–40. Springer, Heidelberg (2004)
11. de Alfaro, L., Faella, M., Henzinger, T.A., Majumdar, R., Stoelinga, M.: The element of surprise in timed games. In: Amadio, R.M., Lugiez, D. (eds.) CONCUR 2003. LNCS, vol. 2761, pp. 142–156. Springer, Heidelberg (2003)
12. Félegyházi, M., Hubaux, J.-P., Buttyán, L.: Nash equilibria of packet forwarding strategies in wireless ad hoc networks. IEEE Trans. Mobile Computing 5(5), 463–476 (2006)
13. Henzinger, T.A.: Games in system design and verification. In: TARK'05, pp. 1–4. Nat. Univ., Singapore (2005)
14. Nash, J.F.: Equilibrium points in n-person games. Proc. Nat. Academy of Sciences of the USA 36(1), 48–49 (1950)
15. Paul, S., Simon, S.: Nash equilibrium in generalised Muller games. In: FSTTCS'09. LIPIcs, vol. 4, pp. 335–346. Leibniz-Zentrum für Informatik (2009)
16. Thomas, W.: Infinite games and verification. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, pp. 58–64. Springer, Heidelberg (2002) (invited tutorial)
17. Ummels, M.: Rational behaviour and strategy construction in infinite multiplayer games. In: Arun-Kumar, S., Garg, N. (eds.) FSTTCS 2006. LNCS, vol. 4337, pp. 212–223. Springer, Heidelberg (2006)
18. Ummels, M.: The complexity of Nash equilibria in infinite multiplayer games. In: Amadio, R.M. (ed.) FoSSaCS 2008. LNCS, vol. 4962, pp. 20–34. Springer, Heidelberg (2008)
19. Ummels, M., Wojtczak, D.: The complexity of Nash equilibria in simple stochastic multiplayer games. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009. LNCS, vol. 5556, pp. 297–308. Springer, Heidelberg (2009)
20. Ummels, M., Wojtczak, D.: Decision problems for Nash equilibria in stochastic games. In: Grädel, E., Kahle, R. (eds.) CSL 2009. LNCS, vol. 5771, pp. 515–529. Springer, Heidelberg (2009)

Natural Domain SMT: A Preliminary Assessment

Scott Cotton

Verimag, Centre Équation
2, Ave. de Vignate
38610 Gières, France

Abstract. SMT solvers have traditionally been based on the DPLL(T) algorithm, where the driving force behind the procedure is a DPLL search over truth valuations. This traditional framework allows for a degree of modularity in the treatment of theory solvers. Over time, theory solvers have become more and more closely integrated into the DPLL process, and consequently less and less modular. In this paper, we present a DPLL-like algorithm for SMT solving in which the search takes place over the natural domain of the variables in the problem. As a case study, we analyze its application to continuous domain linear arithmetic, present implementation techniques and some experimentation with difference logic. Results indicate the method can sometimes outperform leading SMT solvers but that the method is not yet robust.

1 Introduction

SMT solvers have traditionally been based on the DPLL(T) [GHN+04] algorithm, where the driving force behind the procedure is a DPLL search over truth valuations. This traditional framework allows for a degree of modularity in the treatment of theory solvers. Over time, theory solvers have become more and more closely integrated into the DPLL process with such techniques as theory propagation, theory driven case splitting, and theory learning. These techniques generally correspond to the introduction of new proof rules, and hence non-determinism, in the underlying proof system. Consequently, SMT solvers which incorporate these techniques face the problem of deciding when to apply them. In general, increased non-determinism in the solving process presents a corresponding need for more effective heuristics to guide the solving process. Unfortunately, such heuristics are not well understood, and in fact are often static, theory specific, and perhaps over-tuned to competition benchmarks.

In this paper, we present a DPLL-like algorithm for SMT solving which unifies theory solving and DPLL search into a single process, directly searching for a model over the space of variable valuations, rather than searching piecewise for theory models of (partial) propositional models. The goal of this effort is to find a natural domain formalism which supports dynamic and principled heuristics. In particular, by searching over variable valuations we are able to make use of VSID [MMZ+01] variable ordering heuristics found in modern SAT solvers but

K. Chatterjee and T.A. Henzinger (Eds.): FORMATS 2010, LNCS 6246, pp. 77–91, 2010.

applied to arbitrary variables; and by defining a notion of progress over the search space, we are able to determine which learned theory literals and clauses are relevant to progress, thus giving a principled and dynamic heuristic for the creation and maintenance of a set of theory literals.

However, our algorithm also introduces problematics not found in the traditional DPLL(T) based techniques. In particular, theories with unbounded derivations introduce strong restrictions on the algorithm if completeness is desired. Thus our framework is by no means a panacea and in fact only outperforms traditional methods on a minority of the benchmarks we evaluated. Nonetheless, we find the abstract formulation, analysis, and implementation-level solutions interesting and potentially useful to the design of future solvers.

1.1 Related Work

On the abstract level, our work is most closely related to GDPLL [MKS09], providing a general satisfiability solving framework. Unlike [MKS09], our generic framework is explicitly based on variable valuations and clause learning. As a result, our framework guarantees that progress is made during the search – new variable valuations are always explored because the search space is properly restricted over time. Using this notion of progress, we establish how learned clauses can be safely forgotten, a topic which was not addressed in [MKS09]. Finally, we define a notion of a proof graph with some properties similar to propositional resolution proof graphs, and show how this relates to completeness of the method for unbounded proof systems.

On a more concrete level, we present an application to linear real arithmetic, and some experimentation on various difference logic problems. Previously in [KTV09], a (conjunctive) theory solver for linear arithmetic has been presented which uses numeric variable valuations to detect and resolve conflicts in a manner similar to clause learning in SAT solving. In [MKS09], the GDPLL-QFLRA algorithm is presented to solve CNF formulas whose literals are linear constraints, and experimentation is presented on difference logic problems. Both works make use of a fixed variable ordering. In this work, we show some conditions under which dynamic variable orderings are complete and show that using dynamic variable orderings in an incomplete way can be much faster in practice.

More generally, most SMT solving makes use of the DPLL(T) framework, and some extensions to the basic DPLL(T) proof system allow for arbitrary generation of theory literals [BSST09]. In its most general form, this extension may be used to simulate our algorithm. Various restricted instantiations [BDdM08, dMB08, WGG06] of this extension exist, but none views the solving process as a search on variable valuations. On the other hand, the constraint programming community has a long history of work on search in non-Boolean domains, including the use of interval constraint propagation [BG06] for *bounded* continuous domains and some techniques from constraint logic programming for unbounded domains [MSW06]. To the author's best knowledge, the work presented here is distinct from this body of work in that we apply SAT based

heuristics and apply conflict driven learning on top of a local consistency condition to unbounded domains.

1.2 Organization

The rest of this paper is organized as follows. We present our abstract search algorithm in Section 2. Section 3 presents generic implementation problems and solutions. Section 4 presents an application to linear real arithmetic. Experiments on the sub-theory of difference logic problems are presented in 5. Section 6 concludes.

2 Univariate Consistent Search

The framework we present is, at its base, a depth first search over the natural domain of variables. By natural domain, we have in mind sets such as the integers $\mathbb{Z}$ or rationals $\mathbb{Q}$, or perhaps explicitly represented finite domains, though we place no *a priori* restrictions on the variable domains. By depth first search, we mean a backtracking search takes place by assigning variables to values, one at a time until either a satisfying assignment is found or it is clear that the formula under the assigned variable valuations is inconsistent. In this latter case, the search backtracks, learning a clause which excludes some part of the variable assignment.

In this respect, the framework we present is very much like a conflict-driven clause learning propositional SAT solver. However, the coordination of search, consistency checking, and learning becomes much more crucial when we consider a multi-valued search over arbitrary variable domains. In the following, the main concept behind this coordination is the notion of *univariate consistency*, which is a simple local consistency condition. Consider a quantifier free CNF formula $\phi \triangleq \bigwedge C$ where each $c \in C$ is a clause. Now given a variable x, we denote by

$$\phi|_x \triangleq \bigwedge \{c \in C \mid vars(c) = \{x\}\}$$

That is, $\phi|_x$ is the set of all x-univariate clauses. We say ϕ is *univariate consistent*, or simply u-consistent, if for every variable $x \in vars(\phi)$ it is the case that $\exists x.\phi|_x$. Similarly, given an assignment α, we denote by $\phi|_{\alpha,x}$ the set of all clauses $c \in \phi$ such that $vars(c[\alpha]) = \{x\}$.

In a depth first search framework, u-consistency can be used to determine what partial assignments can be extended, whether or not to backtrack, and what to learn upon backtracking. Below we present each of these functionalities as a procedure. Later, we will analyze different ways in which these procedures can be composed.

`select`(ϕ, α). This procedure takes a formula ϕ and a partial assignment α as arguments and returns a pair (x, a) where a is in the domain of x, $x \in vars(\phi[\alpha])$ and where the assignment $x \mapsto a$ is feasible for all x-univariate clauses in $\phi[\alpha]$. This is an analog of selecting a decision variable and phase in a DPLL sat solver.

`isUC`(ϕ) This procedures tests the u-consistency of a conjunction of clauses. It returns true if the formula ϕ is u-consistent and false otherwise.

`resolve`(ϕ, α, x) This procedure takes a formula ϕ, a partial assignment α and a variable x such that $(\phi[\alpha])|_x$ is unsatisfiable, and returns a clause w such that
1. $\bigwedge\{c \mid c \in \phi \wedge vars(c[\alpha]) = \{x\}\} \models w$
2. $w[\alpha]$ contains no variables and evaluates to false.

Resolve is used for learning new clauses when u-consistency is violated for some variable x and assignment α. It finds some consequence w of the x-univariate clauses under α which is false under α.

Suppose that we have implementations of the three above-mentioned procedures. Using these, we formulate a simple recursive algorithm for deciding CNF formulas, which is written in pseudo-code in Figure 1. We will use this algorithm both as a formal point of reference for analysis, and subsequently as a basis on which various implementations may be built.

```
    UC-Search(φ, α)
1     if isUC(φ[α]) then
2       let (x, a) = select(φ, α)
3       if vars(φ[α]) = {x} then
4         return (1, α ∪ {x ↦ a})
5       let (r, w) = UC-Search(φ, α ∪ {x ↦ a})
6       if r = 1 or x ∉ vars(w) then
7         return (r, w)
8       else
9         return UC-Search(φ ∧ w, α)
10    else
11      let x be s.t. φ[α]|_x is unsat
12      let w = resolve(φ, α, x)
13      return (0, w)
```

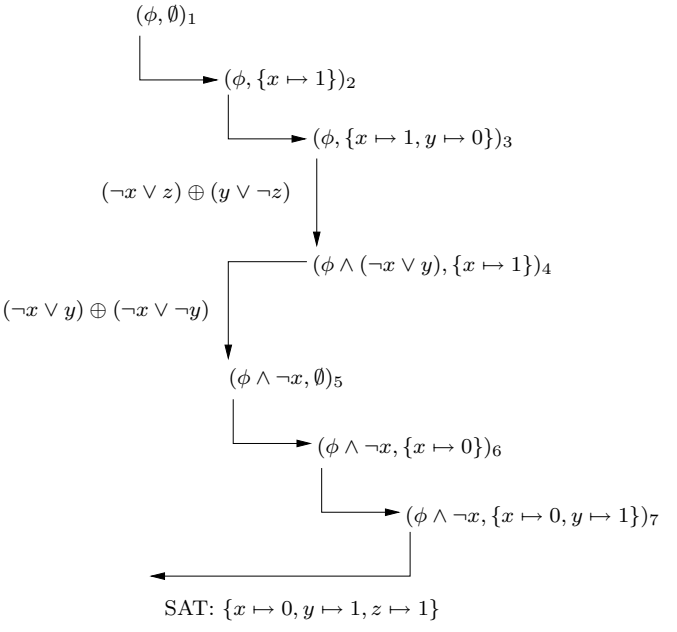
Fig. 1. `UC-Search` takes two arguments, ϕ, α, where ϕ is a formula and α is a partial assignment to the variables in ϕ. It returns either a pair $(1, \alpha)$ where α is a satisfying assignment for ϕ or a pair $(0, w)$ where w is a false clause *i.e* a clause whose every literal has no variables and such that each literal evaluates to false. The algorithm implements a depth first search which either extends the assignment (at line 2) or learns and records clauses (lines 12 and 9). Backtracking is expressed implicitly as the return of a function call, and can have the effect of retracting a partial assignment (line 7) or forgetting a learned clause (line 9).

Perhaps the most interesting aspect of the `UC-Search` algorithm is that memory of learned clauses is managed by the call stack. In particular, line 9 of the algorithm extends the call stack without extending the assignment but rather by extending the set of learned clauses. The following example shows how the call stack is also used to forget learned clauses.

Example 1 (`UC-Search` run). Consider the propositional formula ϕ

$$\phi \triangleq (x \vee y) \wedge (\neg y \vee z) \wedge (\neg x \vee z) \wedge (\neg z \vee y) \wedge (\neg x \vee \neg y)$$

A diagram of a possible trace of the algorithm deciding ϕ follows.



The nodes in the graph are annotated with subscripted pairs $(\phi, \alpha)_n$ representing a sequence of calls to the procedure `UC-Search`. The recursion depth in which each call occurs is represented by horizontal position. Each call opens a new scope which in turn triggers at most 2 calls directly contained in that scope. The first such call occurs at line 5 and the second at line 9 in the pseudo-code listing. No calls are triggered when the formula $\phi[\alpha]$ is not univariate consistent nor when the assignment satisfies the formula. Outgoing edges of calls which fail the consistency check are labeled with resolution operations $c \oplus d$. Observe that the clause $(\neg x \vee y)$ derived between calls 3 and 4 is subsequently forgotten because it falls out of the scope of call 2.

2.1 Some Properties of `UC-Search`

Theorem 1. `UC-Search` *Soundness*

Proof. In every invocation of `UC-Search`, the argument ϕ is built up from the original formula together with learned clauses recorded at line 9. Such recorded clauses are generated by `resolve`, and hence are a consequence of the original formula. Hence in every invocation of `UC-Search` the argument ϕ is a consequence of the original formula.

Now a top level call to `UC-Search` returns either a value $(1, \alpha)$, or a value $(0, w)$. In the first case, the procedure must have returned $(1, \alpha \cup \{x \mapsto a\})$ at line 5 when $\phi[\alpha]$ is univariate consistent and $x \mapsto a$ is a satisfying assignment for $\phi[\alpha]|_x$ and $\{x\} = vars(\phi[\alpha])$. The assignment $\alpha \cup \{x \mapsto a\}$ thus satisfies ϕ, and hence the original formula. In the second, case w is generated by `resolve` and

must be variable-free. Additionally, `resolve` guarantees that w evaluates to false and is a consequence of ϕ, which again by the reasoning above is a consequence of the original formula.

The following theorem shows that the `UC-Search` procedure is guaranteed to make progress in the sense that the search space becomes more and more constrained over time.

Theorem 2. `UC-Search` *Progress*
Let $U(\phi, \alpha)$ *denote the set of all univariate-feasible* 1*-extensions of* α

$$U(\phi, \alpha) \triangleq \{(x, a) \mid x \in vars(\phi[\alpha]), (\phi[\alpha]|_x)[x \mapsto a] \text{ is true}\}$$

Let $(\phi_1, \alpha_1)(\phi_2, \alpha_2) \ldots (\phi_k, \alpha_k)$ *denote a sequence of calls to* `UC-Search` *during a run of the procedure. Consider a cycle with two calls* i, j*, such that* $i > j$ *and* $\alpha_i = \alpha_j$*. Let* $\alpha \triangleq \alpha_i$*. Then* $U(\phi_i, \alpha) \subset U(\phi_j, \alpha)$*.*

Progress is an analog of the termination argument for a DPLL solver which learns asserting clauses [ZM03]. Of course, one cannot guarantee termination without specifying something more about the theory or the resolution procedure. However, progress is more subtle in `UC-Search` because of the fact that when a variable is univariate constrained it may or may not be assigned under α. This situation leads to the possibility of the procedure cycling with respect to a partial assignment. Progress simply says that whenever this happens, the procedure is in a state in which the search space is properly constrained with respect to the variable order in which the variables are assigned. Note that progress takes place even though `UC-Search` forgets clauses from deeper in the call stack. Thus progress allows a solver to safely forget clauses, but still there is no limit on the minimum number of clauses necessary to guarantee progress because many learned univariate clauses can be recorded under a given assignment before the procedure backtracks over that assignment. Also note that as stated, progress requires that upon learning a clause w, `UC-Search` backtracks to the *maximal* assignment under which w is univariate. A similar notion of progress holds if `UC-Search` backtracks to the minimal assignment, which we omit for simplicity.

The notion of progress is extremely weak by comparison to termination of DPLL by asserting clauses; which brings us directly to the question of exactly when `UC-Search` terminates. Having established progress, it is not hard to see that if the closure of a finite set of clauses under `resolve()` is finite, then the procedure terminates. However, for arbitrary variable domains, `resolve()` is not necessarily finite. For example, from

$$(x > 1) \wedge (y > x + 1) \wedge (x > y + 1)$$

`resolve` may produce $y > 2$ and subsequently $x > 3$, $y > 4$, etc.

To better address this issue, we introduce the notion of the proof graph traced by `UC-Search` in terms of the input-output relation of the procedure `resolve`. Consider a call

$$v = \texttt{resolve}(\phi, \alpha, x)$$

v is a consequence of some subset W of the clauses in ϕ where each $w \in W$ is x-univariate under α. The proof graph then consists of one edge (w, v) for each $w \in W$. For example, consider a call to $\texttt{resolve}(c \wedge d \wedge e \wedge f, \alpha, x)$ which results in the conclusion g. Then the proof graph representing this step may consist of the edges $(d, g), (e, g), (f, g)$, provided `resolve()` found that $d, e, f \models g$. We label each edge in this graph with x, referred to as the *pivot variable*. As in propositional resolution, the proof graph is *regular* if for every path in the graph, the corresponding sequence of pivot variables contains each variable at most once. The proof graph is *tree-like*, if each instance of a derived clause[1] can be the antecedent of at most one other derived clause.

Consider the topological properties of a proof graph generated by calls to `resolve()` in `UC-Search`. The graph is not necessarily tree-like nor necessarily regular, because the variables are chosen in any order. One may restrict the form of resolution using the following notion.

Definition 1 (Exhaustively Asserting). *We say that* `UC-Search` *is* exhaustively asserting *if any variable chosen after a conflict is the variable constrained by the last learned clause.*

To illustrate this property, consider a backtrack sequence. Every time there is an inconsistency, `UC-Search` returns a clause w derived by `resolve`. Either w has no variables, and the problem is unsatisfiable, or the clause w has a variable x which is maximal in the search, and w excludes an assignment $\alpha \cup \{x \mapsto a\}$. In this later case, $(\phi \wedge w)[\alpha]$ may or may not be univariate consistent. If it is not univariate consistent, the backtrack sequence is not maximal and `resolve` is called again. Otherwise, $(\phi \wedge w)[\alpha]$ is univariate consistent and a free variable is selected. In the case that `UC-Search` always selects x within such a context, we say it is exhaustively asserting.

Theorem 3 (Restricted Resolution). *If* `UC-Search` *is exhaustively asserting then the proof graph traced by* `UC-Search` *is tree-like and regular.*

Restricted resolution, in turn, allows us to relax the requirements on `resolve()` necessary for termination. Consider the property of finite width:

Definition 2 (Finite Width). *Let $\mathcal{L}$ be a language of literals (atomic predicates or their negations). Given a finite set of clauses ϕ whose literals fall in $\mathcal{L}$ and a distinguished variable $x \in vars(\phi)$, let*

$$r(x) \triangleq \{w \mid \exists \alpha \,.\, \phi[\alpha]|_x \text{ is unsat, and } w = \texttt{resolve}(\phi, \alpha, x)\}$$

denote the set of all derivable clauses under any variable valuation around x. We say that `resolve` *has* finite width *for $\mathcal{L}$, if for any finite set of clauses ϕ over $\mathcal{L}$, $r(x)$ is finite and contains only literals in $\mathcal{L}$, for all $x \in vars(\phi)$.*

Theorem 4. *Termination Sufficiency*
`UC-Search` *terminates for a CNF formula ϕ with literal language $\mathcal{L}$ if*

[1] Derived clauses can appear multiple times in the graph. An instance of a derived clause corresponds to a particular call to `resolve`.

1. `resolve` *has finite width for* $\mathcal{L}$*; and*
2. `UC-Search` *is exhaustively asserting.*

Proof. Having established progress (Theorem 2), it will suffice to show that the set of learned clauses is finite. Since `resolve` has finite width it will suffice to show that every learned clause falls in the k-closure of `resolve` for some bound k. By Theorem 3, the proof graph is regular, and so for a formula of n variables, every learned clause falls in the n-closure of `resolve`. □

Since progress and termination implies completeness, we have a convenient criterion for establishing completeness provided an appropriate implementation of `resolve()`. Note however that the proof relies indirectly on the fact that the resolution graph is tree-like, *i.e* that `UC-Search` *forgets* clauses on backtracking. This is contrary to the intuition that the more clauses one adds to the formula, the "closer" to proving unsatisfiability. The potential problem introduced by keeping clauses if `resolve` has finite width but infinite closure is that one risks creating infinitely long chains of resolution steps.

3 Implementing `UC-Search`

In this section we consider implementing some variations of the `UC-Search` algorithm, each corresponding to different degrees of restriction on the underlying proof graph. At the same time, we are interested in an efficient implementation, which supports incremental, lazy, and backtrack-friendly data structures. We consider three kinds of restrictions on underlying proof graphs.

1. Tree-like regular proofs. This configuration corresponds to the formal algorithm in Figure 1, with the restriction that the process is exhaustively asserting. Variable orders are dynamic and learned clauses are necessarily forgotten upon backtracking.
2. Directed proofs. In this configuration, the implementation uses a fixed variable ordering and remembers learned clauses for at least as long as is necessary to satisfy the notion of progress. Learned clauses may be kept longer – those learned clauses not required for progress are cached and the cache size is limited by using a simple activity heuristic as found in many SAT solvers. Modulo implementation details and forgetting clauses, this configuration closely corresponds to GDPLL_QFLRA [MKS09]
3. Semi-constrained proofs. In this configuration, the process is exhaustively asserting and learned clauses are cached as above, but dynamic variable orderings are used. If the proof system is unbounded, this configuration is incomplete.

Thus one main problematic for implementation is the management of learned clauses, in particular the identification of those clauses which are essential to progress (Theorem 2). Another problematic is making consistency checking incremental and backtrack-friendly. Since consistency checking only applies to univariate clauses, it is convenient to treat consistency checking on a per-variable

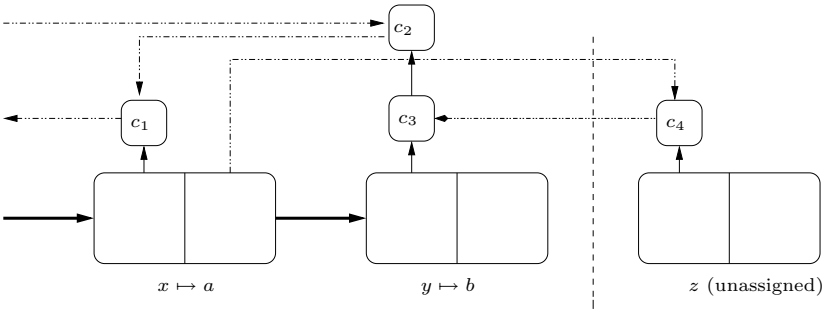


Fig. 2. Constraint Stack Indexing. A variable stack holds the partial assignments in `UC-Search` and is represented on the horizontal axis with bold links. Each variable is placed in an assignment stack and maintains privately two stacks of clauses. In the figure, clauses are labeled c_i with i an index of the time at which the procedure discovered that the clause became univariate. Each univariate clause belongs to two stacks. First, a stack representing constraints of the variable for which the clause is univariate. These stacks are placed vertically associated with each variable in the figure. Second, clauses are placed in a stack associated with the (possibly empty) variable assignment which caused them to become univariate. These stacks are linked by dashed lines in the figure. Clauses which are not essential to progress (Theorem 2) are those clauses which do not occur in any variable stack. Each variable assignment $x \mapsto a$ may cause some clauses to become univariate, and each such clause triggers an incremental consistency check for the associated variable.

basis, in such a way that the constraints placed on a given variable are treated in an incremental, backtrack-friendly fashion. Figure 2 illustrates a data structure which provides efficient support for these operations.

The constraint index also facilitates backtracking. Backtracking occurs immediately after a call to `resolve`. Backtracking is then a function of the current state of the constraint index and a newly learned clause. Backtracking occurs in per-variable units of work. As each variable x is unassigned, all its outgoing constraints are popped from x's outgoing stack and the constrained variable's incoming stack. This occurs until the newly learned clause w is univariate. Once it is univariate, a consistency check occurs. If the check succeeds, the procedure stops backtracking and passes control to the `select` function. If the check fails, `resolve` is called again, resulting in a new learned clause w' which replaces w and backtracking starts over again with w' and the new state of the constraint index.

4 Application to Linear Real Arithmetic

In this section, we present an application of `UC-Search` to real linear arithmetic. We are interesting in deciding a set of clauses whose literals are linear constraints over the reals, such as

$$
\begin{array}{l}
((2x - 7y \leq 43) \vee (2x + 7y + z > 42) \vee (x > 9)) \\
\wedge \\
((2x - 7y \leq 41) \vee (2x + 7y + z > 49) \vee (z \leq 0)) \\
\wedge \\
\ldots \\
((x < 0 \vee x > 0))
\end{array}
$$

4.1 Consistency Checking

Variable consistency checking, *i.e.* an implementation of `isUC()` plays an important role in the `UC-Search` algorithm because it occurs very frequently. In `UC-Search`, the method `isUC()` is called initially and in response to variable assignments as well as in response to clause learning. Each call to the method checks the consistency of all free variables. Of course, one may simply consider the constraints placed on each variable x independently. Over the course of a `UC-Search` run, the constraints over a variable x are asserted incrementally and may be subsequently un-asserted if the procedure backtracks or forgets a clause.

Thus consistency checking begs a per-variable incremental and backtrack friendly implementation. Section 3 describes data structures centered around identifying when clauses become univariate, free of true predicates, and non-trivial on the fly. Accordingly, we will assume that consistency checks occur upon the assertion of one non-trivial x-univariate clause at a time, for every variable x. More particularly, we consider a sequence of clause assertions $c_{x,1}c_{x,2}\ldots c_{x,k}$ where each clause $c_{x,i}$ is a non-trivial x-univariate clause. A convenient means to maintain consistency for x incrementally is to under-approximate the feasible set with lower and upper bounds l_x, u_x such that

$$l_x \wedge u_x \models \bigwedge_{1 \leq i \leq k} c_i$$

It is possible to maintain such an under-approximation with constant time updates to the values l_x, u_x upon assertion of an x-univariate clause c as follows. Initially, we let $l_x = u_x = \top$. Let $l_x(c)$ and $u_x(c)$ denote the weakest lower and upper bounds for x found in c, defaulting to $\bot$ in the case that there is no respective bound in c. After assertion of c, the next state l'_x, u'_x of the under-approximation may be computed as

$$(l'_x, u'_x) \triangleq \begin{cases} (l_x, u_x) & \text{if} & l_x \wedge u_x \models c \\ (l_x(c), u_x) & \text{else if} & u_x(c) = \bot \\ (l_x, u_x(c)) & \text{else if} & l_x(c) = \bot \\ (l_x(c), u_x) & \text{else if} & l_x(c) \models l_x \\ (l_x, u_x(c)) & \text{else if} & u_x(c) \models u_x \\ (\bot, \bot) & \text{otherwise} & \end{cases}$$

If $l'_x \wedge u'_x$ is satisfiable, the under-approximation may defer a real consistency check until a new x-univariate constraint is asserted. Otherwise, a real consistency check needs to take place with respect to the current set of clauses to find

whether there is a set of inconsistent clauses. In case the clauses are consistent, a new under-approximation needs to be computed. A real consistency check exploits the fact that an x-univariate clause defines an interval in which x is infeasible. If the union of the infeasible intervals defined by a set of x-univariate clauses covers the real line, the clauses are inconsistent.

4.2 Resolution

As an implementation of `resolve()`, resolution takes as a starting point an assignment α and a set of clauses W such that $W[\alpha]$ is x-univariate and inconsistent. Let us denote by $I_{\alpha,c}$ the infeasible interval of clause c under α. Since W is inconsistent, there are always two clauses l, u whose infeasible intervals properly overlap, *i.e.* so that the weakest lower bound on x in $l[\alpha]$ is greater than the weakest upper bound on x in $u[\alpha]$. A resolution rule for such a pair of clauses was described in [MKS09] and we briefly recall it here.

We can write l in the form $l_1 \vee l_2 \vee \ldots \vee l_m \vee A$ where each l_i bounds x from below. Similarly we can write u in the form $u_1 \vee u_2 \vee \ldots \vee u_n \vee B$ where each u_i bounds x from above. To derive a consequence r, one can compute

$$r \triangleq A \vee B \vee \bigvee \{\exists x \,.\, l_i \wedge u_j \mid 1 \leq i \leq m, 1 \leq j \leq n\}$$

One then eliminates x from each quantified disjunct in r by Fourier-Motzkin elimination, which results in a single linear constraint rather than a conjunction. It is straightforward to verify that $I_{\alpha,r} = I_{\alpha,l} \cup I_{\alpha,u}$, and thus either $r[\alpha]$ contains no variables or we can resolve another pair of clauses from $W \setminus \{l, u\} \cup \{r\}$, continuing until we derive a clause which does not contain x.

Since, up to linear equivalence, Fourier-Motzkin elimination of a single variable from a set of literals can only produce finitely many literals, and each clause contains finitely mainy literals, this implementation of resolve has finite width, provided we normalize the literal language in such a way that linearly equivalent constraints are also syntactically equivalent.

5 Experimentation with Difference Logic

We implemented a prototype solver for linear real arithmetic based on the ideas presented in this paper. The solver is restricted to CNF formulas, and represents rationals as a pair of 64 bit integers. Our prototype supports the three configurations mentioned in section 3 with dynamic variable orderings using VSID heuristic based on the number of times `resolve` was called for a given variable. The prototype also supports non-chronological backtracking, toggling value selection to upper or lower bounds or using cached values at various times, coding propositional literals in real linear arithmetic, and performing unit propagation for those literals.

As a result of the fixed precision rationals and CNF requirements, we only applied the solver to CNF difference logic problem sets in SMT-LIB [BRST08],

since difference logic in general doesn't require arbitrary precision arithmetic. Overall, we found that the treelike resolution configuration was very slow, directed resolution was very slow on scheduling problems and often slow elsewhere, and semi-constrained resolution (which is incomplete) performed the best.

All experiments were run on a Sun Java VM version 1.6.0_11 on a Debian Linux machine with dual Xeon 3.20 GHz processors and 4GB RAM. Below we detail the results.

5.1 Wide Net Experiment

Our first experiment consisted of casting a wide net over the configuration space for jobshop problems in QF_RDL/scheduling. We identified a set of configurations for experimentation; namely reasonable combinations of resolution configuration, variable selection, value selection, and backtrack depth selection (*i.e* whether or to use non-chronological backtracking). The fixed variable ordering does not make sense with varying backtracks, and so we only tested one fixed variable selection configuration. Otherwise, all combinations of semi-constrained resolution were tried, yielding a total of 19 configurations to run on 105 benchmark problems. To limit total computation time, we limited each try of a configuration on a benchmark to 15 seconds. There were a total of 1995 problem/configuration tries, of which only 576 were solved in the 15 second time limit.

The variable selection entries, all of which assume the process is exhaustively asserting, may be one of

1. *fix*. The solver uses a fixed variable ordering based on variable identities and uses directed resolution.
2. *vsid*. All variables are selected according to VSID heuristics, and all variables are incremented on every clause resolution step.
3. *vsid + ncb*. All variables are selected according to VSID heuristics and the solver does non-chronological backtracking, which decreases the backtrack depth.

The value selection strategies either used the last assigned value, a recent value derived from consistency checking, or a value based on the constraints at the time of assignment, which we refer to as a "current" value. All values were placed on the boundaries of constraints and initialized to 0. The search direction for an assignment (towards upper or lower bounds) was toggled alternatively on every assignment, never, or only immediately after asserting a learned clause.

A table of the results is appended to this paper. The most important configuration choice appears to be whether or not a fixed variable order is used. Apart from this, we observe that value selection plays a very important role and that the "last" configuration outperforms the "recent" configuration which in turn generally outperforms the "current" configuration. The bias flipping mechanism also appears to have a significant impact on performance. Generally, toggling search directions on assignments seems to work best. But in the best overall

configuration			solved	time (s)	sat	time	unsat	time
var	val	bias	toggle					
vsid+ncb	cur	all		31	78.1	18	58.0	13	20.1
vsid+ncb	cur	asrt		30	100.9	14	50.2	16	50.7
vsid+ncb	cur	no		28	68.2	15	43.1	13	25.1
vsid+ncb	last	all		33	93.7	17	63.9	16	29.8
vsid+ncb	last	asrt		38	87.9	21	59.3	17	28.6
vsid+ncb	last	no		31	69.3	16	57.5	15	11.8
vsid+ncb	rec	all		33	75.3	17	26.6	16	48.8
vsid+ncb	rec	asrt		29	52.2	14	21.9	15	30.3
vsid+ncb	rec	no		28	49.6	13	29.5	15	20.1
fix	last	no		9	23.6	2	7.0	7	16.6
vsid	cur	all		32	81.4	16	44.1	16	37.3
vsid	cur	asrt		30	91.2	15	72.9	15	18.4
vsid	cur	no		24	97.5	9	54.6	15	42.9
vsid	last	all		33	86.4	16	50.9	17	35.5
vsid	last	asrt		36	85.8	19	53.8	17	32.1
vsid	last	no		35	84.6	19	66.4	16	18.3
vsid	rec	all		32	74.3	16	44.9	16	29.3
vsid	rec	asrt		31	58.1	16	43.4	15	14.7
vsid	rec	no		33	95.0	17	71.0	16	24.0

configuration, bias toggling on assertion appears to work best. There is a large span of improvement over the space of configurations: the best configuration solves more than 4 times the problems of the worst.

5.2 Comparison to Z3

In our second experiment, we compared our best configuration to the state-of-the-art solver Z3 on the scheduling problems, diamond problems, and parity game problems from SMT-LIB. On the scheduling problems, Z3 vastly outperformed our method. On diamond problems, our method outperforms Z3. This is consistent with the observation in [MKS09] that the resolution procedure can generate exponentially shorter proofs for diamond problems. On the parity game problems, the results are largely off-diagonal with a majority in favor of Z3. The off-diagonal results indicate that the strengths of the framework presented in this paper are orthogonal to traditional approaches for this set of problems.

6 Conclusion

We have presented an abstract algorithm, `UC-Search`, for deciding a wide range of quantifier-free CNF formulas. The goal of this work is to find a decision procedure formalism which supports more principled and dynamic heuristics for SMT solving. Our procedure supports VSID style heuristics over arbitrary variables, as well as principled and dynamic heuristics for forgetting clauses based on the notion of progress. While this procedure may be applied to a wide range of theories, we observed that unbounded proof procedures introduce significant restrictions on the algorithm. Despite this fact, experiments indicate that even an incomplete version of the procedure is much faster than a leading SMT solver based on traditional techniques on a significant portion of the benchmarks we performed. While our results do not achieve improvements as a *general-purpose* solver for linear arithmetic, by applying the algorithm in this way we have discovered some fundamental costs associated with introducing dynamic variable orderings in systems with unbounded proof systems. To remedy this situation, a solver based on arbitrary regular resolution, rather than directed or tree-like resolution could be explored. Alternatively, `UC-Search` could be applied to other theories or embedded within a traditional framework, providing a basis for deriving and forgetting theory literals.

References

[BDdM08] Bjørner, N., Dutertre, B., de Moura, L.: Accelerating Lemma Learning Using Joins – DPLL(⊔). In: Int. Conf. Logic for Programming, Artif. Intell. and Reasoning, LPAR (2008)

[BG06] Benhamou, F., Granvilliers, L.: Continuous and interval constraints. In: Rossi, F., van Beek, P., Walsh, T. (eds.) Handbook of Constraint Programming, ch. 16. Elsevier, Amsterdam (2006)

[BRST08] Barrett, C., Ranise, S., Stump, A., Tinelli, C.: The Satisfiability Modulo Theories Library, SMT-LIB (2008), `http://www.SMT-LIB.org`

[BSST09] Barrett, C., Sebastiani, R., Seshia, S.A., Tinelli, C.: Satisfiability Modulo Theories February 2009. Frontiers in Artificial Intelligence and Applications, ch. 26, vol. 185, pp. 825–885. IOS Press, Amsterdam (2009)

[dMB08] de Moura, L., Bjørner, N.: Engineering DPLL(T) + Saturation. In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR 2008. LNCS (LNAI), vol. 5195, pp. 475–490. Springer, Heidelberg (2008)

[GHN+04] Ganzinger, H., Hagen, G., Nieuwenhuis, R., Oliveras, A., Tinelli, C.: DPLL(T): Fast Decision Procedures. In: Alur, R., Peled, D.A. (eds.) CAV 2004. LNCS, vol. 3114, pp. 175–188. Springer, Heidelberg (2004)

[KTV09] Korovin, K., Tsiskaridze, N., Voronkov, A.: Conflict resolution. In: Constraint Programming (2009)

[MKS09] McMillan, K.L., Kuehlmann, A., Sagiv, M.: Generalizing DPLL to Richer Logics. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 462–476. Springer, Heidelberg (2009)

[MMZ+01] Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an Efficient SAT Solver. In: DAC'01 (2001)

[MSW06] Marriott, K., Stuckey, P.J., Wallace, M.: Constraint logic programming. In: Rossi, F., van Beek, P., Walsh, T. (eds.) Handbook of Constraint Programming, ch. 12, Elsevier, Amsterdam (2006)

[WGG06] Wang, C., Gupta, A., Gannai, M.K.: Predicate Learning and Selective Theory Deduction. In: Design Automation Conference, DAC (2006)

[ZM03] Zhang, L., Malik, S.: Validating sat solvers using an independent resolution-based checker: Practical implementations and other applications. In: Design, Automation and Test in Europe Conference and Exhibition (DATE'03), p. 10880 (2003)

Robust Satisfaction of Temporal Logic over Real-Valued Signals

Alexandre Donzé and Oded Maler

CNRS-Verimag, 2 Av. de Vignate, 38610 Gières, France
donze@imag.fr

Abstract. We consider temporal logic formulae specifying constraints in continuous time and space on the behaviors of continuous and hybrid dynamical system admitting uncertain parameters. We present several variants of robustness measures that indicate how far a given trajectory stands, in space and time, from satisfying or violating a property. We present a method to compute these robustness measures as well as their sensitivity to the parameters of the system or parameters appearing in the formula. Combined with an appropriate strategy for exploring the parameter space, this technique can be used to guide simulation-based verification of complex nonlinear and hybrid systems against temporal properties. Our methodology can be used for other non-traditional applications of temporal logic such as characterizing subsets of the parameter space for which a system is guaranteed to satisfy a formula with a desired robustness degree.

1 Introduction

Analyzing the behavior of complex hybrid and nonlinear systems admitting uncertain parameters and inputs is an important ingredient in the design of *control systems* and *analog circuits* as well as in studying *biochemical reactions*. The adaptation of verification techniques to this domain proceeds along different threads, two of which are combined in the present paper. *Property checking*, also known as *monitoring* or *runtime verification*, uses temporal formulae to express desired behaviors and then checks whether individual system behaviors satisfy them, without worrying whether the set of behaviors checked *covers* the reachable state space. *Simulation-based verification* attempts to guide the generation of traces so as to demonstrate the satisfaction or violation of a property based on finitely many of them. In [8] we developed an algorithm that verifies this way safety properties of high-dimensional nonlinear systems. In this work we extend this technique to arbitrary properties expressed in a suitable temporal logic, for which we have devised a monitoring procedure [15]. The approach of [8] is strongly based on *sensitivity*, which in a nutshell, is the derivative of one continuous quantity with respect to another. To apply this concept to temporal formulae, we need to "continualize" their semantics by making it real-valued, to extend the monitoring procedure to compute this semantics efficiently for a given trace and compute the sensitivity of this semantics to parameters. This is what we do in the present paper after giving some background and motivation.

The introduction by Amir Pnueli [20] of linear-time temporal logic (LTL) into systems design as a formalism for specifying desirable and acceptable behaviors of reactive

K. Chatterjee and T.A. Henzinger (Eds.): FORMATS 2010, LNCS 6246, pp. 92–106, 2010.

systems is recognized as an important turning point in verification, putting the *ongoing sequential behavior* of a system at the center stage of the verification process. The verification framework developed over the years by Manna and Pnueli [18] consisted of three major components:

1. A *system description formalism*, specifying a behavior-generating mechanism S;
2. A *property specification formalism*, describing sets of acceptable behaviors, those that satisfy a formula φ;
3. A *verification methodology* for checking whether all behaviors of S satisfy φ.

In addition to verification where one checks whether *all* system behaviors satisfy φ, LTL is also used for lightweight verification (monitoring, runtime verification [5]) where the satisfaction of a formula by one or more *individual* behaviors is checked. For both uses, monitoring and verification alike, we point out three fundamental features of this framework, the first two related to the nature of the systems and behaviors considered while the third is related to the very notion of property and logical truth.

1. *Discrete qualitative time*: behaviors are defined as *sequences* of states or events, which are often interpreted in a purely qualitative manner, that is, without considering the metric distance between subsequent time instances;
2. *Discrete state space*: the sequences are defined over discrete and often finite domains emphasizing control rather than data;
3. *Yes/No answer*: the satisfaction of a temporal formula by a behavior is considered as membership in a set, with no quantitative degree of satisfaction.[1]

The first two features are natural for all sorts of *transition system* models expressed in various syntactic forms. Logically speaking, they imply the use of *discrete-time* and typically *propositional* temporal logic. The third feature often goes without saying. In the past two decades various attempts have been made to extend this methodology toward more refined models of systems and behaviors, going from *discrete* to *timed* and then to *hybrid* (discrete-continuous) systems. Such extensions involve departures from each of the above features.

The first departure consists in replacing the discrete time domain by the dense and metric domain $\mathbb{R}$. Behaviors of this type are generated by timed system models such as timed automata [1] and similar formalisms that can express the *durations* of discrete processes. Natural extensions of LTL to handle dense time are logics such as MTL [14] or MITL [2] which can express requirements concerning the time elapsing between events. Timed behaviors can be viewed as either *Boolean signals*, which are functions from the real time axis to $\mathbb{B}^n$, or as *timed words* consisting of instantaneous events, taken from some alphabet, separated by real time durations, see [4]. The second departure, motivated by the application of verification and monitoring techniques to continuous and hybrid systems, consists of letting predicates over the reals, such as inequalities,

[1] In a probabilistic setting one assigns probabilities to the satisfaction of a formula but this is done with respect to a *set* of behaviors, while the satisfaction by *individual* behaviors remains Boolean.

play the role of atomic propositions[2] and thus specify properties of real-valued signals. In [15] the logic STL (*signal temporal logic*) which combines the dense time modalities of MITL with such numerical predicates has been introduced, along with a monitoring tool [17,19] for deciding satisfaction of such properties by time-stamped traces produced by numerical simulators.

However, the third premise of a yes/no answer is not fully compatible with *quantitative* entities in the continuous domain where real-valued *distance* functions play an important role. As an illustration, consider the inequality $x \leq c$. Boolean satisfaction does not make a difference between $x = c+\epsilon$ and $x >> c$ as both cases are classified as violating the property. Likewise, one cannot distinguish between marginal satisfaction by $x = c - \epsilon$ and a more robust satisfaction by $x << c$. The same criticism applies to satisfaction of timing constraints: a requirement that some event occurs within t time can be violated by its occurrence at $t + \epsilon$, at $t' >> t$ as well as by its complete absence, and a yes/no answer cannot tell the difference.

These issues are extremely important in the continuous setting because such systems are subject to noise and numerical errors, not to mention the inherent approximative character of mathematical models of natural phenomena. Consequently, parameters appearing in system descriptions such as differential equations, as well as in formulae, are often a result of guessing and estimation and should *not* be regarded as representing universal constants. Thus if a property is violated in a marginal way as in $x = c + \epsilon$, satisfaction can still be achieved by slight modifications in the property, in the behavior or in the parameters of the generating system. In fact, the use of temporal logic in a *scientific* context, as in *systems biology* is methodologically different from its use in the *engineering* context of system design. In biology, it is often the case that the role of the temporal formula is to provide a *succinct abstract model* of the *observed* behavior of a complex network of chemical reactions [10,3]. The question there is not whether the system model satisfies a *given* specification but rather to find a formula, as semantically-tight as possible, compatible with the system model or with a set of observed behaviors. In such a model-search process it is important to know how close we are to a satisfactory model and which parameters should be changed (and in which direction) in order to approach it.

These observations have led several researchers to look more closely at the notion of *robust satisfiability* by continuous signals of properties expressed in STL or similar formalisms [11,22].[3] Fainekos and Pappas [11] define the notion of *robustness degree* as a real number associated with a property-behavior pair, based on, roughly speaking, the *distance* between the behavior and the (boundary of) the set of all behaviors that satisfy the property. This measure is more positive when the behavior is deeper inside the set of satisfying behaviors and more negative the further is the behavior outside that set. Hence it satisfies a natural notion of *soundness* of such a robustness measure,

[2] Such predicates are used, of course, also in more conventional applications of temporal logic to programs with *numerical variables* and also in similar logics introduced and used in biology [3,10]. All such logics are quantifier-free fragments of first-order temporal logic.

[3] The study of robust satisfaction and, more generally, of *multi-valued* and *quantitative* interpretation of logical formalisms is, of course, very old. We focus on works relevant to our motivation, *sensitive monitoring* of *continuous signals.*

namely, being positive for signals that satisfy the property and negative for violating signals. They propose an inductive procedure for computing the robustness degree, as well as a recent tool [12]. Their approach is behavior oriented, indicating how much should the *signal* be modified in order to satisfy or violate the property.

The measures introduced by Rizk et al. [22] are more property-oriented and intend to assess how far a given formula, written in some variant of LTL[4] augmented with numerical predicates, is from being an adequate description of a given simulation trace. Their real-valued degree of satisfaction is obtained first by replacing all constants in the formula by parameters and hence viewing all formulae admitting the same form as points in an Euclidean parameter space. They compute the set of all points that correspond to formulae satisfied by a given trace (or set of traces) and define a real-valued satisfaction/violation degree based on the distance between the original formula and the set of satisfied formulae. They use this measure to guide a search in the parameter space. Their approach has been integrated into the BIOCHAM tool [6] and has been applied to numerous biological examples.

In this paper we extend these works in several directions. First we propose an alternative quantitative semantics for MITL/STL which focuses on the robustness of satisfaction with respect to *time*. We then show how these two robustness measures can be combined into a generalized robustness measure which captures both space and time, from which both the space robustness of [11] and our time robustness are obtained as special cases. Then we present an efficient dense-time algorithm for computing these measures for piecewise-linear signals. Finally we extend the algorithm to compute the sensitivity of these measures with respect to parameter variations thus paving the way for the extension of sensitivity-based parameter-space exploration methodology of [7] to handle STL formulae.

2 Logics for Real-Valued Signals

In this section, we partly and briefly recall the framework set in [15,17] to define an appropriate logic for real-valued continuous time signals. In the rest of the paper, we implicitly assume the existence of a system under study whose state is described by a set of n variables $V = \{x_1,\ x_2, \ldots, x_n\}$. The domain of valuation of V is denoted by $\mathbb{D} = \mathbb{D}_1 \times \mathbb{D}_2 \times \ldots \times \mathbb{D}_n$. The domain $\mathbb{R}$ is the set of real numbers, $\mathbb{B} = \{\text{true},\ \text{false}\}$ is the Boolean domain and the time set is $\mathbb{T} = \mathbb{R}_{\geq 0}$. As a preparation for the real-valued semantics we will view $\mathbb{B}$ as $\{-1, +1\}$ rather than $\{0, 1\}$. Disjunction and conjunction are realized as $\max$ and $\min$ and negation as minus. This way the passage to the quantitative semantics of [11] will be immediate. A trace (or signal or behavior) w describing an evolution of the system is a function from $\mathbb{T}$ to $\mathbb{D}$. We define a set $P = \{p_1, p_2, \ldots, p_n\}$ of projectors, so that for a given trace w, $p_i(w[t]) = x_i[t]$ for all t. If the context is not ambiguous, we write $p_i[t]$ instead of $p_i(w[t])$.

We first consider continuous-time Boolean signals, i.e., when $\mathbb{D} = \mathbb{B}^n$. A popular logic to characterize such timed behaviors is the *Metric Interval Temporal Logic* (MITL) [2] whose grammar is given by:

[4] Their treatment of time, interpreting the *next* operator as referring to the next integration step, is too implementation-dependent to serve as a solid basis for defining time robustness.

$$\varphi := p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \,\mathcal{U}_{\mathcal{I}}\, \varphi_2 \tag{1}$$

where φ, φ_1 and φ_2 are MITL formulae, $p \in P = \{p_1, p_2, \ldots, p_n\}$ and $\mathcal{I}$ is an interval of the form $\mathcal{I} = (i_1, i_2)$, $(i_1, i_2]$, $[i_1, i_2)$ or $[i_1, i_2]$, where $i_1 < i_2$ are in $\mathbb{T}$. For t in $\mathbb{T}$, $t + \mathcal{I}$ is the set $\{t + t' \mid t' \in \mathcal{I}\}$. Traditionally the satisfaction of an MITL formula φ by a trace w at time t is denoted by $(w, t) \vDash \varphi$. We will use instead the characteristic function $\chi(\varphi, w, t)$ which is 1 when $(w, t) \vDash \varphi$ and -1 otherwise.

Definition 1 (Semantics). *The characteristic function of an MITL formula relative to a trace w at time t is defined inductively as*

$$\chi(p, w, t) = p[t] \tag{2}$$
$$\chi(\neg\varphi, w, t) = -\chi(\varphi, w, t) \tag{3}$$
$$\chi(\varphi_1 \wedge \varphi_2, w, t) = \min(\chi(\varphi_1, w, t), \chi(\varphi_2, w, t)) \tag{4}$$
$$\chi(\varphi_1 \,\mathcal{U}_{\mathcal{I}}\, \varphi_2, w, t) = \max_{t' \in t + \mathcal{I}} \min(\chi(\varphi_2, w, t'), \min_{t'' \in [t, t']} \chi(\varphi_1, w, t'')) \tag{5}$$

Note again that the semantics of *until* is equivalent to the more familiar notation:

$$(w, t) \vDash \varphi_1 \,\mathcal{U}_{\mathcal{I}}\, \varphi_2 \Leftrightarrow \exists t' \in t + \mathcal{I} \text{ s.t. } (w, t') \vDash \varphi_2 \text{ and } \forall t'' \in [t, t'], (w, t'') \vDash \varphi_1$$

where dense Boolean operations are replaced by dense $\min$ and $\max$. From the basic operators of MITL, other standard operators can be defined such as *eventually* and *always*: $\Diamond_{\mathcal{I}}\varphi \triangleq \text{true}\ \mathcal{U}_{\mathcal{I}}\ \varphi$, $\Box_{\mathcal{I}}\varphi \triangleq \neg\Diamond_{\mathcal{I}}\neg\varphi$.

Signal temporal logic (STL) [15] allows one to apply MITL-like reasoning to traces over $\mathbb{D} = \mathbb{R}^n$. The connection is done via a finite set $M = \{\mu_1, \ldots, \mu_k\}$ of predicates (Booleanizers), each mapping $\mathbb{R}^n$ to $\mathbb{B}$. For a given $1 \leq j \leq k$, the predicate μ_j is of the form $\mu_j \equiv f_j(x_1, x_2, \ldots, x_n) \geq 0$ where f_j is some real-valued function.

Definition 2. *We call x_i the* primary *signals of w and call their images by f_j* secondary *signals that we denote by $\{y_1, \ldots, y_k\}$.*

The syntax is thus identical to MITL except for the predicates in M replacing the atomic propositions in P. The semantics is identical to that of Definition 1 except for the base case (we interpret $\text{sign}(0)$ as 1):

$$\chi(\mu, w, t) = \text{sign}(f(x_1[t], \ldots, x_n[t])) \tag{6}$$

Figure 1 illustrates the semantics of the STL formulae $\Diamond_{[1,2]}(x_1 + 2x_2 - 2 \geq 0)$ relative to a two-dimensional real-valued signal. The methodology developed in [15,17] for deciding satisfaction of an STL formula φ by a real-valued signal is based on assigning to each sub-formula ψ of φ a *satisfaction signal*, that is a Boolean signal whose value at t is equal to $\chi(\psi, w, t)$. The computation goes bottom up (and backwards, for the future fragment of the logic) starting from the primary signals from which the secondary signals and their Booleanizations are computed, and then climbing up the parse tree of φ until the satisfaction signal of the top formula is computed. This procedure is similar in spirit to the novel translation from MITL to timed automata [16] based on the compositional principles of *temporal testers* initiated in [13,21] where each formula is associated with a transducer which computes the satisfaction signal of the formula based on those of its sub-formulae. In Section 4 we present a monitoring procedure for the quantitative semantics.

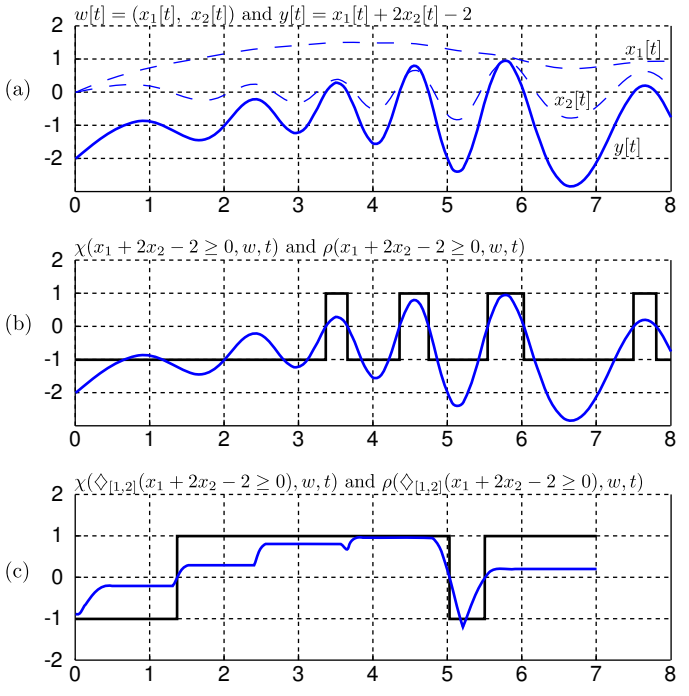


Fig. 1. A two-dimensional real-valued signal w and the time evolution of $\chi(\varphi, w, t)$ and $\rho(\varphi, w, t)$ for $\varphi \triangleq \Diamond_{[1,2]}(x_1 + 2x_2 - 2 \geq 0)$. (a) the primary signals x_1, x_2 and the secondary signal $y = x_1 + 2x_2 - 2$; (b) the truth value and spatial robustness of the sub-formula corresponding to $y \geq 0$; (c) the truth value and spatial robustness of $\Diamond_{[1,2]} y \geq 0$ (notice the smoothing effect of the non-punctual eventually operator).

With the flexibility of the definition of STL we can express a rich collection of relevant temporal properties for continuous signals. However, it has two drawbacks: first, the values of the primary and secondary signals at *different* time instances cannot "communicate" before being Booleanized and then handled by the temporal operators. This limitation can be alleviated by letting predicates use also the *shift* operator (which is a punctual version of $\Diamond$). Although our implementation allows this feature we will assume here only simple point-wise predicates in order to focus on the second drawback mentioned in the introduction: the loss of quantitative information due to Booleanization. In the next section, we propose several quantitative semantics for STL reflecting the *robustness*, in space and time, of satisfaction or violation.

3 Quantitative Semantics

The first quantitative measure of satisfaction that we present is a simple reformulation of the spatial robustness degree of [11], after which we proceed to a novel measure of *temporal* robustness and finally to a generalized measure that combines both. The definitions of all these measures are identical to Definition 1 except for the base cases.

Definition 3 (Space Robustness). *The space robustness of an STL formula relative to a trace w at time t, denoted by $\rho(\varphi, w, t)$, is defined inductively as*

$$\begin{aligned}
\rho(\mu, w, t) &= f(x_1[t], \ldots, x_n[t]) \text{ where } \mu \equiv f(x_1, \ldots, x_n) \geq 0 \\
\rho(\neg\varphi, w, t) &= -\rho(\varphi, w, t) \\
\rho(\varphi_1 \wedge \varphi_2, w, t) &= \min(\rho(\varphi_1, w, t), \rho(\varphi_2, w, t)) \\
\rho(\varphi_1 \,\mathcal{U}_{\mathcal{I}}\, \varphi_2, w, t) &= \max_{t' \in t+\mathcal{I}} \big(\min(\rho(\varphi_2, w, t'), \min_{t'' \in [t,t')} \rho(\varphi_1, w, t'')\big)
\end{aligned}$$

It is not hard to see that the definition above is sound in the sense of [11]: $\chi(\varphi, w, t) = \text{sign}(\rho(\varphi, w, t))$. Figure 1 illustrates the behavior of this measure. It captures the robustness of the satisfaction to noise in w. Let the point-wise distance between two finite signals of the same length be $||y - y'|| = \max_t |y[t] - y'[t]|$. The following is a reformulation of the results of [11]:

Theorem 1 (Property of Space Robustness). *If $\rho(\varphi, w, t) = r$ then for every w' in which every secondary signal satisfies $||y_j - y'_j|| < r$, $\chi(\varphi, w, t) = \chi(\varphi, w', t)$.*

While this semantics captures the robustness of satisfaction with respect to *point-wise* changes in the value of the signal or in constants appearing in the predicates, it does not fully capture the effect of changes in the constants appearing in the *temporal operators* of the formula nor in changes in the signal along the time axis. Such changes are often expressed using a *retiming function*, a monotone function $\alpha : \mathbb{T} \to \mathbb{T}$ which transforms a signal x to x' by letting $x'[t] = x[\alpha(t)]$. Figure 2 shows three different signals satisfying $\rho(\varphi, w_1, 0) = \rho(\varphi, w_2, 0) = \rho(\varphi, w_3, 0) > 0$ for the formula $\varphi \triangleq \Diamond_{[i_1,i_2]}(x > 0)$. Intuition tells us, however, that w_1 satisfies φ more robustly being positive during a large part of the $[i_1, i_2]$ interval while w_2 has only a short positive spike and w_3 almost misses the deadline for satisfying the positivity condition. All those signals have the same value of ρ because they admit the same maximal value in the interval and the point-wise space robustness cannot account for these differences. For the same reason, it cannot capture the similarity between w_3 and the property-violating signal w_4, obtained from w_3 by a slight shifting in time, that is, $w_4[t] = w_3[t - \varepsilon]$.

This observation motivates our definition of *time robustness* which indicates the effect on satisfaction of *shifting* events in *time*, where the term *event* refers to rising and falling edges in Boolean signals, the moments where certain secondary signals cross a threshold and their predicates change their truth value. Since the notion of a change in a truth value is discrete, we define time robustness with respect to MITL and Boolean signals but the definition applies as well to the standard Boolean semantics of STL where Booleanizers replace atomic propositions.

Definition 4 (Time Robustness). *The left and right time robustness of an MITL formula φ with respect to a trace w at time t are defined inductively by letting*

$$\begin{aligned}
\theta^-(p, w, t) &= \chi(\varphi, w, t) \cdot \max\{d \geq 0 \text{ s.t. } \forall t' \in [t-d, t],\ \chi(\varphi, w, t') = \chi(\varphi, w, t)\} \\
\theta^+(p, w, t) &= \chi(\varphi, w, t) \cdot \max\{d \geq 0 \text{ s.t. } \forall t' \in [t, t+d],\ \chi(\varphi, w, t') = \chi(\varphi, w, t)\}
\end{aligned}$$

and then applying to each of (θ^-, θ^+) the rules (3-4) as in Definition 1.

Figure 3 illustrates the time robustness of the satisfaction of p by a Boolean signal. Note that there are (unavoidable) discontinuities in the evolution of these measures at

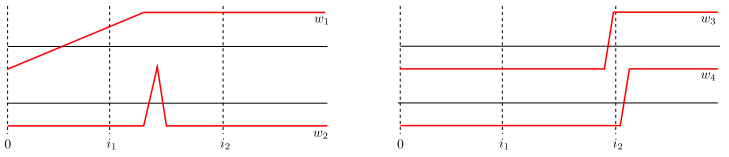


Fig. 2. Limitations of the point-wise quantitative semantics: signals w_1, w_2 and w_3 are considered as satisfying $\Diamond_{[i_1,i_2]}(x > 0)$ from $t = 0$ at the same degree, while the similarity between w_3 and the violating w_4 is not captured.

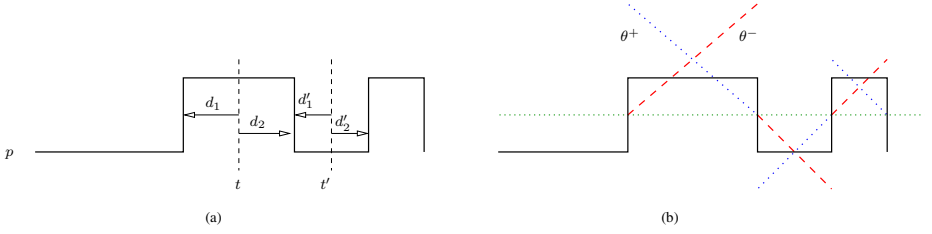


Fig. 3. (a) Illustration of time robustness of a propositional formula p with respect to a Boolean signal: $\theta^-(p, w, t) = d_1$; $\theta^+(p, w, t) = d_2$; $\theta^-(p, w, t') = -d'_1$; $\theta^+(p, w, t) = -d'_2$; (b) The evolution of θ^- and θ^+ with time.

rising/falling edges and that $\theta^-(p, w, t) + \theta^+(p, w, t)$ is constant inside an interval in which the truth value of p is uniform. The following property holds naturally for atomic propositions and is preserved by temporal operators:

Theorem 2 (Property of Time Robustness). *If $\theta^-(\varphi, w, t) = s$ (resp. $\theta^+(\varphi, w, t) = s$) then for any signal w' obtained from w by shifting events to the right (resp. to the left) by less than s, we have $\chi(\varphi, w, t) = \chi(\varphi, w', t)$.*

We can now move to a combined space-time robustness which reflects trade-offs between space and time robustness: the same signal will be more robust temporally if we are ready to compromise its spatial robustness and vice versa. The space-time robustness of an STL formula φ with respect to a trace w at time t is a family of function pairs $\{\theta^+_c, \theta^-_c\}_{c\in\mathbb{R}}$ which map every spatial robustness c to left/right temporal robustness. For an STL predicate μ, let $\chi_c(\mu, w, t) = \text{sign}(\rho(\mu, w, t) - c)$, that is, a Boolean whose value is positive if and only if the robustness of μ for w at t is at least c.

Definition 5 (Space-Time Robustness). *The temporal robustness of a formula φ relative to a spatial robustness c is obtained by letting*

$$\theta^-_c(\mu, w, t) = \theta^-(\chi_c(\mu, w, t))$$
$$\theta^+_c(\mu, w, t) = \theta^+(\chi_c(\mu, w, t))$$

and then applying the rules of Definition 4.

Geometrically, these functions define the basis of the largest rectangle of height c that can be constructed around t while remaining below the secondary signal associated with μ, as illustrated in Figure 4. In fact, the set of realizable triples $(\rho, \theta^-, \theta^+)$ represents the possible trade-offs (a kind of Pareto curve) between these measures. It could

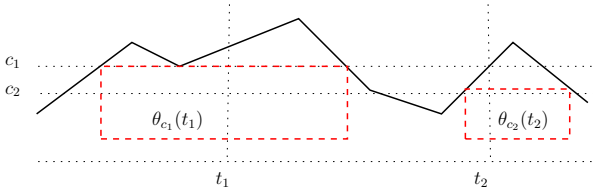


Fig. 4. Illustration of the combined time-space robustness

equivalently be captured by a function ρ_{d^-,d^+} which maps time robustness to space robustness. The previously-defined space and time robustness measures are obtained as the special cases: $\rho = \rho_{0,0}$, $\theta^- = \theta_0^-$ and $\theta^+ = \theta_0^+$.

4 Computing Robustness Degrees

In this section we present an algorithm to compute the different quantitative semantics for a given signal w and a formula φ. We focus on space robustness ρ but the computation for θ or θ_c (for any value of c) is similar. The robustness function $\rho(\varphi, w, \cdot)$ is computed inductively on the structure of the formula, beginning with the computation of the secondary signals based on the primary signals in w and the predicates in M. Thus we need to compute the right-hand side terms of the semantics in Definition 1 which reduces to the following subproblems corresponding, respectively, to operators $\neg$, $\wedge$ and $\mathcal{U}_{\mathcal{I}}$:

1. Given $y : \mathbb{T} \to \mathbb{R}$, compute $z : \mathbb{T} \to \mathbb{R}$ such that $\forall t \in \mathbb{T}$, $z[t] = -y[t]$;
2. Given $y, y' : \mathbb{T} \to \mathbb{R}$, compute $z : \mathbb{T} \to \mathbb{R}$ such that
$$\forall t \in \mathbb{T},\ z[t] = \min(y[t], y'[t]) \tag{7}$$
3. Given $y, y' : \mathbb{T} \to \mathbb{R}$ and an interval $\mathcal{I}$, compute $z : \mathbb{T} \to \mathbb{R}$ such that
$$\forall\, t \in \mathbb{T},\ z[t] = \max_{\tau \in t+\mathcal{I}} (\min(y'[\tau],\ \min_{s \in [t,\tau]} y[s])) \tag{8}$$

The first of these being trivial, we focus on the second and third. The difficulty lies in the fact that we compute *functions* and not only single values at specific time instants. Dealing with continuous time and space, a natural input for our algorithm is a sequence of time-stamped values of w obtained via variable step-size numerical simulation that we interpret as a *piecewise-linear* function by linear interpolation. More precisely, we assume that a secondary signal y is a piecewise-affine function of time with a finite sequence of points $\{t_k\}_{1<k<n_y}$ where its derivative changes. We assume that y is right-continuous and admits a right-derivative noted $dy[t] \triangleq \lim_{\varepsilon \to 0} \frac{y[t+\varepsilon]-y[t]}{\varepsilon}$, which is simply its slope at t. If y is not continuous at t, we note $y[t^-]$ (resp. $y[t^+]$) its limit to the left (resp. to the right) of t. Finally, we extend the trace to be unbounded by letting $y[t] = y[t_0]$ for $t < t_1$ and $y[t] = y[t_{n_y}]$ for $t > t_{n_y}$.[5] Clearly, to compute z satisfying

[5] There are various other ways to interpret temporal logic over finite traces, such as the weak semantics of [9] and other solutions surveyed in [17]. This topic is orthogonal to the rest of the paper.

(7) or (8) it is sufficient to know its values and its slopes at a finite sequence of time instances $\{r_k\}_{1\leq k\leq n_z}$ such that z is continuous and dz is constant on each interval $[r_k, r_{k+1})$. We note

$$\text{NextEvent}(z,t) = \begin{cases} \infty \text{ if } (z,dz) \text{ is continuous for } r > t \\ \min\{r > t \mid (z[r^-], dz[r^-]) \neq (z[r^+], dz[r^+])\} \text{ otherwise} \end{cases}$$

thus z can be computed incrementally by the generic Algorithm 1. An interesting feature of this approach is that it can easily be adapted to work *online* where input w is revealed progressively. Indeed, each time a new pair $(z(r_k), dz(r_k))$ is computed, the algorithm can be paused to wait for additional input data w needed to compute r_{k+1} and $(z(r_{k+1}), dz(r_{k+1}))$.

Algorithm 1. An iterative algorithm to compute the robustness z of the conjunction or the *until* of two secondary signals y and y'

1: **Init** r_1, $k = 1$
2: **Repeat**
3: Compute $(z[r_k], dz[r_k])$ from (y, y')
4: Compute $r_{k+1} = \text{NextEvent}(z, r_k)$ from (y, y')
5: Let $k = k + 1$
6: **Until** $r_k = \infty$

To implement Algorithm 1, we need to implement the intialization (line 1), the computation of (z, dz) at a given time instant (line 3) and the NextEvent function (line 4) based on the representation of y and y' sampled at $\{t_k\}_{1\leq k\leq n_y}$ and $\{t'_k\}_{1\leq k\leq n_y}$.

4.1 Conjunction: $z[t] = min(y[t], y[t'])$

Initialization and computation of (z, dz). In the case of the conjunction, we initialize r_1 to $\min(t_1, t'_1)$ and we note that computation of (z, dz) satisfying (7) is trivial except when $y[t] = y'[t]$. In that case it can be seen easily though that $dz[r] = \min(dy[z], dy'[r])$ (the $\min$ operator preserves the right-derivative) so that if we extend the $\min$ function with the lexicographic order, we have

$$\forall t \in \mathbb{T},\ (z[t], dz[t]) = \min\{\ (y[t], dy[t]),\ \ (y'[t], dy'[t])\ \} \tag{9}$$

Computation of next event. Observe first that since we know y and y', we know their NextEvent functions. Then, the slope of z can be discontinuous in $r > r_k$ only if r is a time event for y or y' or if $y[r] = y'[r]$. Thus there are three possibilities:

1. $(z[r_k], dz[r_k]) = (y[r_k], dy[r_k])$ then
$$\text{NextEvent}(z, r_k) = \min\{\ \text{NextEvent}(y, r_k),\ \ \arg\min_{t>r_k}\{y[t] = y'[t]\}\}$$
2. $(z[r_k], dz[r_k]) = (y'[r_k], dy'[r_k])$ then
$$\text{NextEvent}(z, r_k) = \min\{\ \text{NextEvent}(y', r_k),\ \ \arg\min_{t>r_k}\{y[t] = y'[t]\}\}$$
3. $(z[r_k], dz[r_k]) = (y[r_k], dy[r_k]) = (y'[r_k], dy'[r_k])$ then
$$\text{NextEvent}(z, r_k) = \min\{\ \text{NextEvent}(y, r_k),\ \text{NextEvent}(y', r_k)\}$$

4.2 The *Until* Operator: $z[t] = \max_{\tau \in t+\mathcal{I}}(\min(y'[\tau], \min_{s\in[t,\tau]} y[s]))$

The computation for the *until* operator involves the detection of the change in the maximal or minimal value of a signal over a moving time window. We note $\underline{y}[t,\tau] = \min_{s\in[t,\tau]} y[s]$, so that we have to compute

$$\forall\, t \in \mathbb{T},\ z[t] = \max_{\tau\in t+\mathcal{I}}(\min(y'[\tau], \underline{y}[t,\tau]))) \tag{10}$$

The value $z[t]$ is provided either by y or by y' at some time instant in $[t, t+i_2]$. The following result, that we use to compute (z, dz) and the NextEvent function, shows that if $z[t]$ is not provided by y', then it has to be $\underline{y}[t, t+i_1]$.

Lemma 1. *We say that τ is an admissible time for y' iff $y'[\tau] \leq \underline{y}[t,\tau]$. If there is no admissible time for y', then $z[t] = \underline{y}[t, t+i_1] = \min_{s\in[t,t+i_1]} y[s]$.*

Proof. Since there is no admissible time for y', there must be $t^* \in [t, t+i_2]$ such that $z[t] = y[t^*]$. Moreover, $y[t^*]$ has to be equal to $\min_{s\in[t,\tau]} y[s]$ for some τ in $[t+i_1, t+i_2]$. Since $[t, t+i_1] \subseteq [t,\tau]$, $z[t] = y[t^*] \leq \min_{s\in[t,t+i_1]} y[s]$. Furthermore, $z[t] = y[t^*] = \max_{\tau\in t+\mathcal{I}}(\min_{s\in[t,\tau]} y[s])$ and since $t+i_1 \in t+\mathcal{I}$, $z[t] \geq \min_{s\in[t,t+i_1]} y[s]$ so that $z[t] = \min_{s\in[t,t+i_1]} y[s]$.

Initialization and computation of (z, dz). Since $(z[t], dz[t])$ depends on the values of y and y' on the interval $[t, t+i_2]$, the first time event for z is $r_1 = \min(t_1 - i_2, t'_1 - i_2)$. Then to compute $z[t]$ for a given t, we first scan chronologically the values of y on $[t, t+i_1]$ to compute their minimum $\underline{y}[t, t+i_1]$. Then we scan the values of y' and y for τ in $[t+i_1, t+i_2]$, updating $\underline{y}[t,\tau]$, then getting the minimum with $y'[\tau]$ and finally the maximum with the value obtained before τ. At the end, we get the value $z[t]$.

In the process, we also compute $dz[t]$ starting by initializing it to $dy[t]$. Then, each time a new value for the minimum of y is found for some τ in the interval $(t, t+i_1)$, we compare the current value for $dz[t]$ with 0 (because τ does not depend on t) and keep the minimum. For $\tau = t+i_1$, we compare the current value for $(z[t], dz[t])$ with $(y'[t+i_1], dy'[t+i_1])$ and $(y[t+i_1], dy[t+i_1])$ and keep the minimum (in the lexicographic sense). For τ in the interval $(t+i_1, t+i_2)$, as long as we do not find an admissible time for y', we only update $dz[t]$ if $y[\tau]$ is equal to the current value of $z[t]$, in which case we set to $dz[t] = min(0, dz[t])$. When a new admissible time τ is found, $(z[t], dz[t])$ is updated to $\max((y'[\tau], 0), (z[t], dz[t]))$. Finally, if $t+i_2$ is admissible, we let $(z[t], dz[t]) = \max((y'[\tau], dy'[\tau]), (z[t], dz[t]))$. We note $\arg z[t]$ the time instant in $[t, t+i_1]$ when the value of $(z[t], dz[t])$ is determined.

Computation of next event. Assuming that we computed $(z[r_k], dz[r_k])$, we need to compute the minimum time $r_{k+1} > r_k$ such that z is discontinuous in r_{k+1} and/or $dz[r_{k+1}]$ is different from $dz[r_k]$. Firstly, an event can occur at $r > r_k$ due to an event for y or y' at r_k, $r_k + i_1$ or $r_k + i_2$, i.e. if

$$\begin{aligned} r = \min\{&\text{NextEvent}(y,t), \text{NextEvent}(y',t') \text{ such that} \\ &t \in \{r_k,\ r_k+i_1, r_k+i_2\}, t' \in \{r_k+i_1,\ r_k+i_2\}\} \end{aligned} \tag{11}$$

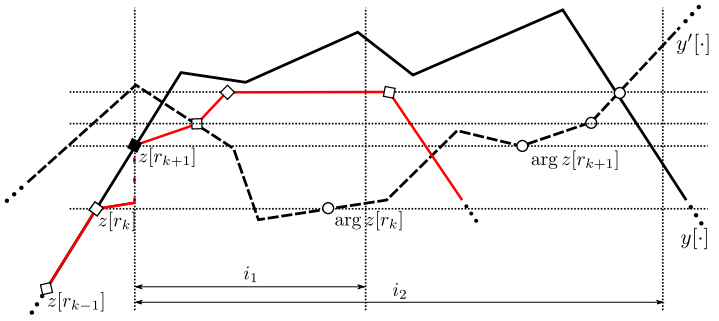


Fig. 5. Computation of *until* robustness. Here, $\arg z$ represents the point that determines z. For $\tau < r_k$, there is no admissible time for y' and $z[\tau] = y[\tau]$. The following admissible time is $r_k + i_1$, since $y[\tau] = y'[\tau + i_1]$ (second condition of (13)). At r_{k+1}, another time becomes admissible, which causes z to be discontinuous (first condition of (13)).

The second possibility of event depends on whether the value of $(z[r_k], dz[r_k])$ comes from y or y'. If there is no admissible time for y', we showed that $z[r_k]$ is $\underline{y}[r_k, r_k + i_1]$ so an event can occur when $\underline{y}[r, r + i_1]$ and/or its derivative is discontinuous. We can show that this can happen only if r satisfies:

$$\begin{array}{l} \min\{y[t_j], y[t_j^-] \mid r < t_j < r + i_1\} = \min(y[r], y[r + i_1]) \\ \text{or } \min\{y[t_j], y[t_j^-] \mid r < t_j < r + i_1\} > y[r] = y[r + i_1] \end{array} \tag{12}$$

Now, whether there are admissible times for y' or not, we have to monitor the appearance of new admissible (adm.) times. This can happen only if r satisfies:

$$\begin{array}{l} y[r] = \min\{y'[t'_j], y'[t'^-_j] \mid r + i_1 < t'_j < r + i_2 \text{ with } t'_j \text{ not adm. for } r_k\} \\ \text{or } y'[r + i_1] = y[r] \text{ with } r + i_1 \text{ not adm. for } r_k \\ \text{or } y'[r + i_2] = \underline{y}[r, r + i_2] \end{array} \tag{13}$$

Finally, we have to monitor discontinuities in the maximum values of y' among existing admissible times. They can happen only if

$$\begin{array}{l} r+i_1 \text{ is adm. and } y'[r] = \max\{y'[t'_j], y'[t_j^-] \mid r+i_1 < t'_j(\text{adm.}) < r+i_2\} \\ \text{or } r+i_2 \text{ is adm. and } y'[r+i_2] = \max\{y'[t'_j], y'[t'^-_j] \mid r+i_1 < t_j(\text{adm.}) < r+i_2\} \\ \text{or } y'[r+i_1] = y'[r+i_2] \text{ with } r+i_1,\ r+i_2 \text{ adm.} \end{array} \tag{14}$$

Then the expressions (11-14) provides conservative conditions for $r > r_k$ to be NextEvent(z, r_k). The algorithm computes the minimum $\tilde{r}_k > r_k$ satisfying one of these conditions, then $(z[\tilde{r}_k], dz[\tilde{r}_k])$ and iterates with $r_k = \tilde{r}_k$ until (z, dz) is found to be discontinuous in $\tilde{r}_k$. Figure 5 illustrates the process.

4.3 Computational Cost

The complexity of the robustness computation for a given signal and a formula is clearly linear with respect to the computational cost of Algorithm 1, the constant being the

size of the formula. The cost (time and space) of algorithm 1 depends on the number of events generated. In the case of conjunction, there can be as many as $n_y + n_{y'} + \max(n_y, n'_y)$ (the third term counting the maximal number of intersections between y and y') so it is linear in the number of samples in y and y'.

In the case of the *until* operator, this number is more difficult to estimate due to the complex inter-dependance of values of y and y' in the intervals $[t, t + i_2]$. We conjecture that this bound is also linear in $n_y + n_{y'}$. Then each time step requires the computation of (z, dz) for a given interval $[t, t + i_2]$ involving scanning and comparing the values of y and y' in this interval. The worst complexity for this operation is of the order of the sum of the maximum number of sampling points of y and y' that $[t, t+i_2]$ can contain. We write this number $n_{(y\cup y')\cap\mathcal{I}}$, so that our conjectured complexity is $\mathcal{O}\big((n_y + n_{y'}) \times n_{(y\cup y')\cap\mathcal{I}}\big)$. In practice, we found that the complexity is usually better since the value of z may have much less changes due to the fact that the maximum/minimum of a signal over a moving time window is likely to remain constant for long periods.

We compared our implementation with TaLiRo [12], the only other known tool for computing robustness degrees of temporal formulae. Their algorithm, for which the implementation details are not provided, appeared experimentally to behave linearly in the size of the input signals but, at least in some cases, exponentially (in time and memory) in the size of the formula. For the same data and formulae, our algorithm was behaving as (or better than) the above analysis suggests.[6]

5 Robustness Sensitivity

In [8,7] we have developed a methodology, based on numerical simulation and sensitivity analysis, to explore the parameter space of a dynamical system in order to determine the region in this space which induces some qualitative behavior. This work has been restricted so far to simple reachability properties and the development in this paper extends its applicability to the whole range of temporal properties, with sensitivity defined as follows.

Assume that the robust satisfaction of a formula φ by a signal w is parameterized by some $\lambda \in \mathbb{R}$, i.e., for $t \in \mathbb{R}$, its value is $\rho(\varphi, w, t, \lambda)$, and that it is differentiable with respect to λ, with derivative noted $d_\lambda\rho(\varphi, w, t, \lambda)$. As a simple illustration, consider the predicate μ: $x_1 + 2x_2 - \lambda \geq 0$ with the corresponding secondary signal y. We have $\rho(x_1+2x_2-\lambda \geq 0, w, t, \lambda) = y[t, \lambda] = x_1[t]+2x_2[t]-\lambda$, which is differentiable with respect to λ, with $d_\lambda y[t, \lambda] = -1$. Another common situation is when the signal results from the simulation of a system with an uncertain parameter λ. Then w is a function of λ and we can get the derivative of the primary signals from a sensitivity-aware simulator and deduce those of the secondary signals by applying the chain rule.

Having defined the robustness sensitivity to λ for some base formulae (supposedly for all λ and t), we remark that our algorithm presented above can be easily adapted to compute the robustness sensitivity of any formula built from these base formulae. As for the robustness, the core of the algorithm needs only to implement the derivative

[6] See `http://www-verimag.imag.fr/~donze/breach_page.html` for experimental data.

of some function z satisfying $z[t,\lambda] = -y[t,\lambda]$, the relation (7) or the relation (8) including λ, corresponding to $\neg$, $\wedge$ and $\mathcal{U}_{\mathcal{I}}$. We first observe that if y and y' are differentiable with respect to λ, then $z = \min(y, y')$ is differentiable everywhere except when $y[t,\lambda] = y'[t,\lambda]$ for some $t \in \mathbb{T}$. However, it has a right- and left-derivative at this point which are

$$d^+_\lambda z[t,\lambda] = \min(d^+_\lambda y[t,\lambda], d^+_\lambda y'[t,\lambda]) \text{ and } d^-_\lambda z[t,\lambda] = \max(d^-_\lambda y[t,\lambda], d^-_\lambda y'[t,\lambda]) \tag{15}$$

Thus we can adapt Algorithm 1 to compute the right- and left-derivatives of z in a way similar to the way we compute dz. In addition to computing $(z[r_k]), dz[r_k])$, we compute $(d^+_\lambda z[r_k,\lambda], d^-_\lambda z[r_k,\lambda])$: each time z is updated with the comparison of two signals taking the same value, we update $d^+_\lambda z$ and $d^-_\lambda z$ using rule (15).

6 Discussion

We have contributed to further proliferation of logic-based ideas to the study of continuous and hybrid systems. Temporal logic offers a complementary way to evaluate real-valued signals, a way which is different from other commonly-used norms, measures and metrics, most of which are either point-wise or based on summation and averaging. We strongly believe that the measures introduced in this paper will find their application niche in situations where the interaction between timing and magnitude is non trivial as is the case in the design of mixed-signal circuits or the analysis of biochemical pathways.

Future work includes the application of these measures and algorithms to the exploration of the parameter space of examples coming from the above application domains, including the incorporation of the sensitivity measure into gradient-based optimization procedures. To make the exploration procedure more effective, we intend to augment these measures with a more refined *diagnostics* to indicate more precisely what (Boolean combinations of) changes in the primary signals are required or sufficient in order to secure satisfaction. To this end we will need to propagate the information down from the secondary to primary signals and resolve possible conflicts due to the fact that the same primary signal may appear in several predicates.

References

1. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science 126(2), 183–235 (1994)
2. Alur, R., Feder, T., Henzinger, T.A.: The benefits of relaxing punctuality. J. ACM 43(1), 116–146 (1996)
3. Antoniotti, M., Policriti, A., Ugel, N., Mishra, B.: Model building and model checking for biochemical processes. Cell Biochem. Biophys. 38(3), 271–286 (2003)
4. Asarin, E., Caspi, P., Maler, O.: Timed regular expressions. J. ACM 49(2), 172–206 (2002)
5. Bensalem, S., Peled, D. (eds.): RV 2009. LNCS, vol. 5779. Springer, Heidelberg (2009)
6. Calzone, L., Fages, F., Soliman, S.: Biocham: an environment for modeling biological systems and formalizing experimental knowledge. Bioinformatics (Oxford, England) 22(14), 1805 (2006)

7. Donzé, A., Krogh, B., Rajhans, A.: Parameter synthesis for hybrid systems with an application to simulink models. In: Majumdar, R., Tabuada, P. (eds.) HSCC 2009. LNCS, vol. 5469, pp. 165–179. Springer, Heidelberg (2009)
8. Donzé, A., Maler, O.: Systematic simulations using sensitivity analysis. In: Bemporad, A., Bicchi, A., Buttazzo, G. (eds.) HSCC 2007. LNCS, vol. 4416, pp. 174–189. Springer, Heidelberg (2007)
9. Eisner, C., Fisman, D., Havlicek, J., Lustig, Y., McIsaac, A., Van Campenhout, D.: Reasoning with temporal logic on truncated paths. In: Hunt Jr., W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 27–39. Springer, Heidelberg (2003)
10. Fages, F., Rizk, A.: From model-checking to temporal logic constraint solving. In: Gent, I.P. (ed.) CP 2009. LNCS, vol. 5732, pp. 319–334. Springer, Heidelberg (2009)
11. Fainekos, G.E., Pappas, G.J.: Robustness of temporal logic specifications for continuous-time signals. Theoretical Computer Science 410(42), 4262–4291 (2009)
12. Fainekos, G.E., Pappas, G.J.: A User Guide for TaLiRo V0.1 (2009)
13. Kesten, Y., Pnueli, A.: A compositional approach to CTL* verification. Theor. Comput. Sci. 331(2-3), 397–428 (2005)
14. Koymans, R.: Specifying real-time properties with metric temporal logic. Real-Time Syst. 2(4), 255–299 (1990)
15. Maler, O., Nickovic, D.: Monitoring temporal properties of continuous signals. In: FORMATS/FTRTFT, pp. 152–166 (2004)
16. Maler, O., Nickovic, D., Pnueli, A.: From MITL to timed automata. In: Asarin, E., Bouyer, P. (eds.) FORMATS 2006. LNCS, vol. 4202, pp. 274–289. Springer, Heidelberg (2006)
17. Maler, O., Nickovic, D., Pnueli, A.: Checking temporal properties of discrete, timed and continuous behaviors. In: Pillars of Computer Science, pp. 475–505 (2008)
18. Manna, Z., Pnueli, A.: The Temporal Logic of Reactive and Concurrent Systems: Specification. Springer, New York (1991)
19. Nickovic, D., Maler, O.: AMT: A property-based monitoring tool for analog systems. In: Raskin, J.-F., Thiagarajan, P.S. (eds.) FORMATS 2007. LNCS, vol. 4763, pp. 304–319. Springer, Heidelberg (2007)
20. Pnueli, A.: The temporal logic of programs. In: Proc. 18th Annual Symposium on Foundations of Computer Science (FOCS), pp. 46–57 (1977)
21. Pnueli, A., Zaks, A.: On the merits of temporal testers. In: Grumberg, O., Veith, H. (eds.) 25 Years of Model Checking. LNCS, vol. 5000, pp. 172–195. Springer, Heidelberg (2008)
22. Rizk, A., Batt, G., Fages, F., Soliman, S.: On a continuous degree of satisfaction of temporal logic formulae with applications to systems biology. In: Heiner, M., Uhrmacher, A.M. (eds.) CMSB 2008. LNCS (LNBI), vol. 5307, pp. 251–268. Springer, Heidelberg (2008)

Combining Symbolic Representations for Solving Timed Games

Rüdiger Ehlers[1], Robert Mattmüller[2], and Hans-Jörg Peter[1]

[1] Reactive Systems Group
Saarland University, Germany
{ehlers,peter}@cs.uni-saarland.de
[2] Foundations of Artificial Intelligence Group
Freiburg University, Germany
mattmuel@informatik.uni-freiburg.de

Abstract. We present a general approach to combine symbolic state space representations for the *discrete* and *continuous* parts in the synthesis of winning strategies for timed reachability games. The combination is based on abstraction refinement where *discrete* symbolic techniques are used to produce a sequence of abstract timed game automata. After each refinement step, the resulting abstraction is used for computing an under- and an over-approximation of the timed winning states. The key idea is to identify large relevant and irrelevant parts of the precise weakest winning strategy already on coarse, and therefore simple, abstractions. If neither the existence nor nonexistence of a winning strategy can be established in the approximations, we use them to guide the refinement process. Based on a prototype that combines binary decision diagrams [7,9] and difference bound matrices [5], we experimentally evaluate the technique on standard benchmarks from timed controller synthesis. The results clearly demonstrate the potential of the new approach concerning running time and memory consumption compared to the classical on-the-fly algorithm implemented in Uppaal-Tiga [10,4].

1 Introduction

In the last two decades, the timed automaton model by Alur and Dill [2] has become a de-facto standard for modeling timed asynchronous systems. A natural extension to their classical definition is to distinguish between internally and externally controllable behaviors [15,3]. The automated analysis of such so-called *open* systems requires a game-based computational model where an internal controller plays against an external environment. Solving problems defined on open systems (such as, e.g., timed controller synthesis [15]) is an active area of research [15,3,13,1,10,4,8,16] and usually corresponds to computing winning strategies in two-player games played on timed automata.

A predominant source of complexity in this setting is the large size of the concurrent control structure induced by a network of communicating timed automata. Current timed game solvers such as Uppaal-Tiga [10,4] represent the continuous parts symbolically, but the location information explicitly. Hence,

K. Chatterjee and T.A. Henzinger (Eds.): FORMATS 2010, LNCS 6246, pp. 107–121, 2010.

such *semi-symbolic* representations fail in the analysis of timed systems with many concurrent components causing a *discrete* blow-up in the state space.

In this paper, we tackle this problem by introducing an abstraction refinement approach that uses *discrete symbolic techniques* (e.g., binary decision diagrams (BDDs) [7,9,18]) to produce a sequence of *syntactic* abstractions with increasing precision of the input network of timed automata. We obtain the abstractions by merging locations such that the abstract control structure strictly weakens one player and strengthens her opponent. For each abstraction, we apply traditional solving algorithms to obtain an under- and an over-approximation of the winning states of the reachability player (e.g., one can use [10], which works fine for timed games with few locations, to obtain under-approximations).

Instead of solving the original game on the most precise control structure directly, our key idea is to solve a sequence of simpler games where each solving process reuses the approximations obtained from the previous one. That is, we use winning state set approximations computed on coarse (and therefore simple) abstractions to (1) characterize interpolants for refinement that ensure an increase in precision, and (2) derive pruning rules and optimizations that accelerate subsequent game solving processes over finer abstractions. Both soundness and effectiveness of our approach rely on the fact that whenever an abstract state appears in an under-approximation, all subsumed concrete states are surely winning, and dually, whenever an abstract state is not contained in an over-approximation, all subsumed concrete states are surely not winning.

In our prototype, the use of BDDs allows us to represent sets of locations efficiently and to refine abstract games arbitrarily while retaining an algorithmically simple check for the existence of abstract transitions. Based on (federations of) difference bound matrices (DBMs) [5], we use our own implementation of [10] for obtaining winning state set approximations.

Example. Consider the timed game automaton $\mathcal{G}$ given in Fig. 1(a), where l_0 is the initial and l_3 is the goal location. The reachability player ($\Diamond$) controls the dashed edges and wants to reach l_3 while the safety player ($\Box$) controls the solid edges and wants to avoid l_3. We abstract $\mathcal{G}$ by merging its locations. The abstract locations so obtained are connected via abstract transitions which are either (1) surely available, i.e., there is a corresponding concrete transition from each represented location, or (2) potentially available, i.e., there is a corresponding concrete transition from some represented location. Figures 1(b) and 1(c) show abstract automata with one abstract location representing the concrete goal location l_3 and one abstract location representing the remaining locations l_0, l_1, and l_2. In $\lceil\mathcal{G}\rceil_0$, we strengthen $\Diamond$ by letting her play on the potentially available transitions and weaken $\Box$ by letting him play on the ones that are surely available. Dually, we weaken $\Diamond$ and strengthen $\Box$ in $\lfloor\mathcal{G}\rfloor_0$. In $\lceil\mathcal{G}\rceil_0$, the abstract initial location (together with the initial clock valuation $x = 0$) is winning for $\Diamond$, whereas in $\lfloor\mathcal{G}\rfloor_0$, it is winning for $\Box$. Hence, we cannot determine the winner of $\mathcal{G}$ based on this coarse initial abstraction. Therefore, we refine the abstraction by separating l_0 in an additional abstract location. The finer abstractions are shown in Figures 1(d) and 1(e). Now, $\Diamond$ wins in $\lfloor\mathcal{G}\rfloor_1$ and, therefore, also in $\mathcal{G}$.

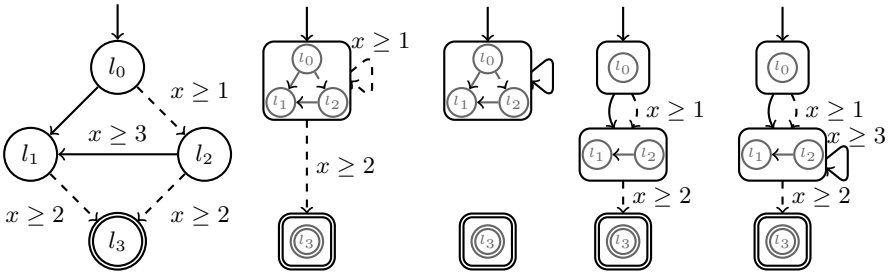


Fig. 1. Game automaton $\mathcal{G}$, abstractions $\lceil \mathcal{G} \rceil_0 / \lfloor \mathcal{G} \rfloor_0$, and refinements $\lceil \mathcal{G} \rceil_1 / \lfloor \mathcal{G} \rfloor_1$

Related Work. The definition of the game solving problem in the framework of timed automata [2] was given by Maler et al. [15,3]. In their fundamental work, the decidability of the problem was shown by demonstrating that the standard discrete attractor construction [19] on the region graph suffices to obtain winning strategies. Henzinger and Kopke showed that this construction is theoretically optimal by proving EXPTIME-completeness of the problem [13]. A first on-the-fly solving technique was proposed by Altisen and Tripakis which, however, requires an expensive preprocessing step [1]. As a remedy to this problem, Cassez et al. proposed a fully on-the-fly solving algorithm that combines the backward attractor construction with a forward zone graph exploration [10,4]. Recently, we developed an incremental variant that takes the compositional nature of networks of timed automata into account [16]. As a continuation of this line of research, the approach presented here can be seen as a further generalization that (1) provides the general basis for more fine-grained (location-based) abstractions, and (2) reports on a concrete application combining BDDs and DBMs.

Henzinger et al. adapted *counterexample-guided abstraction refinement* for games [12] where abstractions are defined over game states, counterexamples are abstract strategies, and refinement corresponds to the splitting of abstract game states. De Alfaro et al. introduced three-valued abstractions [11] where the refinement process is guided by differences between under- and over-approximations of the game states. Due to their *semantic* (i.e., state-based) natures, both techniques can be seen as a generalization of the approach presented in this paper. However, due to the lack of suitable data structures, an immediate implementation of these techniques for timed games, resulting in an efficient solving algorithm, appears not (yet) possible. We argue that location-based abstractions are an interesting sweet spot between granularity and implementability.

We propose an optimization technique that prunes irrelevant moves early in the refinement process which resembles *slicing* from model checking. Brückner et al. proposed a technique that combines slicing with abstraction refinement [6]. While their work only considers model checking problems for closed systems, our approach is capable of handling the strictly more general class of open systems.

Outline. Section 2 recalls the necessary foundations. In Sect. 3, we first formalize the notion of abstract games and give an algorithm that constructs an abstract game from a given concrete game and a location partition. Based on that, Sect. 4 describes an approximation-guided refinement loop, which is the formal basis for our prototype implementation presented in Sect. 5.

2 Preliminaries

2.1 Timed Games

We consider two-player, zero-sum reachability games played on timed automata. We distinguish between the *reachability* player $\Diamond$ whose objective is to eventually reach some goal state, and the *safety* player $\Box$ whose objective is to always avoid goal states. Following the setting of [3], we assume that timed automata are strongly nonzeno, i.e., there are no cycles where a player can play a time-convergent sequence of moves.

Timed Game Automata. A *timed game automaton* (TGA) [2,15,3] $\mathcal{G}$ is a tuple (L, I, Δ, X, G), where L is a finite set of locations, $I \subseteq L$ is a set of initial locations, $\Delta \subseteq L \times \mathcal{C}(X) \times \mathcal{P}(X) \times L$ is the set of transitions[1], X is a finite set of real valued clocks, and $G \subseteq L$ is a set of goal locations. We distinguish between controller $\Delta_\Box$ and environment transitions $\Delta_\Diamond$ such that $\Delta = \Delta_\Box \uplus \Delta_\Diamond$. The clock constraints $\varphi \in \mathcal{C}(X)$ are recursively defined as $\varphi = \mathbf{true} \mid x \bowtie c \mid \varphi_1 \wedge \varphi_2$, where x is a clock in X, c is a constant in $\mathbb{N}_0$, $\bowtie \in \{<, \leq, \geq, >\}$, and φ_1, φ_2 are constraints in $\mathcal{C}(X)$. A *clock valuation* $\boldsymbol{t} : X \to \mathbb{R}_{\geq 0}$ assigns a non-negative value to each clock and can also be represented by a $|X|$-dimensional vector $\boldsymbol{t} \in \mathcal{R}$ where $\mathcal{R} = \mathbb{R}_{\geq 0}^X$ denotes the set of all clock valuations. For a constraint $\varphi \in \mathcal{C}(X)$, we define $[\![\varphi]\!] = \{\boldsymbol{t} \in \mathcal{R} \mid \boldsymbol{t} \models \varphi\}$. We denote clock resets as $\boldsymbol{t}[\lambda := 0]$, for a set $\lambda \subseteq X$, and uniform time elapse as $\boldsymbol{t} + d$, for a $d \in \mathbb{R}_{\geq 0}$.

A *partition* of the locations L of a TGA $\mathcal{G}$ is a set $\Pi = \{\pi_1, \ldots, \pi_n\} \in \mathcal{P}(\mathcal{P}(L) \setminus \{\emptyset\})$ such that $\bigcup_{i=1}^{n} \pi_i = L$ and $\pi_i \cap \pi_j = \emptyset$ for $i \neq j$. We say that a partition Π' is *finer* than a partition Π, written as $\Pi \prec \Pi'$, iff $|\Pi| < |\Pi'|$ and $\forall \pi' \in \Pi' : \exists \pi \in \Pi : \pi' \subseteq \pi$. The *refinement* of a partition Π with a set $R \subseteq L$, is defined as $\Pi|R = \bigcup_{\pi \in \Pi} \{\pi \cap R, \pi \setminus R\} \setminus \{\emptyset\}$. The most fine-grained partition is denoted as Π with $|\Pi| = |L|$.

Timed Game Structures. The semantics of timed game automata is defined in terms of *timed game structures*. A timed game structure (TGS) $\mathcal{S}$ is a tuple $(S, S_0, \Gamma_\Box, \Gamma_\Diamond)$ where S is an infinite set of states, $S_0 \subseteq S$ are the initial states, and $\Gamma_\Box, \Gamma_\Diamond \subseteq S \times S$ are the moves of the players. A TGA $(L, I, \Delta_\Box \cup \Delta_\Diamond, X, G)$ induces a timed game structure $\mathcal{S} = (L \times \mathcal{R}, I \times \{\mathbf{0}\}, \Gamma_\Box, \Gamma_\Diamond)$ with

$$\Gamma_p = \quad (s, s') \mid \exists d > 0 \colon s' = s + d \quad \cup$$

[1] For the sake of simplicity, we omit transition labels in our formal definition since control-related concepts such as synchronization or integer variables are just technicalities in the construction of the symbolic discrete transition relation.

$$((l, \boldsymbol{t}), (l', \boldsymbol{t}')) \mid \exists \langle l, \varphi, \lambda, l' \rangle \in \Delta_p \colon \boldsymbol{t} \models \varphi \wedge \boldsymbol{t}' = \boldsymbol{t}[\lambda := 0] \ ,$$

for a player $p \in \{\Box, \Diamond\}$, where, for a game state $s = (l, \boldsymbol{t}) \in S$ and a delay $d \in \mathbb{R}_{\geq 0}$, we write $s + d$ for $(l, \boldsymbol{t} + d)$.

Strategies and Outcomes. A strategy is a function that determines a particular player's decisions during the course of a game. In general, a strategy is defined over a history of events. However, for reachability games under complete information, it suffices to consider state-based (or memoryless) strategies [15,3]. Formally, a *memoryless strategy* for player p is a function $f_p : S \to S$ such that all $s \in S$ are mapped to an s' with $(s, s') \in \Gamma_p$. Such an s' always exists because even if there is no successor location of the current location, it is always possible to play a time elapse move.

The notion of an *outcome* of a pair of strategies $f_\Diamond$ and $f_\Box$ defines the set of states that are reached if player $\Diamond$ sticks to $f_\Diamond$ and player $\Box$ sticks to $f_\Box$. Let s and s' be two states in S with $s' = f_p(s)$, $p \in \{\Box, \Diamond\}$, then the *time elapse* between s and s', written as $\delta(s, s')$, is defined as (1) $\delta(s, s') = d$, if $s' = s + d$, for a $d \in \mathbb{R}_{>0}$; (2) $\delta(s, s') = 0$, otherwise. The set $\mathsf{Outcome}(f_\Diamond, f_\Box) \subseteq S$ is the smallest subset of S (wrt. set inclusion), such that the following holds:

- $S_0 \subseteq \mathsf{Outcome}(f_\Diamond, f_\Box)$;
- if $s \in \mathsf{Outcome}(f_\Diamond, f_\Box)$, then
  $f_\Box(s) \in \mathsf{Outcome}(f_\Diamond, f_\Box)$, if $\delta(s, f_\Box(s)) < \delta(s, f_\Diamond(s))$, and
  $f_\Diamond(s) \in \mathsf{Outcome}(f_\Diamond, f_\Box)$, if $\delta(s, f_\Diamond(s)) \leq \delta(s, f_\Box(s))$.

With this definition of $\mathsf{Outcome}$, we assume that (1) player $\Diamond$ chooses the initial state, and (2) the scheduler resolving concurrent moves is always playing in favor for player $\Diamond$. Note that this captures the controller synthesis problem accurately since any actual controller implementation (player $\Box$) has to be robust wrt. any low-level scheduling policy or arbitrary environment (player $\Diamond$). Moreover, along with complementary winning objectives, the timed games considered here are always determined and their semantics is equivalent to UPPAAL-TIGA [10,4].

Timed Reachability Games. Let $\mathcal{S} = (S, S_0, \Gamma_\Box, \Gamma_\Diamond)$ be a TGS and $K \subseteq S$ a set of goal states. Then $(\mathcal{S}, K)$ represents a *timed reachability game*. Player $\Diamond$ wins $(\mathcal{S}, K)$ iff she can enforce a visit to K. More formally, player $\Diamond$ wins iff $\exists f_\Diamond \forall f_\Box \colon \mathsf{Outcome}(f_\Diamond, f_\Box) \cap K \neq \emptyset$. A TGA $\mathcal{G}$ with goal locations G induces a timed reachability game $\mathsf{Game}(\mathcal{G}) = (\mathcal{S}, K)$ such that $\mathcal{S}$ is the game structure induced by $\mathcal{G}$ and $K = G \times \mathcal{R}$ is the set of $\mathcal{G}$'s goal states.

2.2 Solving Timed Games

Solving a timed reachability game $(\mathcal{S}, K)$ means computing the set of states from which player $\Diamond$ has a strategy to enforce an outcome that contains some states from K. Before we come to the actual solving algorithm, we formalize the notion of controllability. For a TGS $\mathcal{S} = (S, S_0, \Gamma_\Box, \Gamma_\Diamond)$, the *timed enforceable predecessor operator* $\mathsf{PreEnf} : \mathcal{P}(S) \to \mathcal{P}(S)$ for player $\Diamond$ is defined as

$$\mathsf{PreEnf}(Y) = \quad r \in S \mid \exists s \in Y : (r, s) \in \Gamma_\Diamond \wedge (\forall d > 0 : s = r + d$$

$$\Rightarrow \forall 0 \leq d' < d : \forall (r', s') \in \Gamma_{\Box} : r' = r + d' \Rightarrow s' \in Y) \quad .$$

Intuitively, $\mathsf{PreEnf}(Y)$ comprises the source states of $\Diamond$-moves leading to Y that (1) change the location or (2) delay with no spoiling $\Box$-move in between. It was shown in [15,3] and later in [10] that PreEnf can be effectively computed using clock regions or clock zones.

We define those states from which player $\Diamond$ has a winning strategy to enforce an outcome that eventually visits some state in K as the *attractor* of K. For a reachability game $(\mathcal{S}, K)$, the computation of the attractor $\mathsf{Attr}(\mathcal{S}, K) \subseteq S$ is carried out by iteratively applying PreEnf in a least fixed point construction on K, i.e., $\mathsf{Attr}(\mathcal{S}, K)$ corresponds to $\mu A.A \cup K \cup \mathsf{PreEnf}(A)$ [15,3]. Note that any starting point A', with $K \subseteq A' \subseteq \mathsf{Attr}(\mathcal{S}, K)$, converges to $\mathsf{Attr}(\mathcal{S}, K)$ in the fixed point construction. We will write $\mathsf{Attr}(\mathcal{G})$ as an abbreviation for $\mathsf{Attr}(\mathsf{Game}(\mathcal{G}))$. Player $\Diamond$ wins $\mathsf{Game}(\mathcal{G})$ iff $S_0 \cap \mathsf{Attr}(\mathcal{G}) \neq \emptyset$. Dually, player $\Box$ wins $\mathsf{Game}(\mathcal{G})$ iff player $\Diamond$ does not win, i.e., $S_0 \cap \mathsf{Attr}(\mathcal{G}) = \emptyset$.

Theorem 1. *[13] For a TGA $\mathcal{G}$, constructing $\mathsf{Attr}(\mathcal{G})$ is complete for EXPTIME.*

From a practical point of view, a careful analysis shows that the application of the (nonconvex) symbolic PreEnf operator is very expensive compared to a zone-based forward analysis. For this purpose, the authors of [10] propose an on-the-fly game solving algorithm based on an interleaved fixed point construction that alternates between a forward exploration of the reachable states and a backward propagation of the attractor. Here, the number of PreEnf applications is reduced at the cost of introducing forward steps. In combination with the abstraction refinement technique presented in this paper, we use this algorithm to compute attractor under-approximations. In the following, we refer to algorithms such as [10] as *backend solving algorithms*.

2.3 Boolean Functions and Binary Decision Diagrams

Sets of locations can be represented by Boolean functions (BFs) $F : \mathcal{P}(\mathcal{B}) \rightarrow$ for some finite set of variables $\mathcal{B}$. In practice, *reduced ordered binary decision diagrams* (BDDs) [7,9] are the predominantly used data structure for this task. Since the usual operations on Boolean functions such as conjunction, disjunction and negation can be implemented as manipulations of BDDs, we treat Boolean functions and BDDs interchangeably here. In addition, BDDs support existential (and universal) abstraction. Given a set of variables $\mathcal{B}' \subseteq \mathcal{B}$ and a BDD F, the existential abstraction of F wrt. $\mathcal{B}'$ is written as $\exists \mathcal{B}'.F$ and denotes the BDD that maps those and only those $x \subseteq \mathcal{B}$ to **true** for which there exists some $x' \subseteq \mathcal{B}'$ such that $F(x' \cup (x \setminus \mathcal{B}')) = \mathbf{true}$.

3 Abstract Timed Games

We abstract a TGA by merging its locations such that the resulting abstract control structure strictly privileges one player and penalizes her opponent. We

can do this asymmetric abstraction in two ways: (1) we can weaken player $\Diamond$ and strengthen player $\Box$ to obtain a *weakened* reachability game; (2) we can strengthen player $\Diamond$ and weaken player $\Box$ to obtain a *strengthened* reachability game. In this section, we first introduce the formal model which acts as a basis for showing soundness and completeness of our approach. Then, we describe a Boolean function-based algorithm for constructing abstractions.

3.1 Abstract Timed Game Automata

A TGA $\mathcal{G} = (L, I, \Delta, X, G)$ with $\Delta = \Delta_\Diamond \uplus \Delta_\Box$ and a partition Π of L induce a *weakened TGA* $\lfloor \mathcal{G} \rfloor_\Pi$ and a *strengthened TGA* $\lceil \mathcal{G} \rceil_\Pi$:

$$\lfloor \mathcal{G} \rfloor_\Pi = (\Pi, \lfloor I \rfloor_\Pi, \lfloor \Delta_\Diamond \rfloor_\Pi \cup \lceil \Delta_\Box \rceil_\Pi, X, \lfloor G \rfloor_\Pi);$$
$$\lceil \mathcal{G} \rceil_\Pi = (\Pi, \lceil I \rceil_\Pi, \lceil \Delta_\Diamond \rceil_\Pi \cup \lfloor \Delta_\Box \rfloor_\Pi, X, \lceil G \rceil_\Pi).$$

Here, the *weak abstracting operator* $\lfloor \cdot \rfloor$ and the *strong abstracting operator* $\lceil \cdot \rceil$ are defined as follows. For any set $L' \subseteq L$ (and in particular I and G), we define

$$\lfloor L' \rfloor_\Pi = \{\pi \in \Pi \mid \pi \subseteq L'\} \text{ and}$$
$$\lceil L' \rceil_\Pi = \{\pi \in \Pi \mid \pi \cap L' \neq \emptyset\}.$$

Furthermore, for any set $\Delta' \subseteq \Delta$, we define

$$\lfloor \Delta' \rfloor_\Pi = \{\langle \pi, \varphi, \lambda, \pi' \rangle \mid \forall l \in \pi : \exists l' \in \pi' : \langle l, \varphi, \lambda, l' \rangle \in \Delta'\} \text{ and}$$
$$\lceil \Delta' \rceil_\Pi = \{\langle \pi, \varphi, \lambda, \pi' \rangle \mid \exists l \in \pi : \exists l' \in \pi' : \langle l, \varphi, \lambda, l' \rangle \in \Delta'\}.$$

Intuitively, transitions in $\lfloor \Delta' \rfloor_\Pi$ are *surely* available, while transitions in $\lceil \Delta' \rceil_\Pi$ are *potentially* available. It is easy to see that $\lfloor Y \rfloor_\Pi \subseteq \lceil Y \rceil_\Pi$, for every set of locations or transitions Y. We say that a pair of abstract locations $(\pi_1, \pi_2) \in \Pi \times \Pi$ represents a *potential connection* in a TGA $\mathcal{G}$ wrt. a partition Π iff there is a connecting transition from π_1 to π_2 in $\lceil \Delta \rceil_\Pi$. The following lemma states that a refinement never introduces new potential connections.

Lemma 1. *For a TGA $\mathcal{G}$ with locations L and partitions Π and Π' of L with $\Pi \prec \Pi'$, if $(\pi_1, \pi_2) \in \Pi \times \Pi$ is not a potential connection in $\mathcal{G}$ wrt. Π, then there is no potential connection $(\pi_1', \pi_2') \in \Pi' \times \Pi'$ in $\mathcal{G}$ wrt. Π' with $\pi_1' \subseteq \pi_1 \wedge \pi_2' \subseteq \pi_2$.*

In order to compare abstract attractor sets for different partitions, we need to *flatten* them: let $a = (\pi, \boldsymbol{t}) \in \mathcal{P}(L) \times \mathcal{R}$ and $A \subseteq \mathcal{P}(L) \times \mathcal{R}$. Then, the flattenings of a and A are defined as $\widehat{a} = \{(l, \boldsymbol{t}) \mid l \in \pi\}$ and $\widehat{A} = \bigcup_{a \in A} \widehat{a}$. Recall that Π denotes the most fine-grained partition. With these definitions, we can state the central soundness lemma.

Lemma 2. *Let $\mathcal{G}$ be a TGA with locations L and Π be a partition of L. Then,* $\mathsf{Attr}(\lfloor \mathcal{G} \rfloor_\Pi) \subseteq \mathsf{Attr}(\lfloor \mathcal{G} \rfloor_\Pi) = \mathsf{Attr}(\mathcal{G}) = \mathsf{Attr}(\lceil \mathcal{G} \rceil_\Pi) \subseteq \mathsf{Attr}(\lceil \mathcal{G} \rceil_\Pi)$.

On the one hand, Lemma 2 guarantees the soundness of our abstractions: once an abstract state $(\pi, \boldsymbol{t})$ appears in the attractor under-approximation, every

subsumed concrete state $(l, \boldsymbol{t})$, for any $l \in \pi$, is surely winning for player $\Diamond$. Dually, once an abstract state $(\pi', \boldsymbol{t}')$ is not contained in the attractor over-approximation, every subsumed concrete state $(l', \boldsymbol{t}')$, for any $l' \in \pi'$, is surely winning for player $\Box$. On the other hand, the lemma ensures that every refinement process eventually ends up with the precise attractor (e.g., when $\Pi = \Pi$).

Soundness of Zeno Abstractions. Location-based abstractions of timed systems may contain zeno loops giving rise to the existence of physically unmeaningful, and therefore spuriously too powerful, winning strategies. Note that this does not affect the soundness of our approach: there can only be zeno loops in an over-approximating control structure since we require the original system to be strongly nonzeno. Then, giving spuriously more power to an over-approximated player $\Box$ is consistent with the abstraction. On the other hand, giving zeno moves to an over-approximated player $\Diamond$ does not increase her winning possibilities since no moves leading to goal states are added.

We do not require a special treatment of zeno behavior in the backend solving algorithm; we only expect that, for zeno inputs, the algorithm reports sound (though zeno) strategies (which is the case for [10]).

3.2 Constructing Abstractions Using Boolean Functions

The key motivation for considering location-based abstractions is the possibility to use BFs for the construction of the abstract control structure. In this section, we describe an algorithm that constructs abstract (both weakened and strengthened) TGAs from a concrete TGA $\mathcal{G} = (L, I, \Delta, X, G)$ and a location partition Π. Note that we only use BFs for the *construction* of abstract TGAs but not for the actual game solving: abstract TGAs are represented using the standard (explicit location) representation [10].

In a preparation step, we encode Δ as a BF. The set of BF variables $\mathcal{B}$ that we use for our symbolic encoding consists of three disjoint sets $\mathcal{B}_L$, $\mathcal{B}_{L'}$, and $\mathcal{B}_X$, where $\mathcal{B}_L$ and $\mathcal{B}_{L'}$ represent predecessor- and successor-locations of timed transitions (with $|\mathcal{B}_L| = |\mathcal{B}_{L'}| = \lceil \log_2 |L| \rceil$). Furthermore, $\mathcal{B}_X$ is a set containing one variable v_x for each clock x (for encoding resets in the transition relation) and one variable v_φ for each atomic constraint $\varphi = x \bowtie c$ in Δ (for encoding guards).

We formalize these encodings in the following *predicates* over $\mathcal{B}$. First of all, for each location $l \in L$, the predicate $\ulcorner l \urcorner$ over $\mathcal{B}_L$ encodes l in binary form, and similarly, the predicate $\ulcorner l \urcorner'$ over $\mathcal{B}_{L'}$ encodes the primed version of l as a successor location. Formally, $\ulcorner l \urcorner$ and $\ulcorner l \urcorner'$ are functions mapping an assignment to the variables in $\mathcal{B}_L$ and $\mathcal{B}_{L'}$ (respectively) to **true** iff the assignment corresponds to the location l. As long as the binary encoding of the locations guarantees that the locations add up to **true** and are disjoint ($\bigvee_{l \in L} \ulcorner l \urcorner = \bigvee_{l \in L} \ulcorner l \urcorner' = \textbf{true}$ and for all locations $l_1, l_2 \in L$, $\ulcorner l_1 \urcorner \wedge \ulcorner l_2 \urcorner \equiv \textbf{false}$ and $\ulcorner l_1 \urcorner' \wedge \ulcorner l_2 \urcorner' \equiv \textbf{false}$ whenever $l_1 \neq l_2$), the details of the encoding are not important and are therefore not discussed here. We refer to Sect. 5 for further details. Additionally, we define $\ulcorner \varphi \urcorner = v_\varphi$ for all atomic constraints φ appearing in Δ, $\ulcorner \varphi \urcorner = \bigwedge_{i=1}^{n} \ulcorner \varphi_i \urcorner$ for all

Algorithm 1. BF-based construction of the transition relations of $\lfloor \mathcal{G} \rfloor_{\Pi}$ and $\lceil \mathcal{G} \rceil_{\Pi}$, for a given TGA $\mathcal{G}$ and a location partition Π.

1: **for all** $p \in \{\Box, \Diamond\}$ **do**
2: **for all** $(\pi, \pi') \in \Pi \times \Pi$ s.t. $A \equiv \pi \wedge \Delta_p \wedge \pi' \not\equiv$ **false do**
3: **for all** $\varphi \in \mathcal{C}(X)$ and $\lambda \subseteq X$ s.t. $B \equiv A \wedge \varphi \wedge \lambda \not\equiv$ **false do**
4: add $\langle \pi, \varphi, \lambda, \pi' \rangle$ to $\lceil \Delta_p \rceil_{\Pi}$
5: **if** $\pi \Rightarrow (\exists \mathcal{B}_{L'} \cup \mathcal{B}_X . B) \equiv$ **true then**
6: add $\langle \pi, \varphi, \lambda, \pi' \rangle$ to $\lfloor \Delta_p \rfloor_{\Pi}$

nonatomic constraints $\varphi = \varphi_1 \wedge \ldots \wedge \varphi_n$ and $\lambda = \bigwedge_{x \in \lambda} v_x \wedge \bigwedge_{x \in X \setminus \lambda} \neg v_x$ for all resets $\lambda \subseteq X$ appearing in Δ. These predicates are used to encode the guards of a transition and the respective resets in the transition relation.

For a set of locations $\pi \subseteq L$, we write π for $\bigwedge_{l \in \pi} l$. The Boolean predicate that symbolically represents the concrete transition relation for a player $p \in \{\Box, \Diamond\}$ can be defined as $\Delta_p \equiv \bigvee_{\langle s, \varphi, \lambda, t \rangle \in \Delta_p} s \wedge \varphi \wedge \lambda \wedge t'$. Note that the extension of this definition by, e.g., an action-based synchronization of distributed components or discrete integer variables used in guards and update expressions is straightforward. However, for the sake of simplicity of our presentation, we stick to the minimalistic, monolithic setting, although our prototype implementation described in Sect. 5 supports these features. Then, for a network of timed automata, by building the transition relation for each automaton separately, explicitly enumerating all locations in the product automaton is avoided.

Finally, Algo. 1 describes the construction of the transition relations for the abstract TGAs $\lfloor \mathcal{G} \rfloor_{\Pi}$ and $\lceil \mathcal{G} \rceil_{\Pi}$ from the concrete TGA $\mathcal{G}$ and a partition Π. In the first two lines, the algorithm iterates over the players and all potential connections (π, π'), which are represented as the BF A. In line 3, we iterate over all combinations of guards φ and resets λ whose corresponding predicates satisfy A, and compute the BF B that represents all concrete transitions $(l, \varphi, \lambda, l')$, with $(l, l') \in \pi \times \pi'$. In line 4, the transition is added to the set of potentially available transitions. Then, in line 5, the algorithm tests if the transition is surely available, i.e., if it also needs to be added to the set of surely available transitions in line 6. It is easy to see that the abstract transition relations constructed by Algo. 1 satisfy the definition from Sect. 3.1.

Note that the iterations in lines 2 and 3 do *not* necessarily induce a global explicit blow-up, as we can use the following optimizations:

1. We use the algorithm only to *update* the abstract TGAs in an incremental way during the refinement process. This way, we only need to consider the abstract locations modified by the respective last refinement step in line 2.
2. According to Lemma 1, if two abstract locations $\pi_1 \in \Pi$ and $\pi_2 \in \Pi$ are not connected in $\lceil \mathcal{G} \rceil_{\Pi}$, we can safely assume that any pair of refined abstract locations $\pi_1' \subseteq \pi_1$ or $\pi_2' \subseteq \pi_2$ is also not connected in $\lceil \mathcal{G} \rceil_{\Pi'}$, where $\Pi \prec \Pi'$, $\pi_1' \in \Pi'$, and $\pi_2' \in \Pi'$.
3. In line 3, assuming that we use a BDD to represent the BF A, we can inspect the BDD structure of A to skip guard/reset combinations that do not occur for the chosen abstract states π and π'.

4 Approximation-Guided Abstraction Refinement

In an abstraction refinement loop, we incrementally solve a sequence of abstract games with increasing precision converging to the original game. In Sect. 4.1, we first describe how to obtain sets of concrete locations that serve as interpolants for refining abstract locations. Then, Sect. 4.2 describes the actual refinement loop. Finally, Sect. 4.3 investigates optimizations.

4.1 Abstract Location Refinement

We give a general characterization of sets of concrete locations that can be used as interpolants for splitting abstract locations, i.e., location partitions in Π. All interpolants selected by any concrete refinement heuristic must satisfy this characterization. Due to the lack of space, we only describe the refinement for *enlarging* attractor under-approximations; shrinking attractor over-approximations is just the dual case and can be done analogously.

Definition 1. *Let $\mathcal{G} = (L, I, \Delta, X, G)$ be a TGA, Π be a partition of L, and $\lfloor \mathsf{A} \rfloor = \mathsf{Attr}(\lfloor \mathcal{G} \rfloor_\Pi)$. A set of concrete locations $R \subseteq L$ is defined to be an* effective interpolant *if and only if there is at least one $\pi \in \Pi$ with $\emptyset \subsetneq \pi \cap R \subsetneq \pi$ such that either*

(1) there is at least one transition $\langle \pi, \varphi, \lambda, \pi' \rangle \in \lceil \Delta_\Diamond \rceil_\Pi \setminus \lfloor \Delta_\Diamond \rfloor_\Pi$ such that

$$\forall l \in \pi \cap R : \exists l' \in \pi' : \langle l, \varphi, \lambda, l' \rangle \in \Delta_\Diamond \quad \textit{and}$$
$$\exists \boldsymbol{t} \in [\![\varphi]\!] : (\pi, \boldsymbol{t}) \notin \lfloor \mathsf{A} \rfloor \ \wedge \ (\pi', \boldsymbol{t}[\lambda := 0]) \in \lfloor \mathsf{A} \rfloor, \quad \textit{or}$$

(2) there is at least one transition $\langle \pi, \varphi, \lambda, \pi' \rangle \in \lceil \Delta_\Box \rceil_\Pi \setminus \lfloor \Delta_\Box \rfloor_\Pi$ such that

$$\forall l \in \pi : (\exists l' \in \pi' : \langle l, \varphi, \lambda, l' \rangle \in \Delta_\Box) \Rightarrow l \in R \quad \textit{and}$$
$$\exists \boldsymbol{t} \in [\![\varphi]\!] : (\pi', \boldsymbol{t}[\lambda := 0]) \notin \lfloor \mathsf{A} \rfloor.$$

In other words, an effective interpolant R refines some abstract locations whose transitions are either spuriously too weak for player $\Diamond$ or spuriously too powerful for player $\Box$. More precisely, guided by an attractor under-approximation, R is defined based on transitions whose appearance generates winning $\Diamond$-moves or whose disappearance removes spoiling $\Box$-moves. There always exists an effective interpolant unless the abstraction is most precise:

Lemma 3. *If $\mathsf{Attr}(\lfloor \mathcal{G} \rfloor_\Pi) \subsetneq \mathsf{Attr}(\mathcal{G})$, then there exists an effective interpolant.*

Refinements with effective interpolants always ensure progress:

Lemma 4. *If $R \subseteq L$ is an effective interpolant, then $\Pi \prec \Pi|R$.*

Refinements leading to an increase of precision are based on effective interpolants:

Lemma 5. *Let $R \subseteq L$, $\lfloor \mathsf{A} \rfloor = \mathsf{Attr}(\lfloor \mathcal{G} \rfloor_\Pi)$, and $\lfloor \mathsf{A}' \rfloor = \mathsf{Attr}(\lfloor \mathcal{G} \rfloor_{\Pi|R})$, where $\lfloor \mathsf{A} \rfloor$ is the starting point for computing $\lfloor \mathsf{A}' \rfloor$.*
If $\lfloor \mathsf{A} \rfloor \subsetneq \lfloor \mathsf{A}' \rfloor$, then R is an effective interpolant.

4.2 Refinement Loop

For a TGA $\mathcal{G} = (L, I, \Delta, X, G)$, we construct a finite sequence of location partitions of the form $\Pi_0 \prec \Pi_1 \prec \ldots \prec \Pi_n$ in a refinement loop, where n is a natural number and $\Pi_0 = \{L \setminus G, G\}$ is the trivial initial partition that separates non-goal locations from goal locations. We use the Boolean function-based technique from Sect. 3.2 to initially construct and incrementally update a sequence of abstract TGAs converging to $\mathcal{G}$. Note that, instead of constructing the complete abstract TGA in each refinement cycle, we incrementally update the previous one by letting Algo. 1 iterate only over those partitions that were affected by the previous refinement step. Each refinement step is guided by a *refinement heuristic* that determines an effective interpolant as defined in Sect. 4.1. More precisely, after each cycle i, for an effective interpolant $R_i \subseteq L$, we obtain the succeeding partition $\Pi_{i+1} = \Pi_i | R_i$.
We compute the initial and intermediate attractor approximations as follows:

$$\lfloor \mathsf{A}_0 \rfloor = \mathsf{Attr}(\lfloor \mathcal{G} \rfloor_{\Pi_0}) \qquad \text{and} \qquad \lfloor \mathsf{A}_{i+1} \rfloor = \lfloor \mathsf{A}_i \rfloor \cup \mathsf{Attr}(\lfloor \mathcal{G} \rfloor_{\Pi_{i+1}}) ;$$

$$\lceil \mathsf{A}_0 \rceil = \mathsf{Attr}(\lceil \mathcal{G} \rceil_{\Pi_0}) \qquad \text{and} \qquad \lceil \mathsf{A}_{i+1} \rceil = \lceil \mathsf{A}_i \rceil \cap \mathsf{Attr}(\lceil \mathcal{G} \rceil_{\Pi_{i+1}}) .$$

Hence, every maximal sequence of approximations is of the form

$$\lfloor \mathsf{A}_0 \rfloor \subseteq \cdots \subseteq \lfloor \mathsf{A}_n \rfloor = \mathsf{Attr}(\mathcal{G}) = \lceil \mathsf{A}_n \rceil \subseteq \cdots \subseteq \lceil \mathsf{A}_0 \rceil .$$

The loop terminates whenever the existence (nonexistence) of a winning strategy can be established in an under-approximation (over-approximation):
– $(I \times \{\mathbf{0}\}) \cap \lfloor \mathsf{A}_i \rfloor \neq \emptyset$, i.e., player $\Diamond$ surely has a winning strategy, or
– $(I \times \{\mathbf{0}\}) \cap \lceil \mathsf{A}_i \rceil = \emptyset$, i.e., player $\Box$ surely has a winning strategy.
Clearly, this suffices for termination, since if neither of the two conditions is satisfied, Lemma 3 guarantees that some further refinement is possible.

Theorem 2. *The presented abstraction refinement loop always terminates and yields a sound winning strategy for one of the players upon termination.*

4.3 Optimizations

Our abstraction refinement algorithm greatly benefits from several optimizations which can be applied *early* in the refinement loop. They are based on (1) pruning irrelevant moves that do not affect the winning capabilities of either player and (2) identifying surely winning states for player $\Diamond$ based on a strengthened TGA. For any abstract TGA $\mathcal{G} = (\Pi, I, \Delta, X, G)$ and its induced game structure $(S, S_0, \Gamma_\Box, \Gamma_\Diamond)$, with attractor under-approximation $\lfloor \mathsf{A} \rfloor$ and over-approximation $\lceil \mathsf{A} \rceil$, one can apply the following optimizations.

States already determined. We can remove all moves that lead out of states that are already known to be winning for some player. According to Lemma 2, once a state appears in an attractor under-approximation, it is surely winning for player $\Diamond$, and once a state is no more contained in an attractor over-approximation, it is surely winning for player $\Box$. Hence, it is safe to ignore all moves from $\{(s, s') \in \Gamma_\Box \cup \Gamma_\Diamond \mid s \in \lfloor \mathsf{A} \rfloor \vee s \notin \lceil \mathsf{A} \rceil\}$.

Moves already determined. We can remove all moves that lead to states that are already known to be winning for the opponent. Hence, it is safe to ignore all moves from $\{(s, s') \in \Gamma_\Box \mid s' \in \lfloor \mathsf{A} \rfloor\} \cup \{(s, s') \in \Gamma_\Diamond \mid s' \notin \lceil \mathsf{A} \rceil\}$.

States surely winning. Under the assumption that $\mathcal{G}$ is a strengthened TGA, an abstract state in S is surely winning for $\Diamond$ if *each* subsumed concrete state has *some* concrete move leading to $\lfloor \mathsf{A} \rfloor$. Hence, we can safely extend PreEnf by all states $(\pi, \boldsymbol{t}) \in \Pi \times \mathcal{R}$ where
$\pi \subseteq \{l \in L \mid \exists \pi' \in \Pi : \exists l' \in \pi' : (\pi', \boldsymbol{t}) \in \lfloor \mathsf{A} \rfloor \wedge ((l, \boldsymbol{t}), (l', \boldsymbol{t})) \in \Gamma_\Diamond\}$.

The first rule can easily be realized in the backend solving algorithm, when computing $\lfloor \mathsf{A}_{i+1} \rfloor$ (or $\lceil \mathsf{A}_{i+1} \rceil$), by not forward-exploring moves whose source states are already contained in $\lfloor \mathsf{A}_i \rfloor$ (or not contained in $\lceil \mathsf{A}_i \rceil$). The second rule is realized by reusing $\lfloor \mathsf{A}_i \rfloor$ as a starting point for $\lfloor \mathsf{A}_{i+1} \rfloor$ (and $\lceil \mathsf{A}_i \rceil$ for $\lceil \mathsf{A}_{i+1} \rceil$). The third rule is used to extend the results of PreEnf when constructing $\lfloor \mathsf{A}_i \rfloor$.

5 Experimental Results

5.1 Prototype Implementation

We implemented a prototype in C^{++}, where we combined the CUDD BDD library [18] for representing location partitions and the UPPAAL-DBM library [5] for representing federations of clock zones in the attractors.

In the initialization phase, our tool registers all BDD variables after calling the NOVA tool from the SIS toolset [17] for finding efficient assignments of control locations to BDD variable valuations. Then, as described in Sect. 3.2, we construct the symbolic discrete transition relation representing the control structure of the input network of TGAs. Note that, although not discussed in detail in the rest of the paper, in general our approach (and in particular our tool) is able to handle networks of communicating TGAs with integer variables: such pure discrete features are covered in the construction of the discrete transition relation. In the next initialization step, we use the discrete transition relation to compute an over-approximation of the reachable locations in a (cheap) BDD-based least fixed point computation. The initial partition splits this over-approximation into (1) the set of potentially reachable goal locations, (2) the set of potentially reachable locations from which no goal location is reachable, and (3) the remaining locations. At the end of the initialization phase, we use Algo. 1 to construct the initial weakened and strengthened TGAs, where we merge transitions with the same resets whose guards subsume each other.

In the automatic abstraction refinement loop, we use our implementation of the backend solving algorithm proposed in [10] to incrementally update an attractor under-approximation. After each iteration, we check if the concrete initial state is contained in the abstract attractor. In this case, we terminate since we can deduce that player $\Diamond$ surely wins. If this is not the case, we identify abstract transitions which are spuriously too weak for player $\Diamond$ and symbolically compute corresponding effective interpolants (by applying the BDD-based preimage operator). If there are no abstract transitions for player $\Diamond$, we identify

abstract transitions which are spuriously too powerful for player □ and refine likewise. Then, we split the partition with each computed interpolant (by simple BDD-based conjunctions) and update the weakened TGA using Algo. 1. Each refinement step might split a single abstract location by multiple interpolants resulting in an exponential number of split operations. To address this issue, we fix a number $\mathcal{K}$ of maximal split operations per abstract location.

5.2 Benchmarks

We evaluated our approach on two standard benchmarks[2] for timed controller synthesis and compared the results with UPPAAL-TIGA [4] version 4.1.3-0.14.

The *Production Cell* (Prodcell) example [14,10] represents a manufacturing plant consisting of a feeding belt, two robot arms, a press, and a departure belt. The timed game comes into play when synthesizing a controller for the robot arms such that all parts put onto the feeding belt are transported to the press right in time and are finally transported to the departure belt.

The *Gear Production Stack* (GPS) example [16] models a pipeline-like architecture that sequentializes a series of stations, each specialized in a certain processing method. The task is to synthesize a controller for the machine that ensures that the pieces are transported from station to station right in time. We investigate the nonextended version without sub-processing units.

Table 1 shows the results of our comparison where we fixed $\mathcal{K} = 1000$. From left to right, the first two columns describe the name of the benchmark, the length (in number of plates and stations, resp.), and whether there exists a controller implementation (i.e., a winning strategy for player □). The next three columns show the number of explored states, the running time, and the memory consumption of UPPAAL-TIGA. The last four columns show the number of refinement steps, the final size (in number of locations) of the abstract TGA, the running time, and the memory consumption of our prototype. All benchmarks were executed on an AMD Opteron processor with 2.6 GHz and 4 GB RAM. The running times are given in seconds and the memory consumptions are given in MB. The time limit was set to four hours.

The most striking observation is that for both benchmarks, our approach almost always outperforms UPPAAL-TIGA. Only for small benchmark instances, UPPAAL-TIGA performs slightly better. This is due to the preprocessing phase where all BDD variables are registered and the symbolic discrete transition relation is constructed. However, for benchmark instances of nontrivial size, UPPAAL-TIGA either runs out of memory or needs at least an order of magnitude more running time than our tool.

The impact of different values for $\mathcal{K}$ on the running time and memory consumption is shown in Table 2. Smaller values for $\mathcal{K}$ result in a higher number of refinement steps but lead to a lower memory peak consumption since fewer split abstract locations have to be maintained during a single refinement step. If there is

[2] The UPPAAL-TIGA models of the benchmarks are available at `http://www.avacs.org/Benchmarks/Open/formats10.tgz`

Table 1. Comparison of UPPAAL-TIGA with our prototype

		UPPAAL-TIGA			Our prototype			
Benchmark	Cont	States	Time	Mem	Steps	Abs	Time	Mem
Prodcell 3	No	15241	1	54	14	293	3	94
Prodcell 4	No	131999	5	74	14	935	13	156
Prodcell 5	No	1238698	240	309	14	2762	39	244
Prodcell 6	No	TIMEOUT			14	8212	150	538
Prodcell 7	No	TIMEOUT			15	24757	761	1936
Prodcell 8	No	TIMEOUT			15	75085	6543	2092
Prodcell 3	Yes	15206	1	54	14	294	3	113
Prodcell 4	Yes	133181	5	75	15	940	11	156
Prodcell 5	Yes	1255498	238	314	15	2772	42	246
Prodcell 6	Yes	TIMEOUT			15	8232	172	538
Prodcell 7	Yes	TIMEOUT			16	24792	1068	1936
Prodcell 8	Yes	TIMEOUT			16	75140	6444	2093
GPS 6	No	170470	4	69	14	274	2	81
GPS 7	No	1406744	40	190	16	560	3	117
GPS 8	No	12123700	545	1327	18	1134	6	133
GPS 9	No	MEMOUT			20	2284	20	250
GPS 10	No	MEMOUT			23	5518	91	402
GPS 11	No	MEMOUT			25	11128	307	948
GPS 12	No	MEMOUT			27	22368	1553	3550
GPS 6	Yes	190484	4	69	17	320	2	81
GPS 7	Yes	1647955	48	207	20	704	3	118
GPS 8	Yes	15187763	712	1551	23	1536	9	133
GPS 9	Yes	MEMOUT			26	3328	35	223
GPS 10	Yes	MEMOUT			29	7168	131	402
GPS 11	Yes	MEMOUT			32	15360	461	948
GPS 12	Yes	MEMOUT			35	32768	2207	3550

Table 2. Comparison of different values for $\mathcal{K}$

Benchmark	Cont	$\mathcal{K}$	Steps	Abs	Time	Mem
GPS 12	No	50	28	16187	973	661
GPS 12	No	100	27	16215	1047	711
GPS 12	No	200	27	17974	1254	1201
GPS 12	No	300	27	20236	1341	2237
GPS 12	No	500	27	21859	1970	2893
GPS 12	No	1000	27	22368	1553	3550
GPS 12	No	2000	27	22368	1399	3454
GPS 12	No	5000	MEMOUT			
GPS 12	Yes	50	358	32768	13872	1947
GPS 12	Yes	100	190	32768	9041	1517
GPS 12	Yes	200	73	32768	4774	1585
GPS 12	Yes	300	44	32768	3167	2621
GPS 12	Yes	500	35	32768	2962	3277
GPS 12	Yes	1000	35	32768	2207	3550
GPS 12	Yes	2000	35	32768	1813	3454
GPS 12	Yes	5000	MEMOUT			

a player $\Diamond$ winning strategy (Cont=No), more states can be pruned due to a more fine-grained refinement process. Consequently, the effect of pruning is weaker if there is no player $\Diamond$ winning strategy (Cont=Yes). On the other hand, higher values for $\mathcal{K}$ result in a lower number of refinement steps but require more memory for a single refinement step. The decrease in the running times results from fewer calls of the backend solving algorithm which reuses the attractor under-approximation from the last call but has to recompute the reachable states.

Acknowledgment. This work was supported by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS). The authors want to thank Christoph Scholl for pointing out the SIS toolset [17] for finding efficient assignments of control locations to BDD variable valuations, and the anynomous reviewers for their helpful comments.

References

1. Altisen, K., Tripakis, S.: Tools for controller synthesis of timed systems. In: 2nd Workshop on Real-Time Tools, RT-TOOLS (2002)
2. Alur, R., Dill, D.L.: A theory of timed automata. Theo. Comp. Sci. 126(2) (1994)
3. Asarin, E., Maler, O., Pnueli, A., Sifakis, J.: Controller synthesis for timed automata. In: Proc. 5th IFAC Conference on System Structure and Control (1998)
4. Behrmann, G., Cougnard, A., David, A., Fleury, E., Larsen, K.G., Lime, D.: UPPAAL-Tiga: Time for playing games! In: Damm, W., Hermanns, H. (eds.) CAV 2007. LNCS, vol. 4590, pp. 121–125. Springer, Heidelberg (2007)
5. Bengtsson, J.: Clocks, DBM, and States in Timed Systems. PhD thesis, Uppsala University (2002)
6. Brückner, I., Dräger, K., Finkbeiner, B., Wehrheim, H.: Slicing abstractions. In: vol. 89, pp. 369–392. IOS Press, Amsterdam (2008)
7. Bryant, R.E.: Graph-based algorithms for boolean function manipulation. IEEE Trans. Computers 35(8), 677–691 (1986)
8. Bulychev, P., Chatain, T., David, A., Larsen, K.G.: Efficient on-the-fly algorithm for checking alternating timed simulation. In: Ouaknine, J., Vaandrager, F.W. (eds.) FORMATS 2009. LNCS, vol. 5813, pp. 73–87. Springer, Heidelberg (2009)
9. Burch, J.R., Clarke, E.M., McMillan, K.L., Dill, D.L., Hwang, L.J.: Symbolic model checking: 10^{20} states and beyond. Inf. Comput. 98(2) (1992)
10. Cassez, F., David, A., Fleury, E., Larsen, K.G., Lime, D.: Efficient on-the-fly algorithms for the analysis of timed games. In: Abadi, M., de Alfaro, L. (eds.) CONCUR 2005. LNCS, vol. 3653, pp. 66–80. Springer, Heidelberg (2005)
11. de Alfaro, L., Roy, P.: Solving games via three-valued abstraction refinement. In: Caires, L., Vasconcelos, V.T. (eds.) CONCUR 2007. LNCS, vol. 4703, pp. 74–89. Springer, Heidelberg (2007)
12. Henzinger, T.A., Jhala, R., Majumdar, R.: Counterexample-guided control. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) ICALP 2003. LNCS, vol. 2719, Springer, Heidelberg (2003)
13. Henzinger, T.A., Kopke, P.W.: Discrete-time control for rectangular hybrid automata. Theoretical Computer Science 221(1-2), 369–392 (1999)
14. Lewerentz, C., Lindner, T. (eds.): Formal Development of Reactive Systems - Case Study Production Cell (1995)
15. Maler, O., Pnueli, A., Sifakis, J.: On the synthesis of discrete controllers for timed systems (an extended abstract). In: Mayr, E.W., Puech, C. (eds.) STACS 1995. LNCS, vol. 900, pp. 229–242. Springer, Heidelberg (1995)
16. Peter, H.J., Mattmüller, R.: Component-based abstraction refinement for timed controller synthesis. In: RTSS (2009)
17. Sentovich, E., Singh, K., Lavagno, L., Moon, C., Murgai, R., Saldanha, A., Savoj, H., Stephan, P., Brayton, R.K., Sangiovanni-Vincentelli, A.L.: SIS: A system for sequential circuit synthesis. Technical report, University of California (1992)
18. Somenzi, F.: CUDD: CU Decision Diagram package release 2.4.2 (2009)
19. Thomas, W.: On the synthesis of strategies in infinite games. In: Mayr, E.W., Puech, C. (eds.) STACS 1995. LNCS, vol. 900, Springer, Heidelberg (1995)

Expected Reachability-Time Games

Vojtěch Forejt[1], Marta Kwiatkowska[2], Gethin Norman[3], and Ashutosh Trivedi[2]

[1] Faculty of Informatics, Masaryk University, Botanická 68a, Brno, Czech Republic
[2] Oxford University Computing Laboratory, Parks Road, Oxford, OX1 3QD, UK
[3] Department of Computing Science, University of Glasgow, Glasgow, G12 8RZ, UK

Abstract. In an expected reachability-time game (ERTG) two players, Min and Max, move a token along the transitions of a probabilistic timed automaton, so as to minimise and maximise, respectively, the expected time to reach a target. These games are concurrent since at each step of the game both players choose a timed move (a time delay and action under their control), and the transition of the game is determined by the timed move of the player who proposes the shorter delay. A game is turn-based if at any step of the game, all available actions are under the control of precisely one player. We show that while concurrent ERTGs are not always determined, turn-based ERTGs are positionally determined. Using the boundary region graph abstraction, and a generalisation of Asarin and Maler's simple function, we show that the decision problems related to computing the upper/lower values of concurrent ERTGs, and computing the value of turn-based ERTGs are decidable and their complexity is in NEXPTIME ∩ co-NEXPTIME.

1 Introduction

Two-player zero-sum games on finite automata, as a mechanism for supervisory controller synthesis of discrete event systems, were introduced by Ramadge and Wonham [1]. In this setting the two players—called Min and Max—represent the *controller* and the *environment*, and control-program synthesis corresponds to finding a winning (or optimal) strategy of the controller for some given performance objective. If the objectives are dependent on time, e.g. when the objective corresponds to completing a given set of tasks within some deadline, then games on timed automata are a well-established approach for controller synthesis, see e.g. [2,3,4,5,6].

In this paper we extend this approach to objectives that are quantitative both in terms of timed *and* probabilistic behaviour. Probabilistic behaviour is important in modelling, e.g., faulty or unreliable components, the random coin flips of distributed communication and security protocols, and performance characteristics. We consider games on probabilistic timed automata (PTAs) [7,8,9], a model for real-time systems exhibiting nondeterministic and probabilistic behaviour. We concentrate on *expected reachability-time games* (ERTGs), which are games on PTAs where the performance objective concerns the minimum expected time the controller can ensure for the system to reach a target, regardless of the uncontrollable (environmental) events that occur. This approach has many practical applications, e.g., in job-shop scheduling, where machines can be faulty or have variable execution time, and both routing and task graph scheduling problems. For real-life examples relevant to our setting, see e.g. [10,6].

K. Chatterjee and T.A. Henzinger (Eds.): FORMATS 2010, LNCS 6246, pp. 122–136, 2010.

In the games that we study, a token is placed on a configuration of a PTA and a play of the game corresponds to both players proposing a timed move of the PTA, i.e. a time delay and action under their control (we assume each action of the PTA is under the control of precisely one of the players). Once the players have made their choices, the timed move with the shorter delay[1] is performed and the token is moved according to the probabilistic transition function of the PTA. Players Min and Max choose their moves in order to minimise and maximise, respectively, the payoff function (the time till the first visit of a target in the case of ERTGs). It is well known, see, e.g. [11], that concurrent timed games are not *determined*, which means the *upper value* of the game (the minimum expected time to reach a target that Min can ensure) is strictly greater that the *lower value* of the game (the maximum expected time to reach a target that Max can ensure). A game is determined if the lower and upper values are equal, and in this case, the *optimal value* of the game exists and equals the upper and lower values. We show that a subclass of ERTGs, called *turn-based* ERTGs, where at each step of the game only one of the players has available actions are *positionally determined*, i.e. both players have ε-optimal (optimal up to a given precision $\varepsilon>0$) positional (history-independent and non-randomised) strategies.

The problem we consider is inspired by Asarin and Maler [2] who studied the *brachystochronic problem* for timed automata. This work focused on reachability-time games, i.e. games on a timed automata where the objective concerns the time to reach a target. The techniques of [2] exploit properties of a special class of functions called *simple functions*. The importance of simple functions is also observed in [12] in the context of one-player games. Simple functions have also enabled the computation of a uniform solution for (turn-based) reachability-time games [13] and the proof of correctness of game-reduction for turn-based average-time games [14]. However, we show that the concept of simple functions is not sufficient in the setting of PTAs.

Contribution. We show that the problem of deciding whether the upper (lower, or the optimal when it exists) value of an ERTG is at most a given bound is decidable. An important contribution of the paper is the generalisation of simple functions to *quasi-simple functions*. By using this class of functions and the boundary region abstraction [15,16], we give a novel proof of positional determinacy of *turn-based* ERTGs. We demonstrate that the problem of finding the upper and lower value of general ERTGs is in NEXPTIME $\cap$ co-NEXPTIME. An EXPTIME-hardness lower bound follows from the EXPTIME-completeness of the corresponding optimisation problem [16]. From [17] it follows that the problem is not NEXPTIME-hard, unless NP equals co-NP. Extending this work we get the similar results for expected discounted-time games.

Related Work. Hoffman and Wong-Toi [18] were the first to define and solve optimal controller synthesis problem for timed automata. For a detailed introduction to the topic of qualitative games on timed automata, see e.g. [19]. Asarin and Maler [2] initiated the study of quantitative games on timed automata by providing a symbolic algorithm to solve reachability-time games. The work of [20] and [13] show that the decision version of the reachability-time game is EXPTIME-complete for timed automata with

[1] Min and Max represent two different forms of non-determinism called *angelic* and *demonic*. To prevent the introduction of a third form, we assume the move of Max (the environment) is taken if the delays are equal. The converse can be used without changing the presented results.

at least two clocks. The tool UPPAAL Tiga [5] is capable of solving reachability and safety objectives for games on timed automata. Jurdziński and Trivedi [14] show the EXPTIME-completeness for average-time games on automata with two or more clocks.

A natural extension of reachability-time games are games on priced timed automata where the objective concerns the cumulated price of reaching a target. Both [3] and [4] present semi-algorithms for computing the value of such games for linear prices. In [21] the problem of checking the existence of optimal strategies is shown to be undecidable with [22] showing undecidability holds even for three clocks and stopwatch prices.

We are not aware of any previous work studying two-player quantitative games on PTAs. For a significantly different model of stochastic timed games, deciding whether a target is reachable within a given bound is undecidable [23]. Regarding one-player games on PTAs, [16] reduce a number of optimisation problems on concavely-priced PTAs to solving the corresponding problems on the boundary region abstraction and [24] solve expected reachability-price problems for linearly-priced PTAs using digital clocks. In [25] the problem of deciding whether a target can be reached within a given price and probability bound is shown to be undecidable for priced PTAs with three clocks and stopwatch prices. By a simple modification of the proofs in [25] it can be demonstrated that checking the existence of optimal strategies is undecidable for reachability-price turn-based games on priced (probabilistic) timed automata with three clocks and stopwatch prices.

A full version of this paper, including proofs is also available [26], while a preliminary version appeared as [27].

2 Expected Reachability Games

Expected reachability games (ERGs) are played between two players Min and Max on a state-transition graph, whose transitions are nondeterministic and probabilistic, by jointly resolving the nondeterminism to move a token along the transitions of the graph. The objective for player Min in the game is to reach the final states with the smallest accumulated reward, while Max tries to do the opposite.

Before we give a formal definition, we need to introduce the concept of discrete probability distributions. A *discrete distribution* over a (possibly uncountable) set Q is a function $d : Q \to [0,1]$ such that $supp(d) = \{q \in Q \mid d(q) > 0\}$ is at most countable and $\sum_{q \in Q} d(q) = 1$. Let $\mathcal{D}(Q')$ denote the set of all discrete distributions over Q. We say a distribution $d \in \mathcal{D}(Q)$ is a *point distribution* if $d(q) = 1$ for some $q \in Q$.

Definition 1. *An ERG is a tuple* $\mathsf{G} = (S, F, A_{\mathrm{Min}}, A_{\mathrm{Max}}, p_{\mathrm{Min}}, p_{\mathrm{Max}}, \pi_{\mathrm{Min}}, \pi_{\mathrm{Max}})$ *where:*

- S *is a (possibly uncountable) set of* states *including a set of* final states F*;*
- A_{Min} *and* A_{Max} *are (possibly uncountable) sets of actions* controlled *by players Min and Max and* $\bot$ *is a distinguished action such that* $A_{\mathrm{Min}} \cap A_{\mathrm{Max}} = \{\bot\}$*;*
- $p_{\mathrm{Min}} : S \times A_{\mathrm{Min}} \to \mathcal{D}(S)$ *and* $p_{\mathrm{Max}} : S \times A_{\mathrm{Max}} \to \mathcal{D}(S)$ *are the partial probabilistic transition functions for players Min and Max such that* $p_{\mathrm{Min}}(s, \bot)$ *and* $p_{\mathrm{Max}}(s, \bot)$ *are undefined for all* $s \in S$*;*
- $\pi_{\mathrm{Min}} : S \times A_{\mathrm{Min}} \to \mathbb{R}_{\geq 0}$ *and* $\pi_{\mathrm{Max}} : S \times A_{\mathrm{Max}} \to \mathbb{R}_{\geq 0}$ *are the reward functions for players Min and Max.*

We say that the ERG is *finite* if both S and A are finite. For any state s, we let $A_{\mathrm{Min}}(s)$ denote the set of actions available to player Min in s, i.e., the actions $a \in A_{\mathrm{Min}}$ for which $p_{\mathrm{Min}}(s,a)$ is defined, letting $A_{\mathrm{Min}}(s)=\bot$ if no such action exists. Similarly, $A_{\mathrm{Max}}(s)$ denotes the actions available to player Max in s and we let $A(s)=A_{\mathrm{Min}}(s)\times A_{\mathrm{Max}}(s)$. We say that s is *controlled* by Min (Max) if $A_{\mathrm{Max}}(s)=\{\bot\}$ ($A_{\mathrm{Min}}(s)=\{\bot\}$) and the game G is *turn-based* if there is a partition $(S_{\mathrm{Min}}, S_{\mathrm{Max}})$ of S such that all states in S_{Min} (S_{Max}) are controlled by Min (Max).

A game G starts with a token in some initial state and players Min and Max construct an infinite play by repeatedly choosing enabled actions, and then moving the token to a successor state determined by their probabilistic transition functions where the reward of the move is determined by their reward functions. More precisely, if in state s players Min and Max choose actions a and b respectively, then if $\pi_{\mathrm{Min}}(s,a)<\pi_{\mathrm{Max}}(s,b)$ or $b=\bot$ the probabilistic transition function and reward value are determined by Min's choice, i.e. by the transition function $p_{\mathrm{Min}}(s,a)$ and reward value $\pi_{\mathrm{Min}}(s,a)$, and otherwise are determined by Max's choice. Formally we introduce the following auxiliary functions of an ERG which return the transition function and reward value of the game.

Definition 2. *Let* G *be an ERG. The* probabilistic transition *and* reward *functions* $p : S\times A_{\mathrm{Min}}\times A_{\mathrm{Max}}\to\mathcal{D}(S)$ *and* $\pi : S\times A_{\mathrm{Min}}\times A_{\mathrm{Max}}\to\mathbb{R}_{\geq 0}$ *of* G *are such that for any* $s \in S$ *and* $(a,b) \in A_{\mathrm{Min}}$*:*

$$p(s,a,b) = \begin{cases} \textit{undefined if } a=b=\bot \\ p_{\mathrm{Min}}(s,a) \textit{ if } a\neq\bot \textit{ and either } b=\bot \textit{ or } \pi_{\mathrm{Min}}(s,a)<\pi_{\mathrm{Max}}(s,b) \\ p_{\mathrm{Max}}(s,b) \textit{ otherwise} \end{cases}$$

$$\pi(s,a,b) = \begin{cases} \pi_{\mathrm{Min}}(s,a) \textit{ if } b=\bot \textit{ or } \pi_{\mathrm{Min}}(s,a)<\pi_{\mathrm{Max}}(s,b) \\ \pi_{\mathrm{Max}}(s,b) \textit{ otherwise.} \end{cases}$$

From the conditions imposed on the probabilistic transition function, it follows that $(a,b) \in A(s)$ if and only if $p(s,a,b)$ is defined. Using these definitions, if in state s the action pair $(a,b) \in A(s)$ is chosen, then the probability of making a transition to s' equals $p(s'|s,a,b)\stackrel{\text{def}}{=}p(s,a,b)(s')$ and the reward equals $\pi(s,a,b)$.

A transition of G is a tuple $(s,(a,b),s')$ such that $p(s'|s,a,b)>0$ and a play is an finite or infinite sequence $\langle s_0,(a_1,b_1),s_1,\ldots\rangle$ such that $(s_i,(a_{i+1},b_{i+1}),s_{i+1})$ is a transition for all $i \geq 0$. For a finite play $\rho = \langle s_0,(a_1,b_1),s_1,\ldots,s_k\rangle$, let $\mathit{last}(\rho)$ denote the last state s_k of the play. We write Play ($\mathit{Play}_{\mathit{fin}}$) for the sets of (finite) plays in G and $\mathit{Play}(s)$ ($\mathit{Play}_{\mathit{fin}}(s)$) for the sets of (finite) plays starting from $s \in S$.

A *strategy* of Min is a function $\mu : \mathit{Play}_{\mathit{fin}}\to\mathcal{D}(A_{\mathrm{Min}})$ such that $\mathit{supp}(\mu(\rho)) \subseteq A_{\mathrm{Min}}(\mathit{last}(\rho))$ for all finite plays $\rho \in \mathit{Play}_{\mathit{fin}}$, i.e. for any finite play, a strategy returns a distribution over actions available to Min in the last state of the play. A strategy χ of Max is defined analogously and we let Σ_{Min} and Σ_{Max} denote the sets of strategies of Min and Max, respectively. A strategy σ is *pure* if $\sigma(\rho)$ is a point distribution for all $\rho \in \mathit{Play}_{\mathit{fin}}$, while it is *stationary* if $\mathit{last}(\rho)=\mathit{last}(\rho')$ implies $\sigma(\rho)=\sigma(\rho')$ for all $\rho,\rho' \in \mathit{Play}_{\mathit{fin}}$. A strategy is *positional* if it is pure and stationary and let Π_{Min} and Π_{Max} denote the set of positional strategies of Min and Max, respectively.

For any state s and strategy pair $(\mu,\chi) \in \Sigma_{\mathrm{Min}}\times\Sigma_{\mathrm{Max}}$, let $\mathit{Play}^{\mu,\chi}(s)$ denote the infinite plays in which Min and Max play according to μ and χ, respectively. Using

standard results from probability theory, see e.g., [28], we can construct a probability measure $Prob_s^{\mu,\chi}$ over the set $Play^{\mu,\chi}(s)$. Let X_i and Y_i denote the random variables corresponding to i^{th} state and action of a play (i.e., for play $\langle s_0,(a_1,b_1),s_1,\ldots\rangle$ we have $X_i{=}s_i$ and $Y_{i+1}{=}(a_{i+1},b_{i+1})$), and given a *real-valued random variable* $f : Play \to \mathbb{R}$, let $\mathbb{E}_s^{\mu,\chi}\{f\}$ denote the expected value of f with respect to the probability measure $Prob_s^{\mu,\chi}$. To keep the presentation simple, for the rest of the paper we only consider *transient stochastic games* [29, Chapter 4] (games where every play is finite with probability 1) and for this reason we make the following assumption[2].

Assumption 1. For any strategy pair $(\mu,\chi) \in \Sigma_{\text{Min}}\times\Sigma_{\text{Max}}$, and state $s \in S$ there is $q > 0$ and $n \in \mathbb{N}$ such that $Prob_s^{\mu,\chi}(X_n \in F) \geq q$.

Recall that the objective for Min is to reach a final state with the smallest accumulated reward, while for Max it is the opposite. Starting from s, if Min uses the strategy μ and Max χ, then the expected reward accumulated before reaching a final state is given by:

$$\text{EReach}^{\mu,\chi}(s) \stackrel{\text{def}}{=} \mathbb{E}_s^{\mu,\chi}\left\{\textstyle\sum_{i=0}^{\min\{k-1 \mid X_k\in F\}}\pi(X_i,Y_{i+1})\right\}.$$

Observe when starting at state s, Max can choose actions such that the expected reward is *at least* a value arbitrarily close to $\sup_{\chi\in\Sigma_{\text{Max}}}\inf_{\mu\in\Sigma_{\text{Min}}}\text{EReach}^{\mu,\chi}(s)$. This is called the *lower value* $\text{Val}_*(s)$ of the game when starting at state s. For $\chi \in \Sigma_{\text{Max}}$ let $\text{Val}_\chi(s){=}\inf_{\mu\in\Sigma_{\text{Min}}}\text{EReach}^{\mu,\chi}(s)$. We say χ is *optimal* (ε-*optimal*), if $\text{Val}_\chi(s){=}\text{Val}_*(s)$ ($\text{Val}_\chi(s){\geq}\text{Val}_*(s){-}\varepsilon$) for all $s \in S$. Similarly, Min can make choices such that the expected reward is *at most* a value arbitrarily close to the *upper value* $\text{Val}^*(s){=}\inf_{\mu\in\Sigma_{\text{Min}}}\sup_{\chi\in\Sigma_{\text{Max}}}\text{EReach}^{\mu,\chi}(s)$. In addition, for $\mu \in \Sigma_{\text{Min}}$, we can define $\text{Val}^\mu(s)$ and say when μ is *optimal* or ε-*optimal*.

A game G is *determined* if $\text{Val}_*(s){=}\text{Val}^*(s)$ for all $s \in S$ and then we say that the value of the game exists and equals $\text{Val}(s){=}\text{Val}_*(s){=}\text{Val}^*(s)$. If G is determined, then each player has an ε-optimal strategy for all $\varepsilon{>}0$. A game is *positionally determined* if

$$\text{Val}(s){=}\inf_{\mu\in\Pi_{\text{Min}}}\sup_{\chi\in\Sigma_{\text{Max}}}\text{EReach}^{\mu,\chi}(s) = \sup_{\chi\in\Pi_{\text{Max}}}\inf_{\mu\in\Sigma_{\text{Min}}}\text{EReach}^{\mu,\chi}(s)$$

for all $s \in S$. It is straightforward to see that if a game is positionally determined, then both players have *positional* ε-optimal strategies for all $\varepsilon{>}0$.

Optimality Equations. We complete this section by introducing optimality equations for ERGs. For a game G and function $P : S{\to}\mathbb{R}_{\geq 0}$, we say that P is a solution of the optimality equations $\text{Opt}^*(\mathsf{G})$, and write $P{\models}\text{Opt}^*(\mathsf{G})$, if for any $s \in S$:

$$P(s) = \begin{cases} 0 & \text{if } s \in F \\ \inf\limits_{\alpha\in A_{\text{Min}}(s)}\left\{\sup\limits_{\beta\in A_{\text{Max}}(s)}\left\{\pi(s,\alpha,\beta)+\sum\limits_{s'\in S}p(s'|s,\alpha,\beta)\cdot P(s')\right\}\right\} & \text{if } s \notin F. \end{cases}$$

and P is a solution of the optimality equations $\text{Opt}_*(\mathsf{G})$, and write $P{\models}\text{Opt}_*(\mathsf{G})$, if for any $s \in S$:

$$P(s) = \begin{cases} 0 & \text{if } s \in F \\ \sup\limits_{\beta\in A_{\text{Max}}(s)}\left\{\inf\limits_{\alpha\in A_{\text{Min}}(s)}\left\{\pi(s,\alpha,\beta)+\sum\limits_{s'\in S}p(s'|s,\alpha,\beta)\cdot P(s')\right\}\right\} & \text{if } s \notin F. \end{cases}$$

[2] Techniques (see, e.g., *positive stochastic games* [29, Chapter 4]) for lifting such an assumption are orthogonal to the main idea presented in this paper.

The following result demonstrate the correspondence between these equations and the lower and upper values of the expected reachability game.

Proposition 3. *For any ERG* G *and bounded function* $P : S \to \mathbb{R}_{\geq 0}$:

- *if* $P \models \mathrm{Opt}^*(\mathsf{G})$*, then* $\mathrm{Val}^*(s) = P(s)$ *for all* $s \in S$ *and for any* $\varepsilon > 0$ *player Min has a* positional *strategy* μ_ε *such that* $\mathrm{Val}^{\mu_\varepsilon}(s) \leq P(s) + \varepsilon$ *for all* $s \in S$*;*
- *if* $P \models \mathrm{Opt}_*(\mathsf{G})$*, then* $\mathrm{Val}_*(s) = P(s)$ *for all* $s \in S$ *and for any* $\varepsilon > 0$ *player Max has a* positional *strategy* χ_ε *such that* $\mathrm{Val}_{\chi_\varepsilon}(s) \geq P(s) - \varepsilon$ *for all* $s \in S$*.*

If G is turn-based, then the equations $\mathrm{Opt}^*(\mathsf{G})$ and $\mathrm{Opt}_*(\mathsf{G})$ are the same and we write $\mathrm{Opt}(\mathsf{G})$ for these equations. The following is a direct consequence of Proposition 3.

Proposition 4. *If* G *is a turn-based,* $P : S \to \mathbb{R}_{\geq 0}$ *is a bounded and* $P \models \mathrm{Opt}(\mathsf{G})$*, then* $\mathrm{Val}(s) = P(s)$ *for all* $s \in S$ *and for any* $\varepsilon > 0$ *both players have* ε*-optimal strategies.*

3 Expected Reachability-Time Games

Expected reachability-time games (ERTGs) are played on the infinite graph of a probabilistic timed automaton where Min and Max choose their moves so that the expected time to reach a final state is minimised or maximised, respectively. Before defining ERTGs, we introduce the concept of clocks, constraints, regions, and zones.

Clocks. Let $\mathcal{C}$ be a finite set of *clocks*. A *clock valuation* on $\mathcal{C}$ is a function $\nu : \mathcal{C} \to \mathbb{R}_{\geq 0}$ and we write V for the set of clock valuations. Abusing notation, we also treat a valuation ν as a point in $\mathbb{R}^{|\mathcal{C}|}$. If $\nu \in V$ and $t \in \mathbb{R}_{\geq 0}$ then we write $\nu + t$ for the clock valuation defined by $(\nu + t)(c) = \nu(c) + t$ for all $c \in \mathcal{C}$. For $C \subseteq \mathcal{C}$, we write $\nu[C := 0]$ for the valuation where $\nu[C := 0](c)$ equals 0 if $c \in C$ and $\nu(c)$ otherwise. For $X \subseteq V$, we write $\overline{X}$ for the smallest closed set in V containing X. Although clocks are usually allowed to take arbitrary non-negative values, w.l.o.g [30] we assume that there is an upper bound K such that for every clock $c \in \mathcal{C}$ we have that $\nu(c) \leq K$.

Clock constraints. A *clock constraint* over $\mathcal{C}$ is a conjunction of *simple constraints* of the form $c \bowtie i$ or $c - c' \bowtie i$, where $c, c' \in \mathcal{C}$, $i \in \mathbb{N}$, $i \leq K$, and $\bowtie \in \{<, >, =, \leq, \geq\}$. For $\nu \in V$, let $\mathrm{SCC}(\nu)$ be the finite set of simple constraints which hold in ν.

Clock regions. A *clock region* is a maximal set $\zeta \subseteq V$ such that $\mathrm{SCC}(\nu) = \mathrm{SCC}(\nu')$ for all $\nu, \nu' \in \zeta$. We write $\mathcal{R}$ for the finite set of clock regions. Every clock region is an equivalence class of the indistinguishability-by-clock-constraints relation, and vice versa. We write $[\nu]$ for the clock region of ν and, if $\zeta = [\nu]$, write $\zeta[C := 0]$ for $[\nu[C := 0]]$.

Clock zones. A *clock zone* is a convex set of clock valuations, which is a union of a set of clock regions. We write $\mathcal{Z}$ for the set of clock zones. A set of clock valuations is a clock zone if and only if it is definable by a clock constraint. Observe that, for every clock zone W, the set $\overline{W}$ is also a clock zone.

We now introduce ERTGs which extend classical timed automata [31] with discrete distributions and a partition of the actions between two players Min and Max.

Definition 5 (ERTG Syntax). *A (concurrent) expected reachability-time game (ERTG) is a tuple* $\mathcal{T} = (L, L_F, \mathcal{C}, Inv, Act, Act_{\mathrm{Min}}, Act_{\mathrm{Max}}, E, \delta)$ *where*

- L *is a finite set of* locations *including a set of* final locations L_F*;*
- $\mathcal{C}$ *is a finite set of* clocks*;*

- $Inv : L \to \mathcal{Z}$ *is an* invariant condition*;*
- Act *is a finite set of* actions *and* $\{Act_{\mathrm{Min}}, Act_{\mathrm{Max}}\}$ *is a partition of* Act*;*
- $E : L \times Act \to \mathcal{Z}$ *is an* action enabledness function*;*
- $\delta : L \times Act \to \mathcal{D}(2^{\mathcal{C}} \times L)$ *is a* probabilistic transition function*.*

When we consider an ERTG as an input of an algorithm, its size is understood as the sum of the sizes of encodings of L, $\mathcal{C}$, Inv, Act, E, and δ. As usual [32], we assume that probabilities are expressed as ratios of two natural numbers, each written in binary.

An ERTG is *turn-based* if for each location ℓ, only one player has enabled actions, i.e. $E(\ell, a)=\emptyset$ for all $a \in Act_{\mathrm{Min}}$ or $a \in Act_{\mathrm{Max}}$. In this case, we write L_{Min} and L_{Max} for the set of locations where players Min and Max, respectively, have an enabled action. A *one-player ERTG* is a turn-based ERTG where one of the player does not control any location, i.e., either $L_{\mathrm{Min}}=\emptyset$ or $L_{\mathrm{Max}}=\emptyset$. A (non-probabilistic) *reachability-timed game* is an ERTG such that $\delta(\ell, a)$ is a point distribution for all $\ell \in L$ and $a \in Act$.

A *configuration* of an ERTG is a pair (ℓ, ν), where ℓ is a location and ν a clock valuation such that $\nu \in Inv(\ell)$. For any $t \in \mathbb{R}$, we let $(\ell, \nu)+t$ equal the configuration $(\ell, \nu+t)$. In a configuration (ℓ, ν), a timed action (time-action pair) (t, a) is available if and only if the invariant condition $Inv(\ell)$ is continuously satisfied while t time units elapse, and a is enabled (i.e. the enabling condition $E(\ell, a)$ is satisfied) after t time units have elapsed. Furthermore, if the timed action (t, a) is performed, then the next configuration is determined by the probabilistic transition relation δ, i.e. with probability $\delta[\ell, a](C, \ell')$ the clocks in C are reset and we move to the location ℓ'.

An ERTG starts at some initial configuration and Min and Max construct an infinite play by repeatedly choosing available timed actions $(t_a, a) \in \mathbb{R}_{\geq 0} \times Act_{\mathrm{Min}}$ and $(t_b, b) \in \mathbb{R}_{\geq 0} \times Act_{\mathrm{Max}}$ proposing $\perp$ if no timed action is available. The player responsible for the move is Min if the time delay of Min's choice is less than that of Max's choice or Max chooses $\perp$, and otherwise Max is responsible. We assume the players cannot simultaneously choose $\perp$. We now present the formal semantics which is an ERG with potentially infinite number of states and actions. It is straightforward to show the semantics of a turn-based ERTG is a turn-based ERG.

Definition 6 (ERTG Semantics). *Let* $\mathcal{T}$ *be an ERTG. The semantics of* $\mathcal{T}$ *is given the ERG* $[\![\mathcal{T}]\!]=(S, F, A_{\mathrm{Min}}, A_{\mathrm{Max}}, p_{\mathrm{Min}}, p_{\mathrm{Max}}, \pi_{\mathrm{Min}}, \pi_{\mathrm{Max}})$ *where*

- $S \subseteq L \times V$ *is the (possibly uncountable) set of states such that* $(\ell, \nu) \in S$ *if and only if* $\nu \in Inv(\ell)$ *and* $F = \{(\ell, \nu) \in S \mid \ell \in L_F\}$ *is the set of final states;*
- $A_{\mathrm{Min}} = (\mathbb{R}_{\geq 0} \times Act_{\mathrm{Min}}) \cup \{\perp\}$ *and* $A_{\mathrm{Max}} = (\mathbb{R}_{\geq 0} \times Act_{\mathrm{Max}}) \cup \{\perp\}$ *are the sets of timed actions of players Min and Max;*
- *for* $\star \in \{\mathrm{Min}, \mathrm{Max}\}$, $(\ell, \nu) \in S$ *and* $(t, a) \in A_\star$ *the probabilistic transition function* $p_\star$ *is defined when* $\nu+t' \in Inv(\ell)$ *for all* $t' \leq t$ *and* $\nu+t \in E(\ell, a)$ *and for any* (ℓ, ν')*:*

$$p_\star((\ell, \nu), (t, a))((\ell', \nu')) = \textstyle\sum_{C \subseteq \mathcal{C} \wedge (\nu+t)[C:=0]=\nu'} \delta[\ell, a](C, \ell');$$

- *for* $\star \in \{\mathrm{Min}, \mathrm{Max}\}$, $s \in S$ *and* $(t, a) \in A_{\mathrm{Min}}$ *the reward function* $\pi_\star$ *is given by* $\pi_\star(s, (t, a))=t$*.*

The sum in the definitions of p_{Min} and p_{Max} is used to capture the fact that resetting different subsets of $\mathcal{C}$ may result in the same clock valuation (e.g. if all clocks are

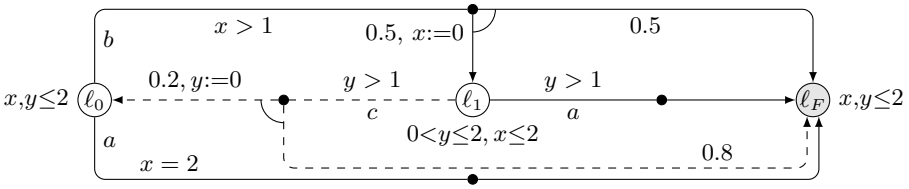


Fig. 1. An expected reachability-time game

initially zero, then we end up with the same valuation, no matter which clocks we reset). Also, notice that the reward function of the ERG corresponds to the elapsed time of each move. For any ERTG $\mathcal{T}$, to ensure Assumption 1 holds on the ERG $[\![\mathcal{T}]\!]$, we require only that the following weaker assumption holds on $[\![\mathcal{T}]\!]$.

Assumption 2. For any strategy pair $(\mu, \chi) \in \Sigma_{\mathrm{Min}} \times \Sigma_{\mathrm{Max}}$, and state $s \in S$ we have that $\lim_{n\to\infty} Prob_s^{\mu,\chi}(X_n \in F) = 1$.

Example 7. Consider the ERTG in Figure 1; we use solid and dashed lines to indicate actions controlled by Min and Max respectively. The shaded circle denotes the final location. Considering location ℓ_1, the invariant condition is $0{<}y{\leq}2 \wedge x{\leq}2$, actions a and c are enabled when $y{>}1$ and, if a is taken, we move to ℓ_F, while if c is taken, with probability 0.2 we move to ℓ_0 and reset y, and with probability 0.8 move to ℓ_F.

Starting in the configuration[3] $(\ell_0, (0, 0))$ and supposing Min's strategy is to choose $(1.1, b)$ (i.e., wait 1.1 time units before performing action b) in location ℓ_0 and then choose $(0.5, a)$ in location ℓ_1, while Max's strategy in location ℓ_1 is to choose $(0.2, c)$. One possible play under this strategy pair is $\langle(\ell_0{,}(0{,}0)),\ ((1.1{,}b), \bot),\ (\ell_1{,}(0{,}1.1)),\ ((0.5{,}a),\ (0.2{,}c)),\ (\ell_0{,}(0.2{,}0)),\ ((1.1{,}b), \bot),\ (\ell_F, (1.3{,}1.1))\rangle$ which has probability $0.5{\cdot}0.2{\cdot}0.5 = 0.05$ and time $1.1{+}0.2{+}1.1 = 2.4$ of reaching the final location. Using the optimality equations $\mathrm{Opt}^*(\mathsf{G})$ and $\mathrm{Opt}_*(\mathsf{G})$, we obtain upper and lower value in state $(\ell_0, (0, 0))$ of $\frac{10}{9}$ and 1, respectively. For details of the equations see [26].

Example 7 above demonstrates that in general expected reachability-time games are not determined. However, our results yield a novel proof of the following fundamental result for turn-based expected reachability-time games.

Theorem 8. *Turn-based ERTGs are positionally determined.*

Since the general ERTG are not determined, we study the following decision problem related to computing the upper-value of a configuration. All presented results also apply to the corresponding lower value problem, and the value problem, if the value exists.

Definition 9 (ERTG Decision Problem). *The decision problem for an ERTG $\mathcal{T}$, a state s of $[\![\mathcal{T}]\!]$, and a bound $T \in \mathbb{Q}$ is to decide whether* $\mathrm{Val}^*(s){\leq}T$.

We now present the second fundamental result of the paper.

Theorem 10. *The ERTG decision problem is in NEXPTIME ∩ co-NEXPTIME.*

From [16] we know that the ERTG problem is EXPTIME-complete even for one player ERTGs with two or more clocks. Hence the ERTG problem for general (two-player,

[3] We suppose the first (second) coordinate in a clock valuation correspond to the clock x (y).

concurrent) ERTG is at least EXPTIME-hard. Moreover, from the results of [17] and [33] it follows that ERTG problem is not NEXPTIME-hard, unless NP = co-NP.

4 Proofs of Theorems 8 and 10

This section is dedicated to the correctness of Theorems 8 and 10. We begin by defining *boundary region abstraction* (BRA) (an instance of an ERG) of an ERTG. In Section 4.2 we show that the solution of the optimality equations for a BRA always exists and is unique. While Section 4.3 demonstrates (Theorem 15) that the solution of the optimality equations of the BRA can be used to construct a solution of the optimality equations of the ERTG. Using these results we can then prove our main results.

Proof outline of Theorem 8. Using Theorem 15, a boudned solution of the equations for the upper and lower values of a ERTG always exists, and hence Proposition 3 implies both players have positional ε-optimal strategies. Since for turn-based ERTGs both equations are equivalent, from Proposition 4 positional determinacy of turn-based ERTGs follows.

Proof outline of Theorem 10. From Theorem 15 the upper value of a state of a ERTG can be derived from that of the boundary region abstraction. Since in the BRA the sub-graph of reachable states from any state is finite (Lemma 12) and its size is at most exponential in size of its ERTG, the upper value of a state in BRA can be computed by analysing an ERG of exponential size. The membership of the ERTG problem in NEXPTIME $\cap$ co-NEXPTIME then follows from the fact that a non-deterministic Turing machine needs to guess a (rational) solution of optimality equations only for exponentially many states, and it can verify in exponential time whether it is indeed a solution.

4.1 Boundary Region Abstraction

The region graph [31] is useful for solving time-abstract optimisation problems on timed automata. The region graph, however, is not suitable for solving timed optimisation problems and games on timed automata as it abstracts away the timing information. The corner-point abstraction [34] is an abstraction of timed automata which retains some timing information, but it is not convenient for the dynamic programming based proof techniques used in this paper. The boundary region abstraction (BRA) [13], a generalisation of the corner-point abstraction, is more suitable for such proof techniques. More precisely, we need to prove certain properties of values in ERTG, which we can do only when reasoning about all states of the ERTG. In the corner point abstraction we cannot do this since it represents only states corresponding to corner points of regions. Here, we generalise the BRA of [13] to handle ERTG.

Timed Successor Regions. Recall that $\mathcal{R}$ is the set of clock regions. For $\zeta, \zeta' \in \mathcal{R}$, we say that ζ' is in the future of ζ, denoted $\zeta \xrightarrow{*} \zeta'$, if there exist $\nu \in \zeta$, $\nu' \in \zeta'$ and $t \in \mathbb{R}_{\geq 0}$ such that $\nu' = \nu{+}t$ and say ζ' is the *time successor* of ζ if $\nu{+}t' \in \zeta \cup \zeta'$ for all $t' \leq t$ and write $\zeta \rightarrow \zeta'$, or equivalently $\zeta' \leftarrow \zeta$, to denote this fact. For regions $\zeta, \zeta' \in \mathcal{R}$ such that $\zeta \xrightarrow{*} \zeta'$ we write $[\zeta, \zeta']$ for the zone $\cup\{\zeta'' \mid \zeta \xrightarrow{*} \zeta'' \wedge \zeta'' \xrightarrow{*} \zeta'\}$.

Thin and Thick Regions. We say that a region ζ is *thin* if $[\nu] \neq [\nu{+}\varepsilon]$ for every $\nu \in \zeta$ and $\varepsilon{>}0$ and *thick* otherwise. We write $\mathcal{R}_{\mathrm{Thin}}$ and $\mathcal{R}_{\mathrm{Thick}}$ for the sets of thin and thick

regions, respectively. Observe that if $\zeta \in \mathcal{R}_{\text{Thick}}$ then, for any $\nu \in \zeta$, there exists $\varepsilon{>}0$, such that $[\nu]{=}[\nu{+}\varepsilon]$ and the time successor of a thin region is thick, and vice versa.

Intuition for the Boundary Region Graph. Recall K is an upper bound on clock values and let $[\![K]\!]_{\mathbb{N}} = \{0, 1, \ldots, K\}$. For any $\nu \in V$, $b \in [\![K]\!]_{\mathbb{N}}$ and $c \in \mathcal{C}$ we define $\mathsf{time}(\nu,(b,c)) \stackrel{\text{def}}{=} b{-}\nu(c)$ if $\nu(c){\leq}b$, and $\mathsf{time}(\nu,(b,c)) \stackrel{\text{def}}{=} 0$ if $\nu(c){>}b$. Intuitively, $\mathsf{time}(\nu,(b,c))$ returns the amount of time that must elapse in ν before the clock c reaches the integer value b. Observe that, for any $\zeta' \in \mathcal{R}_{\text{Thin}}$, there exists $b \in [\![K]\!]_{\mathbb{N}}$ and $c \in \mathcal{C}$, such that $\nu \in \zeta$ implies $(\nu{+}(b{-}\nu(c)) \in \zeta'$ for all $\zeta \in \mathcal{R}$ in the past of ζ' and write $\zeta \rightarrow_{b,c} \zeta'$. The boundary region abstraction is motivated by the following. Consider $a \in Act$, (ℓ,ν) and $\zeta \stackrel{*}{\rightarrow} \zeta'$ such that $\nu \in \zeta$, $[\zeta,\zeta'] \subseteq Inv(\ell)$ and $\nu' \in E(\ell,a)$.

- If $\zeta' \in \mathcal{R}_{\text{Thick}}$, then there are infinitely many $t \in \mathbb{R}_{\geq 0}$ such that $\nu{+}t \in \zeta'$. However, amongst all such t's, for one of the boundaries of ζ', the closer $\nu{+}t$ is to this boundary, the 'better' the timed action (t,a) becomes for a player's objective. However, since ζ' is a thick region, the set $\{t \in \mathbb{R}_{\geq 0} \mid \nu{+}t \in \zeta'\}$ is an open interval, and hence does not contain its boundary values. Observe that the infimum equals $b_{\mathsf{inf}} - \nu(c_{\mathsf{inf}})$ where $\zeta \rightarrow_{b_{\mathsf{inf}},c_{\mathsf{inf}}} \zeta_{\mathsf{inf}} \rightarrow \zeta'$ and the supremum equals $b_{\mathsf{sup}}{-}\nu(c_{\mathsf{sup}})$ where $\zeta \rightarrow_{b_{\mathsf{sup}},c_{\mathsf{sup}}} \zeta_{\mathsf{sup}} \leftarrow \zeta'$. In the boundary region abstraction we include these 'best' timed actions through the actions $(b_{\mathsf{inf}}, c_{\mathsf{inf}}, a, \zeta')$ and $(b_{\mathsf{sup}}, c_{\mathsf{sup}}, a, \zeta')$.
- If $\zeta' \in \mathcal{R}_{\text{Thin}}$, then there exists a unique $t \in \mathbb{R}_{\geq 0}$ such that $\nu{+}t \in \zeta'$. Moreover since ζ' is a thin region, there exists a clock $c \in C$ and a number $b \in \mathbb{N}$ such that $\zeta \rightarrow_{b,c} \zeta'$ and $t = b{-}\nu(c)$. In the boundary region abstraction we summarise this 'best' timed action from region ζ via region ζ' through the action (b,c,a,ζ').

Based on this intuition above the boundary region abstraction is defined as follows.

Definition 11. *For an ERTG $\mathcal{T} = (L, L_F, \mathcal{C}, Inv, Act, Act_{\text{Min}}, Act_{\text{Max}}, E, \delta)$ the BRA of $\mathcal{T}$ is given by the ERG $\widehat{\mathcal{T}} = (\widehat{S}, \widehat{F}, \widehat{A}_{\text{Min}}, \widehat{A}_{\text{Max}}, \widehat{p}_{\text{Min}}, \widehat{p}_{\text{Max}}, \widehat{\pi}_{\text{Min}}, \widehat{\pi}_{\text{Max}})$ where*

- *$\widehat{S} \subseteq L \times V \times \mathcal{R}$ is the (possibly uncountable) set of states such that $(\ell,\nu,\zeta) \in \widehat{S}$ if and only if $\zeta \in \mathcal{R}$, $\zeta \subseteq Inv(\ell)$, and $\nu \in \overline{\zeta}$;*
- *$\widehat{F} = \{(\ell,\nu,\zeta) \in \widehat{S} \mid \ell \in L_F\}$ is the set of final states;*
- *$\widehat{A}_{\text{Min}} \subseteq ([\![K]\!]_{\mathbb{N}} {\times} \mathcal{C} {\times} Act_{\text{Min}} {\times} \mathcal{R}) \cup \{\bot\}$ is the set of actions of player* Min*;*
- *$\widehat{A}_{\text{Max}} \subseteq ([\![K]\!]_{\mathbb{N}} {\times} \mathcal{C} {\times} Act_{\text{Max}} {\times} \mathcal{R}) \cup \{\bot\}$ is the set of actions of player* Max*;*
- *for $\star \in \{\text{Min}, \text{Max}\}$, $s = (\ell,\nu,\zeta) \in \widehat{S}$ and $\alpha = (b_\alpha, c_\alpha, a_\alpha, \zeta_\alpha) \in \widehat{A}_\star$ the probabilistic transition function $p_\star$ is defined if $[\zeta,\zeta_\alpha] \subseteq Inv(\ell)$ and $\zeta_\alpha \subseteq E(\ell,a_\alpha)$ and for any $(\ell',\nu',\zeta') \in \widehat{S}$:*

$$\widehat{p}_\star(s,\alpha)((\ell',\nu',\zeta')) = \textstyle\sum_{C \subseteq \mathcal{C} \wedge \nu_\alpha[C:=0]=\nu' \wedge \zeta_\alpha[C:=0]=\zeta'} \delta[\ell,a_\alpha](C,\ell')$$

  where $\nu_\alpha = \nu{+}\mathsf{time}(\nu,(b_\alpha,c_\alpha))$ and one of the following conditions holds:
  - *$\zeta \rightarrow_{b_\alpha,c_\alpha} \zeta_\alpha$,*
  - *$\zeta \rightarrow_{b_\alpha,c_\alpha} \zeta_{\mathsf{inf}} \rightarrow \zeta_\alpha$ for some $\zeta_{\mathsf{inf}} \in \mathcal{R}$*
  - *$\zeta \rightarrow_{b_\alpha,c_\alpha} \zeta_{\mathsf{sup}} \leftarrow \zeta_\alpha$ for some $\zeta_{\mathsf{sup}} \in \mathcal{R}$;*
- *for $\star \in \{\text{Min}, \text{Max}\}$, $(\ell,\nu,\zeta) \in \widehat{S}$ and $(b_\alpha, c_\alpha, a_\alpha, \zeta_\alpha) \in \widehat{A}_\star$ the reward function $\widehat{\pi}_\star$ is given by $\widehat{\pi}_\star((\ell,\nu,\zeta),(b_\alpha,c_\alpha,a_\alpha,\zeta_\alpha)) = b_\alpha{-}\nu(c_\alpha)$.*

Although the boundary region abstraction is not a finite ERG, for a fixed initial state we can restrict attention to a finite ERG, thanks to the following result of [15,16].

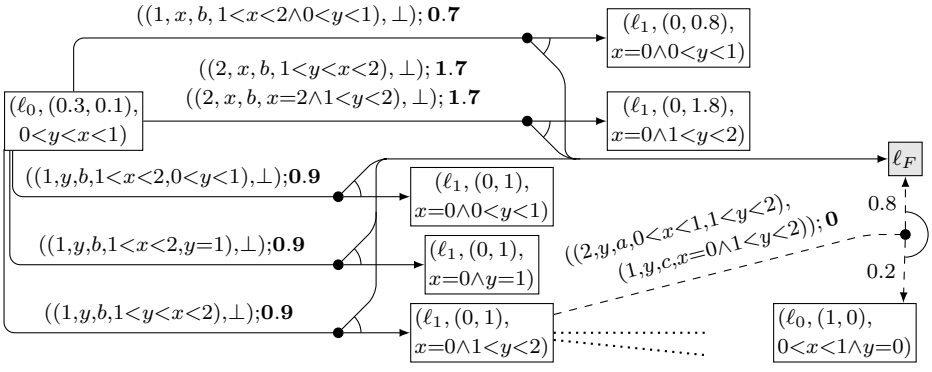


Fig. 2. Sub-graph of the boundary region abstraction for the ERTG of Figure 1

Lemma 12. *For any state of a boundary region abstraction, its reachable sub-graph is finite and is constructible in time exponential in the size of corresponding ERTG.*

Example 13. Sub-graph of BRA reachable from $(\ell_0, (0.3, 0.1), 0{<}y{<}x{<}1)$ for the ERTG of Figure 1 is shown in Figure 2. Edges are labelled (b, c, a, ζ) whose intuitive meaning is to wait until clock c attains the value b and then fire action a. The rewards of edges (indicated in bold) correspond to the time delay before the action is fired. Figure 2 includes the actions available in the initial state and one of action pairs available in $(\ell_1, (0, 1), x{=}0{\wedge}1{<}y{<}2)$. To simplify, the states with location ℓ_F are merged together into a single state labelled ℓ_F and probabilities that are equal to 0.5 are omitted.

4.2 Solving Optimality Equations of a Boundary Region Abstraction

Based on the optimality equations $\mathrm{Opt}^*(\widehat{\mathcal{T}})$ (see Section 2), we define the value improvement function $\mathbf{\Psi} : [\widehat{S}{\to}\mathbb{R}_{\geq 0}]{\to}[\widehat{S}{\to}\mathbb{R}_{\geq 0}]$ where for $f : \widehat{S}{\to}\mathbb{R}_{\geq 0}$ and $s \in \widehat{S}$:

$$\mathbf{\Psi}(f)(s) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } s \in \widehat{F} \\ \min\limits_{\alpha \in \widehat{A}_{\mathrm{Min}}(s)} \left\{ \max\limits_{\beta \in \widehat{A}_{\mathrm{Max}}(s)} \left\{ \widehat{\pi}(s,\alpha,\beta) + \sum\limits_{s' \in \widehat{S}} \widehat{p}(s'|s,\alpha,\beta) \cdot f(s') \right\} \right\} & \text{if } s \notin \widehat{F} \end{cases}$$

By construction, a fixpoint of $\mathbf{\Psi}$ is a solution of $\mathrm{Opt}^*(\widehat{\mathcal{T}})$. The following demonstrates the existence and uniqueness of a fixpoint of $\mathbf{\Psi}$, and thus also the solution of $\mathrm{Opt}^*(\widehat{\mathcal{T}})$.

Proposition 14. *For any ERTG $\mathcal{T}$, the value improvement function $\mathbf{\Psi}$ on the BRA $\widehat{\mathcal{T}}$ has a unique fixed point and equals $\lim_{i\to\infty} \mathbf{\Psi}^i(f)$ for an arbitrary $f \in [\widehat{S} \to \mathbb{R}_{\geq 0}]$.*

Proof. From Assumption 2 and Lemma 12 follows that every $|L{\times}\mathcal{R}|$-th iterate of $\mathbf{\Psi}$ is a contraction. Hence using Banach fixed point theorem the result is immediate. □

4.3 Correctness of the Boundary Region Abstraction Reduction

In this section we show how the optimality equations for the boundary region abstraction can be used to solve optimality equations for its ERTG. Given an ERTG $\mathcal{T}$ and real-valued function $f : \widehat{S} \to \mathbb{R}$ on the states of the BRA $\widehat{\mathcal{T}}$, we define $\widetilde{f} : S \to \mathbb{R}$ by

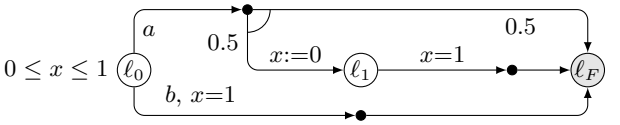


Fig. 3. Example demonstrating optimal strategies are not regionally positional

$\widetilde{f}(\ell,\nu) = f(\ell,\nu,[\nu])$ which gives a real-valued function on the states of $\mathcal{T}$. The following theorem states that, by applying this mapping, the solution of optimality equations for an ERTG is given by that of the optimality equations for its BRA.

Theorem 15. *Let $\mathcal{T}$ be an ERTG. If $P \models \mathrm{Opt}^*(\widehat{\mathcal{T}})$, then $\widetilde{P} \models \mathrm{Opt}^*(\mathcal{T})$.*

To prove Theorem 15 we first introduce quasi-simple functions and state some of their key properties. Next, we show that for any BRA $\widehat{\mathcal{T}}$ the solution of $\mathrm{Opt}^*(\widehat{\mathcal{T}})$ is regionally quasi-simple (a quasi-simple function for every region). Finally, we sketch how Theorem 15 follows from this fact (Proposition 19 and Theorem 21).

Quasi-simple functions. Asarin and Maler [2] introduced simple functions, a finitely representable class of functions, with the property that every decreasing sequence is finite. Given $X \subseteq V$, a function $F{:}X \to \mathbb{R}$ is *simple* if there exists $e \in \mathbb{N}$ and either $F(\nu){=}e$ for all $\nu \in X$, or there exists $c \in C$ and $F(\nu){=}e{-}\nu(c)$ for all $\nu \in X$. A function $F{:}\widehat{S} \to \mathbb{R}_{\ge 0}$ is regionally simple if $F(\ell,\cdot,\zeta)$ is simple for all $\ell \in L$ and $\zeta \in \mathcal{R}$.

For timed games, Asarin and Maler showed that if $f{:}\widehat{S} \to \mathbb{R}_{\ge 0}$ is regionally simple, then $\mathbf{\Psi}(f)$ is regionally simple. Therefore, since $\mathbf{\Psi}$ is a decreasing function, it follows that starting from a regionally simple function in finitely many iterations of $\mathbf{\Psi}$ a fixed point is reached and the upper value in reachability-time games is regionally simple. Also, using the properties of simple functions, [13] shows that for a non-probabilistic reachability-time game, the optimal strategies are *regionally positional*, i.e., in every state of a region the strategy chooses the same action. Unfortunately, in the case of ERTGs, $\mathbf{\Psi}(f)$ is not necessarily regionally simple for any given regionally simple function f. Moreover, as the example below demonstrates, neither is the value of the game necessarily regionally-simple nor optimal strategies regionally positional.

Example 16. Consider the ERTG shown in Figure 3. Observe that for every state (ℓ_0,ν) in the region $(\ell_0, 0{<}x{<}1)$, the optimal expected time to reach ℓ_F equals $\min\{\inf_{t\ge 0}\{t + 0.5{\cdot}1 + 0.5{\cdot}0\}, 1{-}\nu(x)\} = \min\{0.5, 1{-}\nu(x)\}$. Hence optimal expected reachability-time is not regionally simple. Moreover, the optimal strategy is not regionally positional, since if $\nu(x){\le}0.5$, then the optimal strategy is to fire a immediately, while otherwise the optimal strategy is to wait until $\nu(x){=}1$ and fire b.

Due to these results it is not possible to work with simple function, and we define quasi-simple functions. Let $\trianglelefteq \subseteq V{\times}V$ be the partial order on clock valuations, where $\nu \trianglelefteq \nu'$ if and only if there exists a $t \in \mathbb{R}_{\ge 0}$ such that for each clock $c \in C$ either $\nu'(c){-}\nu(c) = t$ or $\nu(c){=}\nu'(c)$. For $x{=}(x_1,\ldots,x_n) \in \mathbb{R}^n$, we let $\|x\|_\infty = \max\{|x_i| \mid 1 \le i \le n\}$.

Definition 17. *Let $X \subseteq V$. A function $F : X \to \mathbb{R}$ is* quasi-simple *if for all $\nu, \nu' \in X$:*

– (Lipschitz Continuous) *there exists $k{\ge}0$ such that $|F(\nu){-}F(\nu')| \le k \cdot \|\nu{-}\nu'\|_\infty$;*

– (Monotonically decreasing and nonexpansive w.r.t. $\trianglelefteq$) $\nu \trianglelefteq \nu'$ *implies* $F(\nu) \geq F(\nu')$ *and* $F(\nu) - F(\nu') \leq \|\nu - \nu'\|_\infty$.

For a convex set $X \subseteq V$ and continuous function $F : X \to \mathbb{R}$, we let $\overline{F} : \overline{X} \to \mathbb{R}$ denote the unique continuous function satisfying $\overline{F}(\nu) = F(\nu)$ for all $\nu \in X$.

Theorem 18 (Properties of Quasi-simple Functions). *Let* $X \subseteq V$.

1. *Every simple function is also quasi-simple.*
2. *If* $F : X \to \mathbb{R}$ *is quasi-simple, then* $\overline{F} : \overline{X} \to \mathbb{R}$ *is quasi-simple.*
3. *If* $F, F' : X \to \mathbb{R}$ *are quasi-simple functions, then both the pointwise minimum and maximum of* F *and* F' *are quasi-simple.*
4. *The limit of a sequence of quasi-simple functions is quasi-simple.*

We say that $f : \widehat{S} \to \mathbb{R}_{\geq 0}$ is regionally quasi-simple if $f(\ell, \cdot, \zeta)$ is quasi-simple for all $\ell \in L$ and $\zeta \in \mathcal{R}$. Using Theorem 18 and Definition 11 we get the following result.

Proposition 19. *If* f *is regionally quasi-simple, then* $\mathbf{\Psi}(f)$ *is regionally quasi-simple.*

From Proposition 14 it follows that for an arbitrary function $f : \widehat{S} \to \mathbb{R}_{\geq 0}$ the limit of the sequence $\langle f, \mathbf{\Psi}(f), \mathbf{\Psi}^2(f), \ldots \rangle$ is the solution of $\mathrm{Opt}^*(\widehat{\mathcal{T}})$. From Proposition 19 it follows that, if we start from a regionally quasi-simple function f, then all the functions in the sequence $\langle f, \mathbf{\Psi}(f), \mathbf{\Psi}^2(f), \ldots \rangle$ are regionally quasi-simple. Since the limit of quasi-simple functions is quasi-simple, the following proposition is immediate.

Proposition 20. *For any ERTG* $\mathcal{T}$*, if* $P \models \mathrm{Opt}^*(\widehat{\mathcal{T}})$*, then* P *is regionally quasi-simple.*

The following result states that, from a regionally quasi-simple solution of the optimality equations for the boundary region abstraction, one can derive the solution of the optimality equations for the expected reachability time-game.

Theorem 21. *For any ERTG* $\mathcal{T}$*, if* $P \models \mathrm{Opt}^*(\widehat{\mathcal{T}})$ *and* P *is regionally quasi-simple, then* $\widetilde{P} \models \mathrm{Opt}^*(\mathcal{T})$.

The following observation is crucial for the proof of Theorem 21.

Lemma 22. *Let* $s = (\ell, \nu) \in S$ *and* $\zeta \in \mathcal{R}$ *such that* $[\nu] \xrightarrow{*} \zeta$*. If* $P : \widehat{S} \to \mathbb{R}$ *is regionally quasi-simple, then the functions:*

$$t \mapsto t + \textstyle\sum_{s' \in S} p(s' | s, (t,a), \bot) \cdot \widetilde{P}(s') \quad \text{and} \quad t \mapsto t + \textstyle\sum_{s' \in S} p(s' | s, \bot, (t,b)) \cdot \widetilde{P}(s')$$

are continuous and nondecreasing on the interval $\{t \in \mathbb{R}_{\geq 0} \mid \nu + t \in \zeta\}$.

5 Conclusions

We introduced expected reachability-time games and showed that the natural decision problem is decidable and in NEXPTIME ∩ co-NEXPTIME. Furthermore, we proved that the turn-based subclass of these games is positionally determined. We believe that the main contribution of this paper is the concept of quasi-simple function that generalise simple functions to the context of probabilistic timed games. In fact, the techniques introduced in this paper extend to expected discounted-time games

(EDTGs)[4] in a straightforward manner, since every expected discounted-time game can be reduced to an expected reachability-time game. Hence all the result presented for ERTGs are valid for EDTGs as well. Regarding other games on probabilistic timed automata, we conjecture that it is possible to reduce expected average-time games to mean payoff games on the boundary region abstraction. However, the techniques presented in this paper are insufficient to demonstrate such a reduction.

Although the computational complexity of solving games on timed automata is high, UPPAAL Tiga [5] is able to solve practical [6,10] reachability and safety properties for timed games by using efficient symbolic zone-based algorithms. A natural future work is to investigate the possibility of extending similar algorithms for probabilistic timed games.

Acknowledgements. We would like to thank anonymous reviewers for their helpful comments and for finding an error in the submitted version. The authors are supported in part by EPSRC grants EP/D076625, EP/F001096, EP/D07956X/2, and by research center ITI, project No. 1M0545.

References

1. Ramadge, P., Wonham, W.: The control of discrete event systems. Proc. IEEE 77(1) (1989)
2. Asarin, E., Maler, O.: As soon as possible: Time optimal control for timed automata. In: Vaandrager, F.W., van Schuppen, J.H. (eds.) HSCC 1999. LNCS, vol. 1569, pp. 19–30. Springer, Heidelberg (1999)
3. Alur, R., Bernadsky, M., Madhusudan, P.: Optimal reachability for weighted timed games. In: Díaz, J., Karhumäki, J., Lepistö, A., Sannella, D. (eds.) ICALP 2004. LNCS, vol. 3142, pp. 122–133. Springer, Heidelberg (2004)
4. Bouyer, P., Cassez, F., Fleury, E., Larsen, K.G.: Optimal strategies in priced timed game automata. In: Lodaya, K., Mahajan, M. (eds.) FSTTCS 2004. LNCS, vol. 3328, pp. 148–160. Springer, Heidelberg (2004)
5. Behrmann, G., Cougnard, A., David, A., Fleury, E., Larsen, K., Lime, D.: UPPAAL-Tiga: Time for playing games! In: Damm, W., Hermanns, H. (eds.) CAV 2007. LNCS, vol. 4590, pp. 121–125. Springer, Heidelberg (2007)
6. Cassez, F., Jessen, J., Larsen, K., Raskin, J., Reynier, P.: Automatic synthesis of robust and optimal controllers: an industrial case study. In: Majumdar, R., Tabuada, P. (eds.) HSCC 2009. LNCS, vol. 5469, pp. 90–104. Springer, Heidelberg (2009)
7. Kwiatkowska, M., Norman, G., Segala, R., Sproston, J.: Automatic verification of real-time systems with discrete probability distributions. Theoretical Computer Science 282 (2002)
8. Jensen, H.: Model checking probabilistic real time systems. In: Proc. 7th Nordic Workshop on Programming Theory. Report 86:247–261, Chalmers University of Technology (1996)
9. Beauquier, D.: Probabilistic timed automata. Theoretical Computer Science 292 (2003)
10. AlAttili, I., Houben, F., Igna, G., Michels, S., Zhu, F., Vaandrager, F.: Adaptive scheduling of data paths using Uppaal Tiga. CoRR abs/0912.1897 (2009)
11. de Alfaro, L., Faella, M., Henzinger, T.A., Majumdar, R., Stoelinga, M.: The element of surprise in timed games. In: Amadio, R.M., Lugiez, D. (eds.) CONCUR 2003. LNCS, vol. 2761, pp. 144–158. Springer, Heidelberg (2003)

[4] In EDTG there is a fixed discount factor $\lambda \in [0,1)$, and when players follow strategies $\mu \in \Sigma_{\mathrm{Min}}$ and $\chi \in \Sigma_{\mathrm{Max}}$ the reward for state s is equal to $\mathbb{E}_s^{\mu,\chi}\left\{\sum_{i=0}^{\infty}\lambda^i \cdot \pi(X_i, Y_{i+1})\right\}$.

12. Courcoubetis, C., Yannakakis, M.: Minimum and maximum delay problems in real-time systems. In: FMSD 1992, vol. 1, Kluwer, Dordrecht (1992)
13. Jurdziński, M., Trivedi, A.: Reachability-time games on timed automata. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) ICALP 2007. LNCS, vol. 4596, pp. 838–849. Springer, Heidelberg (2007)
14. Jurdziński, M., Trivedi, A.: Average-time games. In: Hariharan, R., Mukund, M., Vinay, V. (eds.) Proc. FSTTCS'08. Leibniz International Proceedings in Informatics, vol. 2, Schloss Dagstuhl (2008)
15. Jurdziński, M., Trivedi, A.: Concavely-priced timed automata. In: Cassez, F., Jard, C. (eds.) FORMATS 2008. LNCS, vol. 5215, pp. 48–62. Springer, Heidelberg (2008)
16. Jurdziński, M., Kwiatkowska, M., Norman, G., Trivedi, A.: Concavely-priced probabilistic timed automata. In: Bravetti, M., Zavattaro, G. (eds.) CONCUR 2009 - Concurrency Theory. LNCS, vol. 5710, p. 415. Springer, Heidelberg (2009)
17. Chadha, R., Legay, A., Prabhakar, P., Viswanathan, M.: Complexity bounds for the verification of real-time software. In: Barthe, G., Hermenegildo, M. (eds.) VMCAI 2010. LNCS, vol. 5944, pp. 95–111. Springer, Heidelberg (2010)
18. Hoffmann, G., Wong-Toi, H.: The input-output control of real-time discrete event systems. In: Proc. RTS'92, IEEE Press, Los Alamitos (1992)
19. Asarin, E., Maler, O., Pnueli, A.: Symbolic controller synthesis for discrete and timed systems. In: Antsaklis, P.J., Kohn, W., Nerode, A., Sastry, S.S. (eds.) HS 1994. LNCS, vol. 999, pp. 1–20. Springer, Heidelberg (1995)
20. Brihaye, T., Henzinger, T.A., Prabhu, V.S., Raskin, J.: Minimum-time reachability in timed games. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) ICALP 2007. LNCS, vol. 4596, pp. 825–837. Springer, Heidelberg (2007)
21. Brihaye, T., Bruyère, V., Raskin, J.: On optimal timed strategies. In: Pettersson, P., Yi, W. (eds.) FORMATS 2005. LNCS, vol. 3829, pp. 49–64. Springer, Heidelberg (2005)
22. Bouyer, P., Brihaye, T., Markey, N.: Improved undecidability results on weighted timed automata. Information Processing Letters 98 (2006)
23. Bouyer, P., Forejt, V.: Reachability in stochastic timed games. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009. LNCS, vol. 5556, pp. 103–114. Springer, Heidelberg (2009)
24. Kwiatkowska, M., Norman, G., Parker, D., Sproston, J.: Performance analysis of probabilistic timed automata using digital clocks. FMSD 29 (2006)
25. Berendsen, J., Chen, T., Jansen, D.: Undecidability of cost-bounded reachability in priced probabilistic timed automata. In: Chen, J., Cooper, S. (eds.) TAMC 2009. LNCS, vol. 5532, pp. 128–137. Springer, Heidelberg (2009)
26. Forejt, V., Kwiatkowska, M., Norman, G., Trivedi, A.: Expected reachability-time games. Technical Report RR-10-07, Oxford University Computing Laboratory (2010)
27. Kwiatkowska, M., Norman, G., Trivedi, A.: Quantitative games on probabilistic timed automata. CoRR abs/1001.1933 (2010)
28. Neyman, A., Sorin, S. (eds.): Stochastic Games and Applications. NATO Science Series C, vol. 570. Kluwer Academic Publishers, Dordrecht (2004)
29. Filar, J., Vrieze, K.: Competitive Markov Decision Processes. Springer, Heidelberg (1997)
30. Bouyer, P., Brihaye, T., Bruyère, V., Raskin, J.: On the optimal reachability problem on weighted timed automata. FMSD 31 (2007)
31. Alur, R., Dill, D.: A theory of timed automata. Theoretical Computer Science 126 (1994)
32. Jurdziński, M., Sproston, J., Laroussinie, F.: Model checking probabilistic timed automata with one or two clocks. Logical Methods in Computer Science 4 (2008)
33. Condon, A.: The complexity of stochastic games. Information and Computation 96 (1992)
34. Bouyer, P., Brinksma, E., Larsen, K.G.: Staying alive as cheaply as possible. In: Alur, R., Pappas, G.J. (eds.) HSCC 2004. LNCS, vol. 2993, pp. 203–218. Springer, Heidelberg (2004)

Diagnosis Using Unfoldings of Parametric Time Petri Nets

Bartosz Grabiec[2], Louis-Marie Traonouez[3], Claude Jard[2], Didier Lime[1], and Olivier H. Roux[1,⋆]

[1] École Centrale de Nantes, IRCCyN, Nantes, France
[2] ENS Cachan & INRIA, IRISA, Rennes, France
Université européenne de Bretagne
[3] Software Technologies Laboratory, Università di Firenze, Italy

Abstract. This paper considers the model of Time Petri Nets (TPNs) extended with time parameters and its use to perform on-line diagnosis of distributed systems. We propose to base the method on unfoldings. Given a partial observation, as a possibly structured set of actions, our method determines the causal relation between events in the model that explain the observation. It can also synthesize parametric constraints associated with these explanations. The method is implemented in the tool Romeo. We present its application to the diagnosis of the example of a cowshed with pigs.

Keywords: Unfolding, Time Petri Nets, parameters, diagnosis, supervision.

1 Introduction

In this paper, we decided to bring attention on a dynamic verification method, called *model-based supervision*. It is established that diagnosing dynamical systems, represented as discrete-event systems, amounts to finding what happened to the system from existing observations (an event log) derived from sensors. In this context, the diagnostic task consists in determining the trajectories compatible with the observations. The standard situation is that the observed events correspond to the firing of some transitions of the model, while the other transitions are just internal (this situation is called "partial observation" in supervisory control theory [4]). Supervision, based on unfoldings [7,12] in our case, is implemented by the on-the-fly construction of the unfolding, guided by the observations. With this dynamic approach, since we consider only finite sequences of observations, decidability questions become much easier. The only requirement is to be able to decide whether a transition can be fired or not. Petri nets for supervisory control and diagnosis have been proposed in numerous papers (see for instance [16] and [9]). In most cases the construction of diagnosers is

⋆ This work was partially funded by the ANR national research program DOTS (ANR-06-SETI-003).

K. Chatterjee and T.A. Henzinger (Eds.): FORMATS 2010, LNCS 6246, pp. 137–151, 2010.

based on the state graph (i.e. the interleaving view). The use of unfoldings is more recent. Safe ordinary nets are used in [8] with emphasis on distributed diagnosis. This has been extended to safe time Petri nets in [5]. The parametric case has not been considered yet.

The great interest of unfoldings in that task is their ability to infer the possible causal dependencies, which are not in general part of the observations. We think that adding parameters in specifications is a real need. It is often difficult to fix them a priori: indeed, we expect from the analysis some useful information about their possible values. This feature clearly adds some "robustness" to the modeling phase. It is particularly relevant for the supervision activity we consider, in which an arbitrary choice of parameters often avoids to find explanations compatible with the observations. This leads to rejection of the model; moreover, no additional knowledge how to correct it is provided.

We implemented our method inside the Romeo tool developed in the IRCCyN lab in Nantes [11], available as free software. This implementation allowed us to demonstrate the proposed supervision method on small case studies. In this paper we chose to develop a new case study, the "cowshed with pigs". It is freely inspired from [10], in which the idea was to show how Uppaal can handle some hybrid models. Here, we consider a Time Petri Net modelling with time parameters. By observing some particular transitions of the model, we show that it is possible to infer causalities between the corresponding events, allowing us to correlate them in order to find the root causes. Furthermore, the method can compute constraints on parameters that must be satisfied in order to explain the observations.

We consider here for the first time, the possibility of having a structured set of observations. The goal of the supervision is to produce explanations compatible with this set (no contradiction between the respective causal structures).

The contributions of this paper are:

- a general method for on-line diagnosis based on Time Petri Nets with parameters, able of causal, time, and parametric inference from a structured set of observations.
- an illustration using an original model of cowshed, which could be of general interest for the community.

The paper is organised as follows. In Section 2, we first present the Time Petri net model with parameters and the way it can be unfolded. Then the model of observations is presented in Section 3 and how it is used to guide the construction of the unfolding. Section 4 describes our case study and illustrates the method, before a few words of conclusion.

2 Time Petri Nets with Parameters and Their Unfolding

2.1 General Notations

We denote by the set of non-negative integers, by the set of rational numbers and the set of real numbers. For A denoting the sets or , $A_{\geq 0}$ (resp. $A_{>0}$)

denotes the subset of non-negative (resp. strictly positive) elements of A. Given $a, b \in$ such that $a \leq b$, we denote by $[a..b]$ the set of integers greater or equal to a and less or equal to b. For any set X, we denote by $|X|$ its cardinality. In the symbolic expressions, $\wedge$ denotes the logical conjunction, $\vee$ the logical disjunction and $\neg$ the logical negation operators. We will also use $\Rightarrow$ as the logical implication.

For a function f on a domain D and a subset C of D, we denote by $f_{|C}$ the restriction of f to C.

Let X be a finite set. A (rational) *linear expression* on X is an expression of the form $a_1x_1 + \cdots + a_nx_n$, with $n \in$, $\forall i, a_i \in$ and $x_i \in X$. The set of linear expressions on X is denoted $Expr(X)$. A *linear constraint* on X is an expression of the form $L_X \sim b$, where L_X is a linear expression on X, $b \in$ and $\sim \in \{<, \leq, \geq, >\}$. We will also use abbreviations like $=$ and $\neq$.

For the sake of readability, when non-ambiguous, we will "flatten" nested tuples, *e.g.* $\langle\langle\langle B, E, F\rangle, l\rangle, v, \theta\rangle$ will be written $\langle B, E, F, l, v, \theta\rangle$.

2.2 Petri Nets

Definition 1 (Place/transition net). *A place/transition net (P/T net) is a tuple $\langle P, T, W\rangle$ where: P is a finite set of* places*, T is a finite set of* transitions*, with $P \cap T = \emptyset$ and $W \subseteq (P \times T) \cup (T \times P)$ is the* flow relation*.*

This structure defines a directed bipartite graph.

We further define, for all $x \in P \cup T$, the following sets: ${}^\bullet x = \{y \in P \cup T \mid (y, x) \in W\}$ and $x^\bullet = \{y \in P \cup T \mid (x, y) \in W\}$. These set definitions naturally extend by union to subsets of $P \cup T$.

A *marking* $m : P \rightarrow$ is a function such that (P, m) is a multiset. For all $p \in P$, $m(p)$ is the number of *tokens* in the place p. In this paper we restrict our study to *1-safe* nets, *i.e.* nets such that $\forall p \in P,\ m(p) \leq 1$. Therefore, in the rest of the paper, we usually identify the marking m with the set of places p such that $m(p) = 1$. In the sequel we will call *Petri net* a marked P/T net, *i.e.* a pair $\langle \mathcal{N}, m\rangle$ where $\mathcal{N}$ is a P/T net and m a marking of $\mathcal{N}$, called *initial marking*.

A transition $t \in T$ is said to be enabled by the marking m if ${}^\bullet t \subseteq m$. We denote by $\mathsf{en}(m)$, the set of transitions enabled by m.

There is a path $x_1, x_2, \ldots, x_n$ in a P/T net iff $\forall i \in [1..n]$, $x_i \in P \cup T$ and $\forall i \in [1..n-1], (x_i, x_{i+1}) \in W$.

In an acyclic P/T net, consider $(x, y) \in P \cup T$. x and y are *causally related*, which we denote by $x < y$, iff there exist a path in the net from x to y. x and y are in *conflict*, which we denote by $x \# y$, iff there exists two paths $p, t, \ldots, x$ and $p, t', \ldots, y$, starting from the same place $p \in P$ but such that $t \neq t'$. It is also convenient to consider the relation of direct conflict between transitions, denoted x conf y, indicating that they share in their presets the place that originated the conflict (${}^\bullet x \cap {}^\bullet y \neq \emptyset$). x and y are in *concurrency*, which we denote by x co y, iff none of the two previous relations holds, that is to say $\neg(x < y) \wedge \neg(y < x) \wedge \neg(x \# y)$.

An *occurrence net* is an acyclic P/T net, finite by precedence, and such that no element is in conflict with itself and each place has at most one input transition. We use the classical terminology of *conditions* and *events* to refer to the places and transitions in an occurrence net.

Definition 2 (Branching process). *A* branching process *of a Petri net* $\mathcal{N} = \langle P, T, W, m_0 \rangle$ *is a labeled occurrence net* $\beta = \langle \mathcal{O}, l \rangle$ *where* $\mathcal{O} = \langle B, E, F \rangle$ *is an occurrence net and* $l : B \cup E \to P \cup T$ *is the labeling function such that:*

- $l(B) \subseteq P$ *and* $l(E) \subseteq T$,
- *for all* $e \in E$, *the restriction* $l_{|^\bullet e}$ *of* l *to* $^\bullet e$ *is a bijection between* $^\bullet e$ *and* $^\bullet l(e)$,
- *for all* $e \in E$, *the restriction* $l_{|e^\bullet}$ *of* l *to* $e^\bullet$ *is a bijection between* $e^\bullet$ *and* $l(e)^\bullet$,
- *for all* $e_1, e_2 \in E$, *if* $^\bullet e_1 = {}^\bullet e_2$ *and* $l(e_1) = l(e_2)$ *then* $e_1 = e_2$.

E *should also contain the special event* $\perp$, *such that:* $^\bullet \perp = \emptyset$, $l(\perp) = \emptyset$, *and* $l_{|\perp^\bullet}$ *is a bijection between* $\perp^\bullet$ *and* m_0.

Example 1. Fig. 1b shows one branching process of the net presented in Fig. 1a (ignoring any timing or parameter information). The labels are put inside the nodes. We can see that the branching process in Fig. 1b unfolds the loop t_1, t_2, t_0 once. This loop could be unfolded infinitely many times, leading to an infinite branching process.

Branching processes can be partially ordered by a *prefix relation*. There exists the greatest branching process according to this relation for any Petri net $\mathcal{N}$, which is called the *unfolding* of $\mathcal{N}$, denoted $\mathcal{U}(\mathcal{N})$.

Let $\beta = \langle B, E, F, l \rangle$ be a branching process.

A co-*set* in β is a subset B' of B such that $\forall b, b' \in B',\ b \mathsf{co}\ b'$.

A *configuration* of β is a set of events $E' \subseteq E$ which is causally closed and conflict-free, that is to say $\forall e' \in E', \forall e \in E,\ e < e' \Rightarrow e \in E'$ and $\forall e, e' \in E', \neg(e \# e')$.

For any co-set B', $l(B')$ defines a subset of the marking of the net. A *cut* is a maximal co-set (inclusion-wise). For any configuration E', we can define the set $\mathsf{Cut}(E') = E'^\bullet \setminus {}^\bullet E'$, which is the marking of the Petri net obtained after executing the sequence of events in E'.

An *extension* of β is a pair $\langle t, e \rangle$ such that e is an event not in E , s.t. $^\bullet e \subseteq B$ is a co-set, the restriction of l to $^\bullet e$ is bijection between $^\bullet e$ and $^\bullet t$ and there is no $e' \in E$ s.t. $l(e') = t$ and $^\bullet e' = {}^\bullet e$. Adding e to E and labeling e with t gives a new branching process. Starting from the event $\perp$, and adding successively possible extensions forms the "unfolding algorithm".

2.3 Parametric Time Petri Nets

A mainstream way of adding time to Petri nets is by equipping transitions with a time interval [13,3]. We consider here an extension allowing the designer to leave open the knowledge of time bounds by putting symbolic expressions on parameters in time intervals instead of rational constants.

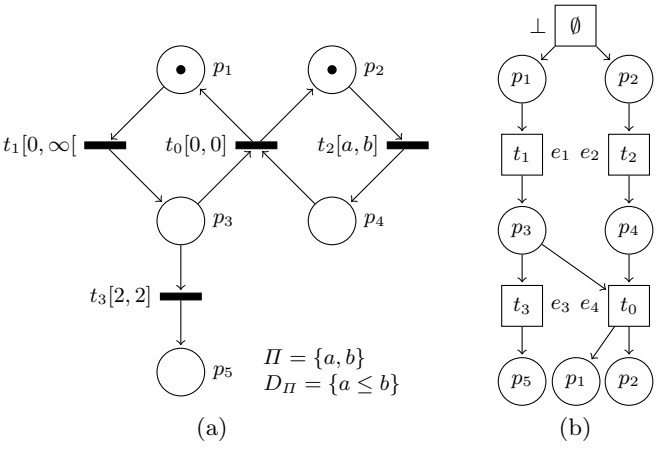


Fig. 1. A parametric time Petri net (a) and a prefix of the unfolding of its underlying (untimed) Petri net (b)

Definition 3 (Parametric Time Petri net). *A parametric Time Petri Net (PTPN) is a tuple* $\langle P, T, W, m_0, \mathsf{eft}, \mathsf{lft}, \Pi, D_\Pi \rangle$ *where:*

$\langle P, T, W, m_0 \rangle$ *is a Petri net,* Π *is a finite set of* parameters ($\Pi \cap (P \cup T) = \emptyset$)*,* D_Π *is a conjunction of linear constraints describing the set of* initial constraints *on the parameters, and* $\mathsf{eft} : T \to \ _{\geq 0} \cup Expr(\Pi)$ *and* $\mathsf{lft} : T \to \ _{\geq 0} \cup \{\infty\} \cup Expr(\Pi)$ *are functions respectively called* earliest (eft) *and* latest (lft) *transition firing times. For each transition* t*, if* $\mathsf{eft}(t)$ *and* $lft(t)$ *are constants, it is assumed that* $\mathsf{eft}(t) \leq \mathsf{lft}(t)$*, otherwise, it is assumed that* $D_\Pi \Rightarrow \mathsf{eft}(t) \leq \mathsf{lft}(t)$.

Example 2. Fig. 1a gives an example of a PTPN. Notice that the time interval of transition t_2 refers to two parameters a and b. The only initial constraint is $D_\Pi = \{a \leq b\}$.

Given a PTPN $\mathcal{N} = \langle P, T, W, m_0, \mathsf{eft}, \mathsf{lft}, \Pi, D_\Pi \rangle$, we denote by $\mathsf{Untimed}(\mathcal{N})$ the Petri net $\langle P, T, W, m_0 \rangle$. The definition of unfolding for PTPN is developed in [14,15]. It relies on an extension and improvement of [6] to have a compact representation of the unfolding and to deal with parameters. The idea is to decorate the unfolding of the underlying net in associating to each event e a symbolic expression $\theta(e)$ representing the constraints that must be satisfied to justify the occurrence of e. For each event e, we consider its firing date represented by the variable θ_e. The expressions on events are boolean expressions on linear constraints on the set of variables and parameters. Fig.2 gives an example of such "decorated" unfolding.

Let $\mathcal{N} = \langle P, T, W, m_0, \mathsf{eft}, \mathsf{lft}, \Pi, D_\Pi \rangle$ be a PTPN and $\beta = \langle B, E, F, l \rangle$ be the associated unfolding of $\mathsf{Untimed}(\mathcal{N})$. We define the *enabling date* of an event $e \in E$ as the expression $\mathsf{TOE}(e)$ standing for $\max_{f \in {}^\bullet e} \theta_f$. It gives the date at which the corresponding transition has been enabled.

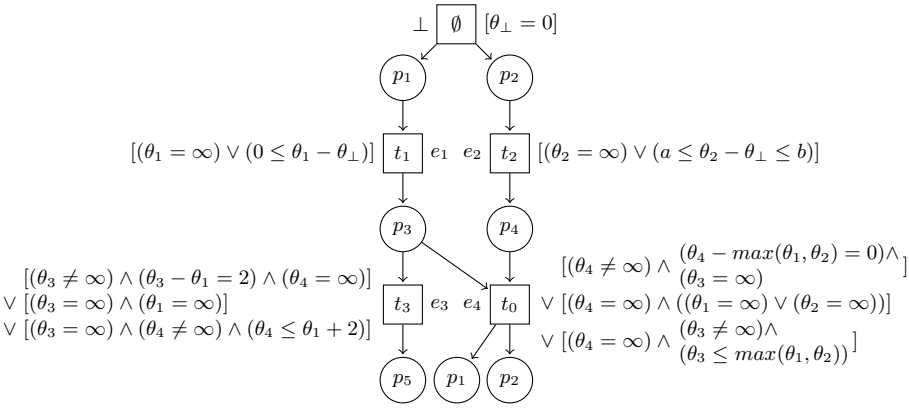


Fig. 2. A prefix of the symbolic unfolding of the PTPN of Fig. 1a

Definition 4 (Valid timing function for an unfolding). *Given a PTPN* $\mathcal{N} = \langle P, T, W, m_0, \mathsf{eft}, \mathsf{lft}, \Pi, D_\Pi \rangle$. *Let* $\beta = \langle B, E, F, l \rangle$ *be the unfolding of* $\mathsf{Untimed}(\mathcal{N})$. *The* timing function θ *is defined by* $\theta(\bot) = 0$ *and* $\forall e \in E$ $(e \neq \bot)$,

$$\begin{cases} \big[(\theta_e \neq \infty) \ \wedge \ (\mathsf{eft}(l(e)) \leq \theta_e - \mathsf{TOE}(e) \leq \mathsf{lft}(l(e))) \wedge \bigwedge_{e' \in E, e' \ \mathsf{conf}\ e} (\theta_{e'} = \infty)\big] \\ \vee \ \big[(\theta_e = \infty) \ \wedge \ \bigvee_{b \in {}^\bullet e} (\theta_b = \infty)\big] \\ \vee \ \big[(\theta_e = \infty) \ \wedge \ \bigvee_{e' \in E, e' \ \mathsf{conf}\ e} [(\theta_{e'} \neq \infty) \wedge \ (\theta_{e'} \leq \mathsf{TOE}(e) + \mathsf{lft}(l(e)))]\big] \end{cases}$$

Note that in this definition, the parameters appear through the functions eft and lft.

The first line of the expression means that the event e has been fired, and consequently that no conflicting events has been fired and that its firing date must conform to its time interval according to the TPN semantics. The remaining two lines consider the case in which the event e has not been fired (coded by the expression $\theta_e = \infty$). There are two possibilities: either a preceding event has not yet fired, or a conflicting event has been fired and has prevented e to occur. This latter case means that such conflicting event has fired while e was enabled.

Example 3. Fig. 2 shows a symbolic prefix of the unfolding of the PTPN in Fig. 1a. We can see that each event is attributed with a symbolic expression. The expressions are formed with variables denoting the firing dates of the considered event and of its neighbourhood (the events that directly precede and those in conflict) and parameters. In practice, the expressions are implemented in polyhedrons.

We also define the set of events temporally preceding an event $e \in E$ as: $\mathsf{Earlier}(e) = \{e' \in E \mid \theta(e') < \theta(e) \text{ is satisfiable}\}$.

Following the standard semantics of TPNs, [2] has defined the notion of valid time configuration, which can be used here:

Definition 5 (valid time configuration). *A configuration* E' *of* $\mathcal{U}(\mathsf{Untimed}(\mathcal{N}))$ *is a* valid time configuration *of* $\mathcal{U}(\mathcal{N})$ *iff* $(\theta_\perp = 0)$ *and*

$$\bigwedge_{e \in E' \setminus \{\perp\}} [\theta_e \geq \mathsf{TOE}(e) + \mathsf{eft}(l(e)) \wedge \bigwedge_{e' \in \mathsf{en}(l(\mathsf{Cut}(\mathsf{Earlier}(e))))} \theta_e \leq \mathsf{TOE}(e') + \mathsf{lft}(l(e'))]$$

Let us consider a maximal (in term of set inclusion) configuration E' of $\mathcal{U}(\mathsf{Untimed}(\mathcal{N}))$, extended with the events that are in direct conflict E'' and equipped with the corresponding symbolic expressions of $\mathcal{U}(\mathcal{N})$. Assuming that events in E' have fired and that events in E'' not, E' is a valid time configuration if the conjunction of all expressions of $E' \cup E''$ is satisfiable. This leads to the following theorem [14].

Theorem 1 (Correctness). *Let* $\langle B, E, F, l, v, \theta \rangle$ *be the unfolding of a parametric time Petri net* $\mathcal{N} = \langle P, T, W, m_0, \mathsf{eft}, \mathsf{lft}, \Pi, D_\Pi \rangle$*. Consider a maximal configuration* $E' \subseteq E$*, and* $E'' = \{e \in E \mid \exists e' \in E', e \ \mathsf{conf}\ e'\}$
E' *is a valid time configuration iff*

$$[\bigwedge_{e \in E'} (\theta_e \neq \infty) \wedge \bigwedge_{e \in E''} (\theta_e = \infty)] \Rightarrow \bigwedge_{e \in E' \cup E''} \theta(e) \text{ is satisfiable.}$$

3 Application to Supervision

We first define the notion of structured observation and then show how to guide the construction of a finite unfolding containing the configurations that are compatible with the observations. We consider that the real distributed system under supervision has been instrumented in such a way that it will produce events (like prints used for debugging) during its execution. These events have a name, picked up in some finite alphabet Σ and can be possibly related to each other. In practice, we consider three cases: two events can be causally related, they can be concurrent, or their relation is not known. As usual, the causal relation must be an order. The two others are just symmetric.

Definition 6 (Observation). *An* observation *is a finite set of events* O*, equipped with a causal order* $\preceq$ *and a symmetric relation co. If two events are not related, their relation is said to be "unknown". An event also has a name, addressed by the labelling function* $\lambda : O \to \Sigma$*.*

In order to relate the observation and the model, we also consider that transitions of the PTPN are labelled by a similar function $\lambda : T \to \Sigma \cup \{\epsilon\}$. The ϵ symbol not belonging to Σ is used to indicate that the occurrence of the transition cannot be linked to an observable event. The labelling does not need to be injective, and in general the same observation can be explained by several trajectories of the model.

We construct the unfolding compatible with the observation. To define this notion of compatibility, we consider the maximal configurations and ask they do not contain events and relations that contradicts the observation. Given an observation O, we consider the Parikh vector $\varpi(O) = (|\lambda^{-1}(a)|)_{a\in\Sigma}$, which counts the number of occurrences of each action in O. The same function can also be applied to configurations, considering that for each event e, $\lambda(e)$ is in fact $\lambda(l(e))$.

Definition 7 (Compatibility). *The unfolding of a PTPN $\mathcal{N}$ is* compatible *with an observation O if all its maximal (in the sense of set inclusion) configurations are. A configuration E is* compatible *with an observation iff:*

- $\forall e \in E, \varpi(E) = \varpi(O)$ *and*
- $\forall o_1, o_2 \in O, o_1 \preceq o_2 \Rightarrow$
  $\exists e_1, e_2 \in E$ *s.t.* $(\lambda(o_1) = \lambda(e_1)) \wedge (\lambda(o_2) = \lambda(e_2)) \wedge (e_1 \leq e_2)$
- $\forall o_1, o_2 \in O, o_1$ *co* $o_2 \Rightarrow$
  $\exists e_1, e_2 \in E$ *s.t.* $(\lambda(o_1) = \lambda(e_1)) \wedge (\lambda(o_2) = \lambda(e_2)) \wedge (e_1$ *co* $e_2)$

Theorem 2 (Finiteness). *Given a finite observation, if the PTPN does not contain loops of ϵ transitions, the set of compatible configurations is finite and thus the unfolding.*

Proof. Because of the finiteness of the original Petri net, the only possibility to obtain an infinite object is to have an infinite configuration. Such a configuration contains some observable events (events e such that $\lambda(l(e)) \neq \epsilon$). They are in finite number, due to the finiteness of the observation and by application of the Parikh constraint. Thus, the only possibility is to have an infinite number of ϵ events. Because of the safeness of the net, this infinite set of events must form a chain of causality, which is prevented by the absence of ϵ-loop in the net.

At the end of the observation, we obtain a finite unfolding in which each event is equipped with a symbolic expression. From Theorem 1, it is possible to extract the valid timed configurations. This is done by considering the maximal configurations of the underlying untimed net, extended by the events that are in direct conflict with some event of the configuration. The associated symbolic constraint is given by Theorem 1. After Boolean simplification, keeping only the configurations in which the expression is satisfiable, we obtain a set of timed configurations which constitutes the set of "explanations". An explanation adds in general a lot of information to the observation:

- It has inferred some added causal and concurrent relations between the observable events;
- it has inserted also some patterns of non observable events;
- it gives the constraint that must be satisfied between all the firing dates of the events;
- it gives some constraints about the possible values of the parameters.

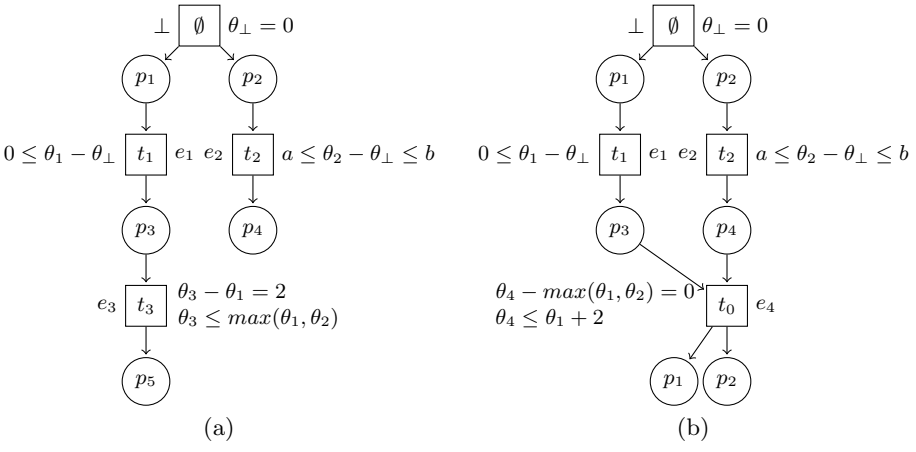


Fig. 3. Two possible explanations

This is illustrated in Fig. 3. We have considered the PTPN of Fig. 1a in which only transitions t_1 and t_2 are observable and labelled with the same letter. Let us now consider a simple observation formed with only two occurrences of the letter. The finite symbolic unfolding we obtain is the one depicted in Fig. 2. There are only two maximal valid timed configurations as shown in Fig. 3.

4 Case Study

4.1 The Continuous Model

In this section, we present a realistic case-study based on the industrial case study for climate control in a cowshed proposed in [10]. The problem is to keep the temperature, humidity, CO2 and ammonia concentrations at specified levels so that the well-being of pigs is ensured. Though it would be relevant to model temperature, humidity, CO2 and ammonia concentration we limit ourselves to modeling only temperature. It would though be easy to include the disregarded climate parameters since the mixing dynamics are, roughly, identical.

The cowshed is divided into distinct climatic zones which interact by exchanging air flow. Besides internal air flow a zone interact with the ambient environment by activating a ventilator in an exhaust pipe and also by opening a screen to let fresh air into the building. Air flowing from outside into the i^{th} zone is denoted $Q_i^{in}[m^3/s]$. Air flowing from the i^{th} zone to outside is denoted $Q_i^{out}[m^3/s]$. Air flowing from zone i to $i+1$ is denoted $Q_{i,i+1}[m^3/s]$(air flow is defined positive from a lower index to a higher index). A stationary flow balance for each zone i is found: $Q_{i-1,i} + Q_i^{in} = Q_{i,i+1} + Q_i^{out}$ where by definition $Q_{0,1} = Q_{N,N+1} = 0$. The flow balance for zone i is illustrated in Fig. 4.

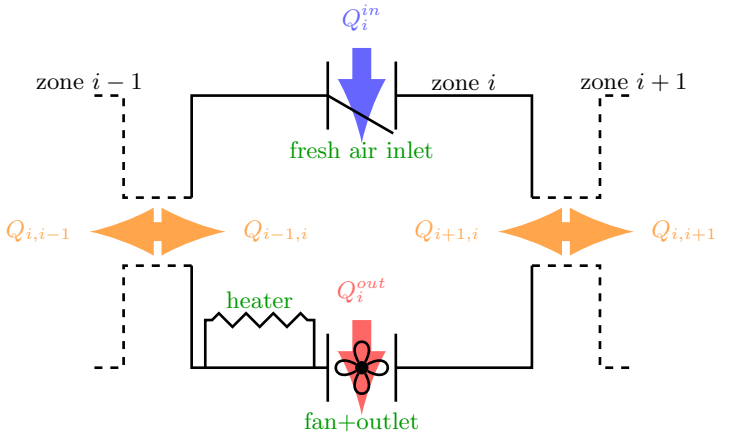


Fig. 4. The zone number i and the air flows through it

The temperature in a given zone is impacted in several ways:

- Each zone is equipped with a heater which can be either on ($u_i = 1$) or off ($u_i = 0$). We denote by $U_i[J/s]$ the resulting heating;
- The pigs in the zone produce heat, denoted by $W_i[J/s]$;
- Air flows from/to adjacent zones;
- Fresh air flows in from outside through the inlet. T_{amb} is the outside temperature. $Q_i^{in,max}$ is the maximum flow of air drawn from outside;
- Air flows outside by means of the fan. $Q_i^{out,max}$ is the maximum flow of air fanned outside.

The evolution of the temperature in zone i is therefore given by the following differential equation, where V_i is the volume of zone i, ρ_{air} the air density [kg/m^3], and c_{air} the specific heat capacity of air [J/kg.C]:

$$\begin{aligned} \frac{dT_i}{dt} &= f(T_{i-1}, T_i, T_{i+1}), \text{with} \\ f(T_{i-1}, T_i, T_{i+1}) &= \frac{1}{V_i}[Q_i^{in} T_{amb} - Q_i^{out} T_i + Q_{i-1,i} T_{i-1} - Q_{i,i-1} T_i \\ &\quad - Q_{i,i+1} T_i - Q_{i+1,i} T_{i+1} + \frac{u_i U_i + W_i}{\rho_{air} c_{air}}] \end{aligned}$$

Among all the factors impacting the temperature in the zone, only three are directly controllable:

- The heater, which is on or off;
- The aperture of the inlet, between 0 and some maximal value inducing $Q_i^{in,max}$;
- The speed of the fan, between 0 and some maximal value inducing Q^{out,max_i}.

In particular, the internal air flows between zones are induced by these last two parameters. We also decided to extend the system with an extra feature which is a possibility of failures of fans (depicted by the state OOO_i in Fig. 5).

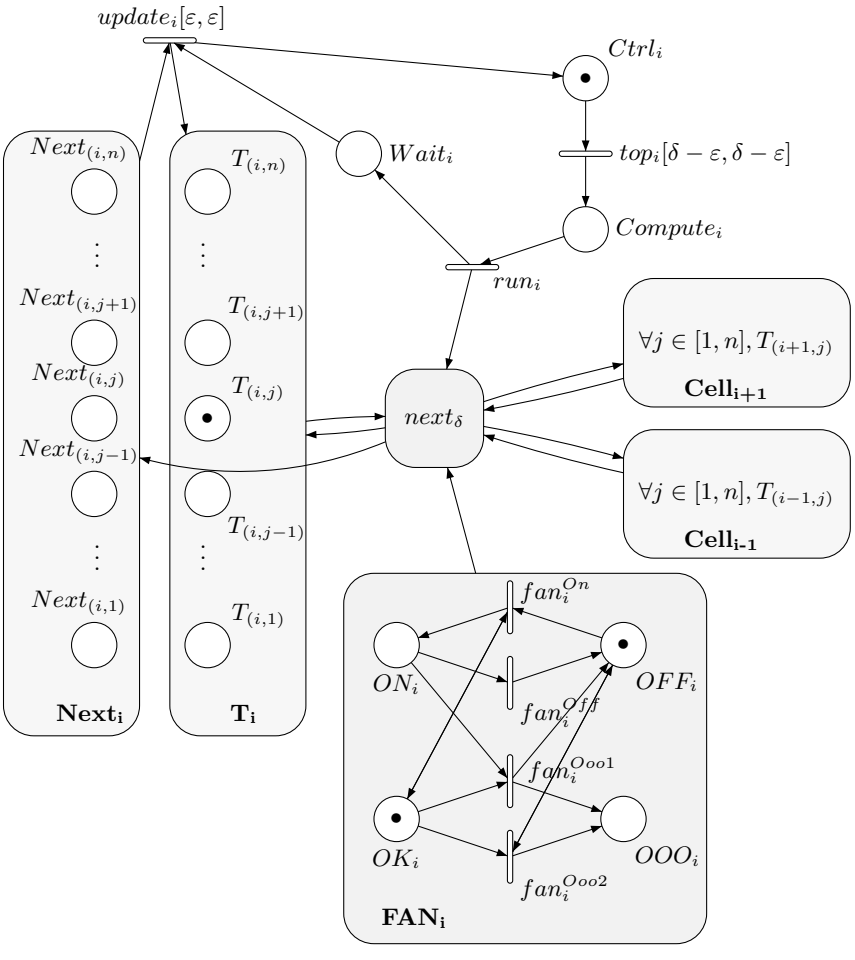


Fig. 5. The cell i

4.2 The PTPN Generation

We consider a discrete evolution of temperature of each cell on a scale of n degrees. Each possible temperature in a cell is represented as a place. The marked place gives the current temperature of the cell. We sample and compute the successor state considering that T_{i-1}, T_i and T_{i+1} are constants. Let us denote $next_\delta(T_i)$ the temperature of cell i, obtained in these conditions after δ units of time (t.u.).

We define $C_\delta \in [0, 1]$ as the coefficient of heat exchange on the duration δ.

1. Without fan (no communication with outside) and without pig:

$$next_\delta(T_i) = T_i + C_\delta * \frac{T_{i-1} - T_i}{3} + C_\delta * \frac{T_{i+1} - T_i}{3}$$

On an infinite delay ($C_\delta = 1$), we would obtain the heat equilibrium: $next_\delta(T_i) = \frac{T_{i-1}+T_i+T_{i+1}+T_{amb}}{3}$

2. Without fan, but with pigs:
   Let T_δ^W be the heat brought by the pigs during δ time units.

$$next_\delta(T_i) = T_i + C_\delta * \frac{T_{i-1} - T_i}{3} + C_\delta * \frac{T_{i+1} - T_i}{3} + T_\delta^W$$

3. With fan and pigs:
   Let $C_\delta^{amb} \in [0, 1[$ the coefficient of heat exchange with outside (depends on the power of the fan: $C_\delta^{amb} = 1$ means a fan with an infinite power).
   $next_\delta(T_i) = C_\delta^{amb} * T_{amb} + (1 - C_\delta^{amb}) * (T_i + C_\delta * \frac{T_{i-1}-T_i}{3} + C_\delta * \frac{T_{i+1}-T_i}{3} + T_\delta^W)$

The model of a cell i, given in Fig. 5, consists of 4 blocks:

- the block $\mathbf{T_i}$ with one place per temperature,
- the block $\mathbf{Next_i}$ is used to store the intermediate state,
- the block $\mathbf{FAN_i}$ is a model of the behaviour of the fan including possibility of failures,
- the block $next_\delta$ compute the next temperature of the cell and performs the exchange of tokens between bloks $\mathbf{T_i}$ and $\mathbf{Next_i}$ using the block $\mathbf{FAN_i}$ and the temperature of adjacent cells. This exchange is given by the quantization (on the n temperature levels) of the function $next_\delta(T_i)$.

The sampling is controlled by the places $Ctrl_i$ $Compute_i$ and $Wait_i$ and transitions top_i, run_i and $update_i$. The new temperature of the cell i is computed in two steps. First, the new temperature $next_\delta(T_i)$ is computed at $(\delta - \epsilon)$ t.u. and the result is stored in the intermediate places of block $\mathbf{Next_i}$. This intermediate result is obtained in zero t.u. and does not depend of the interleaving since marking of block $\mathbf{T_i}$ is not modified by this computation. Then, the intermediate result is moved from $\mathbf{Next_i}$ to $\mathbf{T_i}$ after ε t.u.

All the possibilities are defined, which leads to a complex graph. With n the number of temperature levels, the model have $2 \times n^3$ transitions.This one is tedious to build by hand. Thus we decided to automatically generate the model by programming a tcl-tk generator, parameterized by the number of considered cells and the temperature scale. This generator of 1000 lines builds an PTPN model as a XML file directly read by Romeo. For example, Fig. 6 gives an insight of the model with 2 cells and 3 levels of temperature.

4.3 The Diagnosis Experiment

The goal of the experiment is to show a case in which a certain maximal temperature is exceeded in one of the cells. This leads in turn to the death of some pigs. In the model, we assume that we can observe the changes of temperature in each of the cells, and we can get to know if some pigs died. As a result we would like to know the possible explanations of cases in which a pig died. For example, we can imagine a situation where a fan is broken in a cell. Moreover, we would like to obtain some information about the possible dates of death.

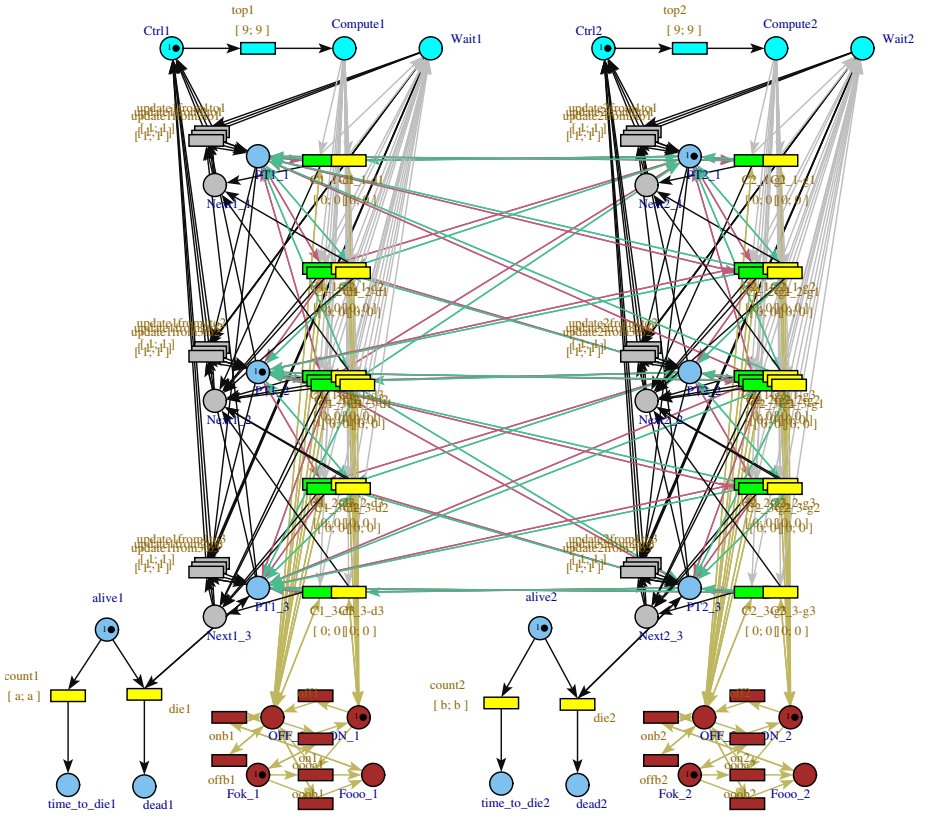


Fig. 6. The model with 2 cells and 3 levels of temperature

The experiment was performed on the system presented in the previous section in Fig. 6. It consists of 2 cells and 3 levels of temperature. In the example, we assume that the temperature of cells is monitored at some given rate, which amounts to 10 units of time.

To be able to execute our scenario, we added some additional transitions to the model (see Fig. 6). They do not change the main functionality of the model. They are used for two reasons. The first reason is to verify whether the critical temperature was reached or not. Consequently, each time the extra transitions are fired, one can observe the death of pigs as a consequence of the excessive temperature. The second reason is to compute a minimal delay between two events: the initial event of the model (at time 0), and the potential death of pigs in a cell. The structure which implements this task is based on two transitions. One of the transitions has a parametrized constraint $[a, a]$. The transitions share an input place with a token. The token is active from the very beginning of the model activity. However, the non-parametrized transition t depends also on some other token at place p (in the model it denotes the maximal temperature).

Our goal is to get to know when the place p can activate the non-parametrized transition (in the model it denotes the death of pig). We can note that using the structure we get the minimal possible date of firing t by reading the parameter a.

For the purpose of the experiment, we set up an initial temperature for each cell and we entered a set of observed events into our tool for analysis, i.e.: log with unordered temperature measurements, and an event which signals death of an animal in one of the cells. In total, there were 8 observable events and a limit of 6 unobservable events in the observation. As a result, we obtain a prefix consisting of 108 events with 4 possible explanations. From the prefix we can observe that in any of the four scenarios, the fans in the both cells have to be broken before the temperature become critical. Moreover, as a result of the experiment, we get possible valuations of the parameters given in the model. Thus, we get to know that the minimal time amount necessary in order to reach the state dangerous for the pigs amounts to 20 units of time.

To perform the experiment we used a prototype implemented in Romeo, which is a software for analysis of time Petri nets. The experiment was executed on a small machine with 1GB of RAM and 2GHz Intel Pentium processor. The computation time of the example needed about 15 seconds. During our experiments we tested also some different variants of the problem: with more cells, with more levels of temperature, and with different observations. In general, size of the model and observations can strongly influence the time complexity of the diagnosis. It is not difficult to observe that one of the issues which plays a great role in the time consumption of the analysis is the number of unobservable, or indistinguishable events in the system. During the tests we observed many difficult cases in the context of time complexity. We intend to address that issue in our future work and improve the software tool we used for our experiments.

5 Conclusion

The current version of the Romeo tool 2.9.0 is available on the webpage [1]. It offers the possibility of computing symbolic unfoldings for safe time Petri Nets with parameters. When guided by a sequence of actions, this feature allows the user to perform some diagnosis. The diagnosis consists in a finite prefix of the unfolding, presenting all the possible explanations of the input sequence. The explanations show the inferred causal relationships between the events of the model and also give the possible values for the parameters. We think that such an integrated method is a real added-value for the analysis of concurrent systems, and opens the door to deal with even more complex models like TPNs with stopwatches, or TPNs with more robust time semantics (e.g. with imperfect clocks).

References

1. Roméo - a tool for time Petri nets analysis, `http://romeo.rts-software.org`
2. Aura, T., Lilius, J.: A causal semantics for time Petri nets. Theoretical Computer Science 243(2), 409–447 (2000)

3. Berthomieu, B., Diaz, M.: Modeling and verification of time dependent systems using time Petri nets. IEEE trans. on Soft. Eng. 17(3), 259–273 (1991)
4. Cassandras, C.G., Lafortune, S.: Introduction to Discrete Event Systems, 2nd edn. Springer, Heidelberg (2008)
5. Chatain, T., Jard, C.: Time supervision of concurrent systems using symbolic unfoldings of time Petri nets. In: Pettersson, P., Yi, W. (eds.) FORMATS 2005. LNCS, vol. 3829, pp. 196–210. Springer, Heidelberg (2005)
6. Chatain, T., Jard, C.: Complete finite prefixes of symbolic unfoldings of safe time Petri nets. In: Donatelli, S., Thiagarajan, P.S. (eds.) ICATPN 2006. LNCS, vol. 4024, pp. 125–145. Springer, Heidelberg (2006)
7. Esparza, J., Heljanko, K.: Unfoldings, A Partial-Order Approach to Model Checking. Monographs in Theoretical Computer Science. Springer, Heidelberg (2008)
8. Fabre, E., Benveniste, A., Haar, S., Jard, C.: Distributed monitoring of concurrent and asynchronous systems. Journal of Discrete Event Systems, special issue 5, 33–84 (2005)
9. Giua, A.: Petri net state estimators based on event observation. In: Proceedings of the IEEE ICDC, pp. 4086–4091 (1997)
10. Jessen, J.K., Rasmussen, J.I., Larsen, K.G., David, A.: Guided controller synthesis for climate controller using uppaal-tiga. In: Raskin, J.-F., Thiagarajan, P.S. (eds.) FORMATS 2007. LNCS, vol. 4763, pp. 227–240. Springer, Heidelberg (2007)
11. Lime, D., Roux, O.H., Seidner, C., Traonouez, L.-M.: Romeo: A parametric model-checker for Petri nets with stopwatches. In: Kowalewski, S., Philippou, A. (eds.) TACAS 2009. LNCS, vol. 5505, pp. 54–57. Springer, Heidelberg (2009)
12. McMillan, K.L.: Using unfolding to avoid the state space explosion problem in the verification of asynchronous circuits. In: Probst, D.K., von Bochmann, G. (eds.) CAV 1992. LNCS, vol. 663, pp. 164–177. Springer, Heidelberg (1993)
13. Merlin, P.M.: A study of the recoverability of computing systems. PhD thesis, Dep. of Information and Computer Science, University of California, Irvine, CA (1974)
14. Traonouez, L.M.: Vérification et dépliages de réseaux de Petri temporels paramétrés. PhD thesis, Ecole centrale de Nantes (2009), `http://www.irccyn.ec-nantes.fr/~traonoue/publications/these.pdf`
15. Traonouez, L.M., Grabiec, B., Jard, C., Lime, D., Roux, O.H.: Symbolic unfolding of parametric stopwatch petri nets. In: 8th International Symposium on Automated Technology for Verification and Analysis (ATVA 2010). LNCS, vol. 6252. Springer, Heidelberg (2010)
16. Ushio, T., Onishi, I., Okuda, K.: Fault detection based on Petri net models with faulty behaviors. In: Proceedings of the IEEE Int. Conf. on Systems, Man, and Cybernetics, pp. 113–118 (1998)

From MTL to Deterministic Timed Automata*

Dejan Ničković[1] and Nir Piterman[2]

[1] IST, Klosterneuburg, Austria
[2] Imperial College London, London, UK

Abstract. In this paper we propose a novel technique for constructing timed automata from properties expressed in the logic MTL, under bounded-variability assumptions. We handle full MTL and include all future operators. Our construction is based on separation of the continuous time monitoring of the input sequence and discrete predictions regarding the future. The separation of the continuous from the discrete allows us to determinize our automata in an exponential construction that does not increase the number of clocks. This leads to a doubly exponential construction from MTL to deterministic timed automata, compared with triply exponential using existing approaches.

We offer an alternative to the existing approach to linear real-time model checking, which has never been implemented. It further offers a unified framework for model checking, runtime monitoring, and synthesis, in an approach that can reuse tools, implementations, and insights from the discrete setting.

1 Introduction

The ability to write high-level, intuitive specifications in temporal logic is what, in many cases, underlies the success of applications in model checking, runtime monitoring, and controller synthesis. In this paper we concentrate on applications where linear-time temporal logic (LTL) [20] is used to describe ongoing behaviors. In the case of model-checking the specification φ is effectively translated to a nondeterministic automaton $\mathcal{A}_\varphi$ that is composed with the system and analyzed in the search of *bad* behaviors. In runtime monitoring, it is not enough to translate φ to a nondeterministic automaton. Indeed, an individual behavior of the system may induce multiple runs, which have to be followed simultaneously. Resolving nondeterminism on-the-fly induces an extra computational cost at every step of the monitoring. Thus, it is best to use a *deterministic* automaton. As monitoring is usually concerned only with finite input sequences, a simple determinization, based on the subset construction, suffices. Finally, in controller synthesis, we would like to automatically generate an implementation of a system S that satisfies a specification φ [24]. Synthesis requires to convert φ to a deterministic automaton, however, full determinization of ω-automata is required [26,23].

Here, we are interested in these three applications in the context of real-time. The main model for reasoning quantitatively about time is timed automata [3]. Timed automata are suitable for modeling certain time-dependent phenomena, and their reacha-

* Part of this work was done while both authors were at Verimag, France. The first author is supported in part by the EU project COMBEST and the EU Network of Excellence ARTIST-DESIGN. The second author is supported by the grant UK EPSRC EP/E028985/1.

K. Chatterjee and T.A. Henzinger (Eds.): FORMATS 2010, LNCS 6246, pp. 152–167, 2010.

bility (or empty language) problem is decidable, facts that have been exploited in verification tools, e.g., Kronos [29] and Uppaal [15].

As in the untimed case, we would like to combine the model of timed automata with a powerful logic and apply algorithms for model checking, runtime monitoring, and controller synthesis. Many variants of real-time logics [14,5,12,11] have been proposed. However, unlike in the untimed case, the correspondence between simply-defined logics and variants of timed automata is not simple. One of the most popular dense-time extensions of LTL is the logic MITL introduced in [4] as a restriction of the logic MTL [14]. The principal modality of MITL is the timed until $\mathcal{U}_I$, where I is some non-singular interval. A formula $p\mathcal{U}_{(a,b)}q$ is satisfied by a model at any time instant t that admits q at some $t_0 \in (t+a, t+b)$, and where p holds continuously from t to t_0. Decidability of MITL was established in [4] by converting an MITL formula to a nondeterministic timed automaton and analyzing the structure of that automaton. Further investigations of MITL and MTL suggested alternative translations of MITL to nondeterministic timed automata [17,18] and used alternating timed automata to show decidability of MTL in certain circumstances [22].

The mentioned translations of MITL to nondeterministic timed automata provide the necessary theory for MITL model checking. However, to the best of our knowledge, (due to their complexity) there are no tools implementing linear-time model checking of timed systems. In addition, these constructions do not offer a viable solution for runtime monitoring or controller synthesis, as they do not produce *deterministic* automata. In general, for MITL it is impossible to construct deterministic automata [16]. Consider, for example, the formula $\varphi = \Box_{(0,a)}(p \rightarrow \Diamond_{(a,b)} q)$, which says that for every $t \in (0, a)$, if p is true at time t then there is a time point $t' \in (t+a, t+b)$ in which q holds. A deterministic automaton for φ would require infinite memory to remember all possible occurrences of p within $(0, a)$. Even if we find a fragment of MITL that can be translated to deterministic timed automata, the known constructions [4,17] cannot be used, as arbitrary timed automata cannot be determinized [3]. Determinization constructions rely on the ability to create a finite representation of a set of runs. When clocks are involved, it is impossible to finitely represent all the values of the clocks in a deterministic timed automaton. Even worse, the realizability problem for MITL is undecidable [10]; and for MTL, realizability is undecidable even for finite words, but becomes decidable if one restricts the number of clocks used by the controller [8].

Synthesis from specifications given as deterministic timed automata is possible [21]. In order to produce deterministic automata one has to resolve two sources of nondeterminism in timed automata arising from MITL:

1. *Unbounded variability:* there can be an unbounded number of events in a fixed interval of time, which the automaton needs to remember. It follows that one cannot fix a bound on the number of clocks that the timed automaton needs to use for memorizing relevant events. Most examples showing that timed automata cannot be determinized or complemented use unbounded variability.
2. *Acausality:* the satisfaction of a formula at time t may depend on values of inputs at time $t' > t$. In order to determine whether the input sequence satisfies the formula at time t, the automaton "predicts" the future values of inputs, and aborts later

(at t') computations corresponding to wrong predictions. Acausality is a source of nondeterminism for both untimed and real-time temporal logics.

Wilke showed that removing the first source of nondeterminism (i.e., bounding the variability of input signals) enables complementation and determinization of timed automata [28]. His proof uses monadic-second order logic and offers no practical solution. Several classes of timed automata that can be determinized are identified in [7]. These automata include strongly non-Zeno timed automata, which reject words that vary more than a certain bound. The construction in [7] is impractical for MITL specifications as it is doubly exponential and uses an exponential number of clocks; leading to a triple exponential construction with a doubly exponential number of clocks. Also, determinization is considered only for finite words.

In [18] a construction from MTL under bounded variability to deterministic timed automata is suggested. This solution, however, only partially solves acausality: only invariants "always φ" are handled, where φ is an MTL formula with past temporal operators and *bounded* future operators. Bounded future operators in the scope of an always are syntactically changed to past operators. As the past is naturally deterministic [16], the standard construction of [17] produces a deterministic automaton. Compared with [7], the construction is single exponential instead of triple exponential.

It follows that bounded variability is a practical solution for constructing deterministic timed automata from real-time specifications. For many applications, bounded variability is a very reasonable assumption [6]. High variability arises from the frequency at which systems operate; and almost all systems have a bound on the frequencies in which they operate. In general, high frequencies are hard to produce making systems more complicated. In practice, bounded variability covers the most interesting cases.

Using the assumption of bounded-variability, we show how to fully handle acausality of MTL semantics. For that, we devise a completely new construction for converting MTL formulas to nondeterministic automata, which relies on bounded variability. Our construction separates the timed automata into two parts: (i) a standard timed automaton that monitors changes in the inputs and memorizes events with clocks; and (ii) a *dependent timed automaton*, that makes passive use of clocks controlled by the first part. The first part is *deterministic* by construction. The second part, namely the dependend timed automaton, generates *discrete* predictions regarding future events.

We then show that dependent timed automata with the following properties can be determinized: (i) transitions cannot be enabled throughout the stay in a state (no dense nondeterminism); and (ii) when a transition is enabled the automaton can stay for a little while in the target state. Both conditions hold for the automata we construct. The determinization itself is a slight variant of determinization for untimed finite automata on infinite words [26,23]. The determinization is exponential and does not change the number of clocks. The overall result is doubly exponential construction from MTL to deterministic timed automata.[1]

Our new construction has many additional advantages even when producing nondeterministic automata. In our construction the number of clocks depends on the number of propositions, the depth of the longest chain of *nested* timed operators, and the

[1] In contrast, an exponential determinization for general timed automata under bounded variability is impossible. The doubly exponential determinization in [7] is optimal.

bounded variability of the input. Updates to clocks are extremely simple and depend directly on the input. In previous constructions clocks are allocated according to variability and depth of each operator separately. In our construction the number of states associated with every unbounded-until is constant, while in previous constructions *every* temporal operator requires states that are proportional to the number of active intervals that the temporal operator may have. For the fragment of bounded future operators considered in [18] our automata are deterministic by construction and do not require the extra determinization step. Finally, previous translations read "singular" (zero-duration) and "open" intervals, while in our construction automata use only left-closed right-open intervals, which matches existing tools, such as Kronos [29] and Uppaal [15].

2 Signals and MTL

Let AP be a set of atomic propositions. A signal over AP is a function $w : \mathbb{T} \to 2^{AP}$, where $\mathbb{T}$ is either the non-negative reals $\mathbb{R}_{\geq 0}$ (infinite-length) or an interval $[0, r)$ (finite length). We denote by w_p the projection of w to the proposition p. Concatenation of two finite signals w_1 and w_2 defined over $[0, r_1)$ and $[0, r_2)$, respectively, is the finite signal $w = w_1 \cdot w_2$, defined over $[0, r_1 + r_2)$ as $w[t] = w_1[t]$ for $t < r_1$ and $w[t] = w_2[t - r_1]$ for $t \geq r_1$. Consider signal w over AP and signal w' over AP' of the same length such that $AP \cap AP' = \emptyset$. Their product $w \times w'$ is the signal such that for $p \in AP$ we have $(w \times w')_p = w_p$ and for $p \in AP'$ we have $(w \times w')_p = w'_p$. Product is defined for sets of words in the natural way. We say that w is of *bounded-variability* l if for every proposition p, every $t \in \mathbb{T}$, and every $t_0 < t_1 < \cdots < t_l \in [t, t + 1)$ there is i such that $w_p[t_i] = w_p[t_{i+1}]$. That is, proposition p changes its value at most $l - 1$ times in every interval of length 1. In this paper we consider only signals of bounded variability.

The syntax of the future fragment of MTL interpreted over dense-time signals is defined by the grammar $\varphi := p \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \; \mathcal{U}_I \; \varphi_2$, where p belongs to a set $AP = \{p_1, \ldots, p_n\}$ of propositions and I is an interval of one of the following forms: $[b, b]$, (a, b), or (a, ∞) where $0 \leq a < b$ are integers. An until $\mathcal{U}_I$ is *unbounded* if I is (a, ∞), otherwise it is *bounded*.

The semantics of an MTL formula φ with respect to a signal w is defined recursively via the satisfiability relation $(w, t) \models \varphi$, indicating that signal w satisfies φ at time t:

$$
\begin{array}{ll}
(w,t) \models p & \leftrightarrow w_p[t] = 1 \\
(w,t) \models \neg\varphi & \leftrightarrow (w,t) \not\models \varphi \\
(w,t) \models \varphi_1 \vee \varphi_2 & \leftrightarrow (w,t) \models \varphi_1 \text{ or } (w,t) \models \varphi_2 \\
(w,t) \models \varphi_1 \mathcal{U}_I \varphi_2 & \leftrightarrow \exists\, t' \in t + I \text{ st } (w,t') \models \varphi_2 \;\wedge\; \forall\, t'' \in (t,t')\; (w,t'') \models \varphi_1
\end{array}
$$

A formula φ is satisfied by w if $(w, 0) \models \varphi$ and $L(\varphi) = \{w \mid (w, 0) \models \varphi\}$. Other Boolean and temporal operators are derived as usual. In particular, $\Diamond_I \varphi = \mathrm{T} \; \mathcal{U}_I \; \varphi$ and $\Box_I \varphi = \neg \Diamond_I \neg\varphi$ are the *eventually* and *always* operators. Similarly, intervals such as $[a, b]$, $[a, b)$, $[a, \infty)$ can be expressed. When $I = (0, \infty)$, we simply omit it.

We suggest a new construction from MTL under bounded variability to timed automata. The construction is based on computing a bound f such that the automaton at time t memorizes the input signal at the interval $[t - f, t)$. The truth value of the formula is computed with a delay: when the automaton is reading time t it computes the value

$\mathsf{fut}(p) = 0$, where p is a proposition.
$\mathsf{fut}(\varphi_1 \vee \varphi_2) = max(\mathsf{fut}(\varphi_1), \mathsf{fut}(\varphi_2))$
$\mathsf{fut}(\neg\varphi_1) = \mathsf{fut}(\varphi_1)$
$\mathsf{fut}(\varphi_1 \mathcal{U}_I \varphi_2) = a + 2 + max(\mathsf{fut}(\varphi_1), \mathsf{fut}(\varphi_2))$, where $I = (a, \infty)$.
$\mathsf{fut}(\varphi_1 \mathcal{U}_I \varphi_2) = b + max(\mathsf{fut}(\varphi_1), \mathsf{fut}(\varphi_2))$, where $I = (a, b)$ or $I = [b, b]$.

Fig. 1. The function $\mathsf{fut}()$

of the formula for time $t - f$. The function fut, which determines the interval's size, is defined in Figure 1.

We compute the truth value of a formula φ at time t by looking on the interval $[t, t + \mathsf{fut}(\varphi))$. The only non-trivial part of this definition is the unbounded until $\varphi = \varphi_1 \mathcal{U}_{(a,\infty)} \varphi_2$, which, apart from a, requires two additional lookaheads $(a + 2)$. In this paragraph, we give an informal overview of the ingredients needed for our construction of determinizable timed automata from an unbounded until formula. The formal definition of the construction is presented in Section 4 and illustrated by Figure 4. We want an automaton to know for sure that φ is true without remembering how long φ_1 held. For that, the interval $(t, t + a]$ is *never* sufficient: if φ_1 holds throughout $(t, t + a]$, we must guess that φ_1 continues until φ_2 holds later and if φ_1 holds until φ_2 holds *within* $[t, t + a)$, then we need to know whether φ_1 held also before t. Thus, we memorize φ_1 and φ_2 in $(t, t + a + 1]$, which is *sometimes* sufficient to know for sure that φ is true. For example, φ holds for sure if φ_1 holds throughout $(t, t + a + 0.5]$ and φ_2 holds at $t + a + 0.5$. We use the information recorded in $(t, t + a + 1]$, combined with a purely **discrete** guess about the satisfaction of the *untimed* formula $\varphi_1 \mathcal{U}_{\geq 0} \varphi_2$ at time $t + a + 1$, to construct an automaton that determines the truth value of φ at t. It distinguishes between three situations: (1) φ_2 is true at some $t' \in (t + a, t + a + 1]$ and φ_1 holds throughout (t, t'), hence φ clearly holds at t; (2) φ_1 does not hold at some $t' \in (t, t + a + 1]$ and φ_2 is false during $(t + a, t')$, hence φ is false at t; (3) φ_1 is true during $(t, t + a + 1]$ and φ_2 is false throughout $(t + a, t + a + 1]$, and we need to predict the truth value of $\varphi_1 \mathcal{U}_{\geq 0} \varphi_2$ at $t + a + 1$. The automaton is nondeterministic: it makes *wrong* guesses and aborts runs. We use the additional lookahead to eliminate one type of wrong guess. Consider, for example, some $t > 0$ and the case that φ_1 holds continuously throughout $[0, \infty)$ and φ_2 holds in $(0, t + a)$ and $(t + a + 1, \infty)$. For every $t' < t$ we have φ holds at time t' and our automaton *sees* that φ is true. At exactly time t, the automaton is blind: φ_2 does not hold throughout $(t + a, t + a + 1]$. It has to guess whether φ continues to hold. If it guesses that φ is false, then at every $t'' > t$ it realizes its mistake and aborts as, indeed, φ_2 holds at $t'' + a + 1$. Hence, the "length of error" was 0: exactly at time t. We eliminate 0-duration errors using the second lookahead $t + a + 2$, which provides enough 'prediction' power to avoid such errors. It follows that the 0-duration error elimination results in an automaton that remains non-deterministic, but this step is important for its determinization described in Section 5.

The function fut is used also to decide where the truth value of a formula expires. For example, if at time t we know the value of q and r in $[t - 8, t)$, our knowledge for the formula $q \mathcal{U}_{(2,\infty)} r$ expires at time $t - 2 - 2$. As $q \mathcal{U}_{(2,\infty)} r$ is unbounded, this truth value may depend further on a guess.

3 Timed Automata

We use a variant of timed automata that differs slightly from the classical definitions [3,13,1]. Our automata read multi-dimensional *dense-time* Boolean signals and output Boolean signals. Input and output are associated with states *and* transitions. We also extend the domain of clock values to include the special symbol $\perp$ indicating that the clock is currently *inactive* and extend the order relation on $\mathbb{R}_{\geq 0}$ accordingly by letting $\perp < v$ for every $v \in \mathbb{R}_{\geq 0}$. We freely use multiplication by -1 and comparison with negative values. It follows that $-\perp > -v$ for every $v \in \mathbb{R}_{\geq 0}$. For a set $A \subseteq \mathbb{R}^n_{\geq 0}$ we use $cl(A)$ to denote its (topological) closure.

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 $\mathcal{H} = (\mathbb{R}_{\geq 0} \cup \{\perp\})^n$. 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. An *atomic clock constraint* is a condition of the form $x \bowtie y + d$ or $x \bowtie d$, where x and y are clocks, $\bowtie \in \{<, \leq, \geq, >\}$, and d is an integer. Let $\mathbb{A}(\mathcal{C})$ denote the set of atomic constraints over the set $\mathcal{C}$ of clocks. For a set X, let $\mathcal{B}^+(X)$ denote the set of positive Boolean formulas over X (i.e., Boolean formulas built from elements in X using $\wedge$ and $\vee$). Let $\mathbb{C}(\mathcal{C}) = \mathcal{B}^+(\mathbb{A}(\mathcal{C}))$ denote the set of *constraints* over the set of clocks $\mathcal{C}$. We view a constraint $c \in \mathbb{C}(\mathcal{C})$ as a subset $c \subseteq \mathcal{H}$. We also introduce free real variables to constraints and quantify over them. That is, we use constraints in the first-order theory of the reals, where clocks in $\mathcal{C}$ are free variables. In the full version we show that application of quantifier elimination results in clock constraints in $\mathbb{C}(\mathcal{C})$. We used the tool in [19] to eliminate quantifiers in some of the examples below.

Definition 1. *A* timed automaton (TA) *is $\mathcal{A} = \langle \Sigma, Q, \mathcal{C}, \lambda, I, \Delta, q_0, \mathcal{F} \rangle$, where Σ is the input alphabet, Q is a finite set of discrete states, and $\mathcal{C}$ is a set of clock variables. We assume that Σ is 2^{AP} for some set of propositions AP. We freely use Boolean combinations of propositions to denote sets of letters. The labeling function $\lambda : Q \to \Sigma$ associates an input letter with every state. The staying condition (invariant) I assigns to every state q a constraint $I(q) \in \mathbb{C}(\mathcal{C})$. 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 $x := 0$, $x := \perp$, or $x := y$ or a set of such assignments. Finally, q_0 is the initial state. Transitions leaving q_0 have the guard* True *and use only updates of the form $x := 0$. We consider generalized Büchi automata, where $\mathcal{F} \subseteq 2^Q$, and parity automata, where $\mathcal{F} = \langle F_0, \ldots, F_k \rangle$ partitions Q. Unless mentioned explicitly, automata are of the first type.*

The behavior of a TA as it reads a signal w consists of a strict alternation between time progress periods, where the TA stays in a state q as long as $w[t] = \lambda(q)$ and $I(q)$ holds, and discrete instantaneous transitions guarded by clock conditions. Formally, a step of the TA is one of the following:

- A time step $(q, v) \xrightarrow{\sigma^t} (q, v + t)$ $t \in \mathbb{R}_{>0}$ where $\sigma = \lambda(q)$ and $(v, v + t) \subseteq cl(I(q))$.
- 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 $v' = \rho(v)$.

Let $v_\perp = (\perp, \ldots, \perp)$ be the assignment of $\perp$ to all clocks. A *run* of the TA starting from a configuration $(q_0, v_\perp)$ is a finite or infinite sequence of strictly alternating time and discrete steps of the form $\zeta : (q_0, v_0) \xrightarrow{\delta_0} (q_1, v_1) \xrightarrow{\sigma_1^{t_1}} (q_1, v_1 + t_1) \xrightarrow{\delta_1} (q_2, v_2) \xrightarrow{\sigma_2^{t_2}} (q_2, v_2 + t_2) \xrightarrow{\delta_2} \cdots$, such that $\sum_i t_i$ diverges. We denote by $\zeta[t]$ the valuation of the clocks in ζ at time t. That is, at $t = \sum_{j=1}^{i} t_j$ we have $\zeta[t] = v_{i+1}$ and for $t' < t_{i+1}$ we have $\zeta[t + t'] = \zeta[t] + t'$. Given a run ζ, the set $inf(\zeta)$ is the set of states q such that the set of time instants in which q is visited is unbounded. A run is accepting according to the generalized Büchi condition $\mathcal{F}$ if for every $F \in \mathcal{F}$ we have $F \cap inf(\zeta) \neq \emptyset$. A run is accepting according to the parity condition $\mathcal{F}$ if the minimal i such that $F_i \cap inf(\zeta) \neq \emptyset$ is even. The input signal carried by the run is $\sigma_1^{t_1} \cdot \sigma_2^{t_2} \cdots$, where we abuse notation and denote by $\sigma_i^{t_i}$ the concatenation of the punctual signal σ_i and the open signal $\sigma_i^{t_i}$. That is $\sigma_i : [0, t_i) \to \Sigma$ such that forall $t \in [0, t_i)$ we have $w(t) = \sigma_i$. An input signal is *accepted* if it is carried by an accepting run. The *language* of A, denoted $L(A)$, is the set of accepted input signals.

Let $A_i = \langle \Sigma_i, Q_i, \mathcal{C}_i, \lambda_i, I_i, \Delta_i, q_0^i, \mathcal{F}_i \rangle$, for $i \in \{1, 2\}$, be two TA such that $\mathcal{C}_1 \cap \mathcal{C}_2 = \emptyset$. The composition $A_1 \parallel A_2 = \langle \Sigma_1 \times \Sigma_2, Q_1 \times Q_2, \mathcal{C}_1 \cup \mathcal{C}_2, \lambda, I, \Delta, (q_0^1, q_0^2), \mathcal{F} \rangle$, where $\lambda(q_1, q_2) = (\lambda_1(q_1), \lambda_2(q_2))$, $I(q_1, q_2) = I_1(q_1) \wedge I_2(q_2)$, and $\mathcal{F} = \{S \times T \mid S \in \mathcal{F}_1 \text{ and } T = Q_2, \text{ or } S = Q_1 \text{ and } T \in \mathcal{F}_2\}$. The transition relation Δ includes:

- *Simultaneous transitions* $((q_1, q_2), g, \rho, (q_1', q_2'))$, where $(q_i, g_i, \rho_i, q_i') \in \Delta_i$ for $i \in \{1, 2\}$, $g = g_1 \wedge g_2$ and $\rho = \rho_1 \cup \rho_2$,
- *Left-side transitions* $((q_1, q_2), g, \rho, (q_1', q_2))$, where $(q_1, g_1, \rho, q_1') \in \Delta_1$ and $g = g_1 \wedge I_2(q_2)$, and
- *Right-side transitions* $((q_1, q_2), g, \rho, (q_1, q_2'))$, where $(q_2, g_2, \rho, q_2') \in \Delta_2$ and $g = I_1(q_1) \wedge g_2$.

These three types of transitions reflect the asynchronicity of the composed TA, allowing one to take a discrete step while the other is in the middle of a time step. Note that the alphabets of A_1 and A_2 are disjoint.

Lemma 1. $L(A_1 \parallel A_2) = L(A_1) \times L(A_2)$

A TA is *deterministic* if from every reachable configuration every event and every 'non-event' leads to exactly one configuration:

Definition 2. *A* deterministic timed automaton *is an automaton whose guards and staying conditions satisfy:*

1. *For every two distinct transitions* (q, g_1, ρ_1, q_1) *and* (q, g_2, ρ_2, q_2) *we have either* $\lambda(q_1) \neq \lambda(q_2)$ *or* $g_1 \wedge g_2$ *is unsatisfiable.*
2. *For every transition* (q, g, ρ, q')*, either* $\lambda(q) \neq \lambda(q')$ *or the intersection of* g *and* $I(q)$ *is either empty or isolated, i.e., there does not exist an open interval* (t, t') *such that* $(t, t') \subseteq I(q)$ *and* $(t, t') \cap g \neq \emptyset$*.*

Dependent timed automata (DTA) are transducers of runs of TA. Accordingly, DTA have output and composition of DTA allows one automaton to read the output of the other. DTA passively read clocks of other TA and have no clocks of their own. Thus, they *depend* on other TA to supply them the clocks.

Definition 3. *A* dependent timed automaton (DTA) *is* $B = \langle \Sigma, \Gamma, Q, \mathcal{C}, \gamma, I, \Delta, q_0, \mathcal{F} \rangle$*, where* Σ*,* Q*,* $\mathcal{C}$*,* q_0*, and* $\mathcal{F}$ *are as in timed automata. DTA have, in addition, an output*

alphabet Γ, and an output function $\gamma : Q \to \Gamma$. The labeling function is replaced by a more general staying condition I that assigns to every state a Boolean combination of atomic constraints and input letters $I : Q \to \mathcal{B}^+(\mathbb{A}(\mathcal{C}) \cup \Sigma)$. The transition relation Δ consists of elements of the form (q, g, o, q'), where $g \in \mathcal{B}^+(\mathbb{A}(\mathcal{C}) \cup \Sigma)$ is a Boolean combination of atomic constraints and input letters, $o \in \Gamma$ is an output, and the clock update is removed. We assume that $\Gamma = 2^{AP}$ for a set AP of propositions and freely use propositions to define staying conditions and transition guards.

Let ζ be the run of TA $\mathcal{A}$ on input signal w. A *run* of the DTA reading ζ and w is a finite or infinite sequence of states and transitions, where the states are annotated by the time the DTA stayed in them. Formally, $\zeta' : q_0 \xrightarrow{\delta_0} q_1^{t_1} \xrightarrow{\delta_1} q_2^{t_2} \cdots$, such that for every $i \geq 0$, there are g_i and o_i such that $\delta_i = (q_i, g_i, o_i, q_{i+1})$ and g_i is satisfied by the values $\zeta[\vec{t}_i]$ and $w[\vec{t}_i]$, where $\vec{t}_i = \sum_{j=1}^{i} t_j$. Furthermore, for every $t \in (\vec{t}_i, \vec{t}_{i+1})$ we have $I(q_i)$ is satisfied by $\zeta[t]$ and $w[t]$. Acceptance is defined as for runs of TA. An accepting run ζ' carries the signal w' over $\Sigma \times \Gamma$ such that $w'_\Sigma = w$, $w'[\vec{t}_i]_\Gamma = o_i$, and for all $t \in (\vec{t}_i, \vec{t}_{i+1})$ we have $w'[t]_\Gamma = \gamma(q_i)$. Given a TA run ζ carrying signal w, we denote by $B(w, \zeta)$ the set of signals carried by accepting runs of B.

Consider two DTA $B_i = \langle \Sigma_i, \Gamma_i, Q_i, \mathcal{C}, \gamma_i, I_i, \Delta_i, q_0^i, \mathcal{F}_i \rangle$ for $i \in \{1, 2\}$, where $\Sigma_2 = \Sigma_1 \times \Gamma_1$. The composition $B_1 \otimes B_2$, where B_2 reads the output of B_1, is the following DTA. Let $B_1 \otimes B_2 = \langle \Sigma_1, \Gamma_1 \times \Gamma_2, Q_1 \times Q_2, \mathcal{C}, \gamma, I, \Delta, (q_0^1, q_0^2), \mathcal{F} \rangle$, where $\gamma(q_1, q_2) = (\gamma_1(q_1), \gamma_2(q_2))$ and $\mathcal{F} = \{S \times T \mid S \in \mathcal{F}_1 \text{ and } T = Q_2, \text{ or } S = Q_1 \text{ and } T \in \mathcal{F}_2\}$. The staying condition is $I(q_1, q_2) = I_1(q_1) \wedge simp(\gamma_1(q_1), I_2(q_2))$ where $simp(\gamma_1(q_1), \varphi)$ is the constraint obtained from φ by replacing $\gamma_1(q_1)$ by true and all other letters in Γ_1 by false. The transition relation Δ includes:

- simultaneous transitions $((q_1, q_2), g, (o_1, o_2), (q_1', q_2'))$, where $(q_i, g_i, o_i, q_i') \in \Delta_i$ for $i \in \{1, 2\}$ and $g = g_1 \wedge simp(o_1, g_2)$,
- left-side transitions $((q_1, q_2), g, (o_1, \gamma_2(q_2)), (q_1', q_2))$, where $(q_1, g_1, o_1, q_1') \in \Delta_1$ and $g = g_1 \wedge simp(o_1, \gamma_2(q_2))$, and
- right-side transitions $((q_1, q_2), g, (\gamma_1(q_1), o_2), (q_1, q_2'))$, where $(q_2, g_2, o_2, q_2') \in \Delta_2$ and $g = I_1(q_1) \wedge simp(\gamma_1(q_1), g_2)$.

Lemma 2. *For every run ζ and signal w, $B_1 \otimes B_2(w, \zeta) = B_2(B_1(w, \zeta), \zeta)$.*

Consider a TA $A_1 = \langle \Sigma_1, Q_1, \mathcal{C}, \lambda_1, I_1, \Delta_1, q_0^1, \mathcal{F}_1 \rangle$ and a DTA $B_2 = \langle \Sigma_1, \Gamma_2, Q_2, \mathcal{C}, \gamma_2, I_2, \Delta_2, q_0^2, \mathcal{F}_2 \rangle$. Their composition $A_1 \otimes B_2$ is the TA $\langle \Sigma_1, Q_1 \times Q_2, \mathcal{C}, \lambda, I, \Delta, (q_0^1, q_0^2), \mathcal{F} \rangle$, where $\lambda(q_1, q_2) = \lambda_1(q_1)$, and $\mathcal{F} = \{S \times T \mid S \in \mathcal{F}_1 \text{ and } T = Q_2, \text{ or } S = Q_1 \text{ and } T \in \mathcal{F}_2\}$. The staying condition is $I(q_1, q_2) = I_1(q_1) \wedge simp(\lambda_1(q_1), I_2(q_2))$. The transition relation Δ includes:

- simultaneous transitions $((q_1, q_2), g, \rho_1, (q_1', q_2'))$, where $(q_1, g_1, \rho_1, q_1') \in \Delta_1$, $(q_2, g_2, o_2, q_2') \in \Delta_2$, $g = g_1 \wedge app(\rho_1, simp(\lambda_1(q_1'), g_2))$, and $app(\rho_1, g_2)$ applies the effect of ρ_1 on g_2, e.g., if ρ_1 includes $x := y$ we replace x in g_2 by y,
- left-sided transitions $((q_1, q_2), g, \rho_1, (q_1', q_2))$, where $(q_1, g_1, \rho_1, q_1') \in \Delta_1$ and $g = g_1 \wedge app(\rho_1, simp(\lambda_1(q_1'), \gamma_2(q_2)))$, and
- right-sided transitions $((q_1, q_2), g, \emptyset, (q_1, q_2'))$, where $(q_2, g_2, o_2, q_2') \in \Delta_2$ and $g = I_1(q_1) \wedge simp(\lambda_1(q_1), g_2)$.

$$
\begin{aligned}
\Delta = \{&(q_{in}, \mathit{True}, \emptyset, q_0), (q_{in}, \mathit{True}, x_1 := 0, q_1)\} \cup \\
&\{(q_{2i}, y_1 < f, x_{i+1} := 0, q_{2i+1}), (q_{2i+1}, y_1 < f, y_{i+1} := 0, q_{2i+2})\} \cup \\
&\{(q_{2i+2}, y_1 = f, \{x_1 := x_2, y_1 := y_2, \ldots, x_i := x_{i+1}, y_i := y_{i+1}, \\
&\qquad x_{i+1} := \bot, y_{i+1} := \bot\}, q_{2i})\} \cup \\
&\{(q_{2i+2}, y_1 = f, \{x_1 := x_2, y_1 := y_2, \ldots, x_i := x_{i+1}, y_i := y_{i+1}, \\
&\qquad x_{i+1} := 0, y_{i+1} := \bot\}, q_{2i+1})\} \cup \\
&(q_{2i+3}, y_1 = f, \{x_1 := x_2, y_1 := y_2, \ldots, x_i := x_{i+1}, y_i := y_{i+1}, \\
&\qquad x_{i+1} := \bot, y_{i+1} := \bot\}, q_{2i+1})\} \cup \\
&\{(q_{2i+3}, y_1 = f, \{x_1 := x_2, y_1 := y_2, \ldots, x_{i+1} := x_{i+2}, y_{i+1} := 0, \\
&\qquad x_{i+2} := \bot\}, q_{2i+2})\}
\end{aligned}
$$

Fig. 2. The transition of proposition monitor

Lemma 3. $L(A_1 \otimes B_2) = \{w \mid \exists \zeta_1$ *accepting run of* A_1 *carrying* w *and* $B_2(w, \zeta_1) \neq \emptyset\}$.

As TA have no output, the composition of a TA with a DTA removes the output. Thus, as in the composition $B_1 \otimes B_2$ DTA B_2 may read the output of B_1, the composition $A \otimes B_1 \otimes B_2$ should be read as $A \otimes (B_1 \otimes B_2)$. If A_1 is generalized Büchi with $\mathcal{F}_1 = \{Q_1\}$ and B_1 is parity with $\mathcal{F}_2 = \langle F_0, \ldots, F_k \rangle$ their composition $A_1 \otimes B_2$ is a parity automaton with $\mathcal{F} = \langle F'_0, \ldots, F'_k \rangle$, where $F'_i = Q_1 \times F_i$. We do not define other compositions of parity conditions.

4 MTL to Nondeterministic Timed Automata

We suggest a novel construction for the conversion of MTL formulas to nondeterministic TA. The advantage of this construction is that it effectively distinguishes between discrete guesses relating to occurrences in the future (made by DTA) and the accumulation of knowledge with clocks (made by deterministic and extremely simple TA). This separation allows us to construct a deterministic automaton for the formula in Section 5. This section starts by introducing proposition monitors, deterministic TA that log information about the input. We then show how to construct the DTA that handle general MTL formulas. Note that the number of clocks depends on the structure of the formula through the function fut and the construction of the proposition monitors.

We introduce a TA that memorizes the times in which a proposition is true. Given a formula φ let $f = \mathsf{fut}(\varphi)$. The automaton is going to memorize all events occurring in the interval $[t - f, t)$. Let k be the bounded-variability value (i.e., the limit on the number of changes possible in a proposition in 1 time unit). It follows that in the interval $[t - f, t)$ there can be at most $\lceil \frac{fk}{2} \rceil$ different sub-intervals in which the proposition is true. Thus, we need $2 \cdot \lceil \frac{fk}{2} \rceil$ clocks to memorize their start and end times. Let $n = \lceil \frac{fk}{2} \rceil$. Formally, for a proposition p, let $\mathcal{A}_p = \langle 2^{\{p\}}, Q, \mathcal{C}, \lambda, I, \Delta, q_{in}, \{Q\} \rangle$, where $Q = \{q_{in}, q_0, \ldots, q_{2n}\}$, $\lambda(q_{2i}) = \emptyset$, $\lambda(q_{2i+1}) = \{p\}$, $\mathcal{C} = \{x_1^p, \ldots, x_n^p, y_1^p, \ldots, y_n^p\}$, for $j > 1$ we have $I(q_j) = y_1^p < f$ and $I(q_0) = I(q_1) = \mathit{True}$ and Δ is given in Figure 2. One such proposition monitor is given in Figure 3.

We use the TA $\mathcal{A}_z$ with one state and one clock z, which measures the time since time 0, to check whether the bound f has been reached. Formally, $\mathcal{A}_z = \langle 2^{\emptyset}, \{q_{in}, q\}$,

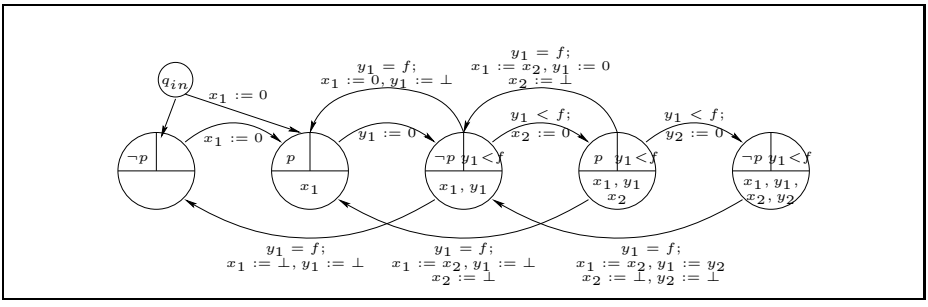


Fig. 3. Proposition monitor for p, where $f = \mathsf{fut}(\varphi)$ and $\lceil \frac{fk}{2} \rceil = 2$. State labels consist of: the label p or $\neg p$; the invariant true (blank) or $y_1 < f$; and clocks active in the state. Transition labels consist of: the transition guard, e.g., $y_1 < f$; and the clock update, e.g., $y_1 := \bot$.

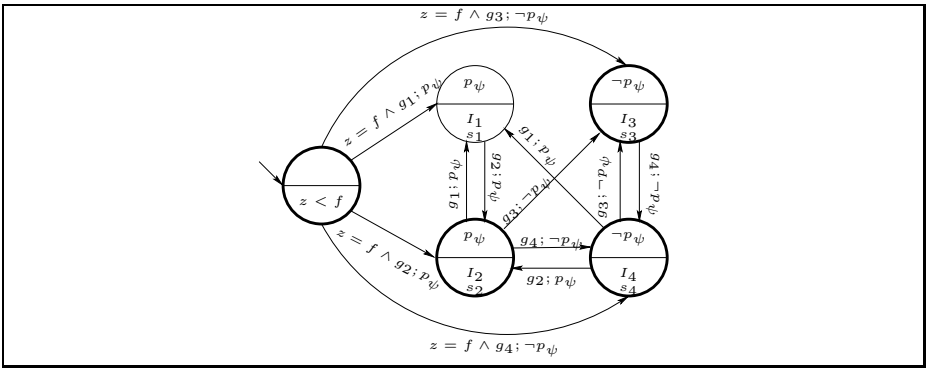


Fig. 4. A DTA for unbounded until. States s_1 and s_3 correspond to $_{(0,a+1)}\varphi_1 \wedge \neg(\varphi_1 \mathcal{U}_{(a,a+1)} \varphi_2)$, state s_2 corresponds to $\varphi_1 \mathcal{U}_{(a,a+1)} \varphi_2$, and state s_4 corresponds to $\neg(\ _{(0,a+1)}\varphi_1) \wedge \neg(\varphi_1 \mathcal{U}_{(a,a+1)} \varphi_2)$.

$\{z\}, \lambda, I, \Delta, q_{in}, \{q\}\rangle$, where $\lambda(q) = \mathit{True}$, $I(q) = z > 0$, and $\Delta = \{(q_{in}, \mathit{True}, \{z := 0\}, q)\}$. Let $\mathcal{A}_m$ denote the composition of all proposition monitors and $\mathcal{A}_z$.

We turn to the construction of DTA. Consider an MTL formula φ. We construct a sequence of DTA that use each other's outputs and eventually are all composed with $\mathcal{A}_m$, which supplies all the clocks that are read by them. For most subformulas ψ of φ, the truth value of ψ can be deduced from the values of clocks in $\mathcal{A}_m$ and the DTA of their subformulas. For such formulas, we define constraints $c_\psi(t)$ that depend on mentioned DTA and $\mathcal{A}_m$ and, intuitively, characterize the truth value of ψ at time t. These constraints are ultimately staying conditions and transition guards in DTA for superformulas of ψ. For an unbounded subformula ψ (i.e., Until where the upper limit is ∞) we construct a small DTA that computes the value of ψ at exactly the time point $\mathsf{fut}(\psi) - \mathsf{fut}(\varphi)$ (i.e., $\mathsf{fut}(\varphi) - \mathsf{fut}(\psi)$ before the current time point). This DTA uses constraints defined for subformulas of ψ. Similar to other formulas, based on the DTA for ψ, we also define the constraint $c_\psi(t)$, which characterizes the truth value of ψ at other times $t \neq \mathsf{fut}(\psi) - \mathsf{fut}(\varphi)$. This constraint is again used for staying conditions and transition guards in superformulas of ψ. Finally, we construct a DTA for the formula φ

itself. Then, we take the composition of all DTA constructed and compose them with $\mathcal{A}_m$ to produce the TA that accepts the language of φ. Formally, we have the following.

The mentioned constraints are in the first-order theory of the reals with clocks of $\mathcal{A}_m$ as free variables. By eliminating quantifiers these constraints can be converted to clock constraints that are used in staying conditions and transition guards. We think about the current time point as 0. For example, if $x_1^r = 2.37$ and $y_1^r = 1.49$, from our point of view r was true during the interval $[-2.37, -1.49)$. Formally, we define by induction the DTA and constraints. For a subformula ψ, the constraint $C_\psi(t)$ is valid for $t \in [-f, -\mathsf{fut}(\psi))$. We assume that quantifier elimination is applied on all constraints.

- For a proposition p and for $t \in [-f, 0)$. We define $C_p(t) = \bigvee_i . - x_i^p \leq t \wedge -y_i^p > t$. The finite disjunction on i depends on the number of clocks in $\mathcal{A}_p$ (part of $\mathcal{A}_m$).
- For subformula ψ of the form $\neg\psi_1$, $\psi_1 \vee \psi_2$, or $\psi_1 \wedge \psi_2$ we define $C_\psi(t)$ using $C_{\psi_1}(t)$ and $C_{\psi_2}(t)$ in the obvious way. The range allowed for t is the minimal range allowed by C_{ψ_1} and C_{ψ_2}. For example, we define $C_{\psi_1 \vee \psi_2}(t) = C_{\psi_1}(t) \vee C_{\psi_2}(t)$.
- For $\psi = \psi_1 \mathcal{U}_I \psi_2$, where $I = (a, b)$ or $I = [b, b]$, by definition $\mathsf{fut}(\psi) = b + max(\mathsf{fut}(\psi_1), \mathsf{fut}(\psi_2))$. It follows that $C_{\psi_1}(t)$ is defined for $t \in [-f, -\mathsf{fut}(\psi_1))$ and $C_{\psi_2}(t)$ is defined for $t \in [-f, -\mathsf{fut}(\psi_2))$. So, for $t \in [-f, -\mathsf{fut}(\psi))$ it is always the case that $t + b$ is in the range where $C_{\psi_1}(t)$ and $C_{\psi_2}(t)$ are defined. In the case that $I = (a, b)$, for $t \in [-f, -\mathsf{fut}(\psi))$ we formally define $C_\psi(t) = \exists t' \in (t + a, t + b).C_{\psi_2}(t') \wedge \forall t'' \in (t, t').C_{\psi_1}(t'')$. In the case that $I = [b, b]$, for $t \in [-f, -\mathsf{fut}(\psi))$ we formally define $C_\psi(t) = C_{\psi_2}(t + b) \wedge \forall t' \in (t, t + b).C_{\psi_1}(t')$.
- Consider a formula $\psi = \psi_1 \mathcal{U}_{(a,\infty)} \psi_2$.
  By definition[2] $\mathsf{fut}(\psi) = a + 2 + max(\mathsf{fut}(\psi_1), \mathsf{fut}(\psi_2))$. It follows that $C_{\psi_1}(t)$ is defined for $t \in [-f, -\mathsf{fut}(\psi_1))$ and $C_{\psi_2}(t)$ is defined for $t \in [-f, -\mathsf{fut}(\psi_2))$. So, for $t \in [-f, -\mathsf{fut}(\psi)]$ it is always the case that $(t, t + a + 2)$ is contained in the range where C_{ψ_1} and C_{ψ_2} are defined. We construct a DTA for ψ and use its output for defining C_ψ.
  - We construct a DTA for the truth value of ψ at time $t = -\mathsf{fut}(\psi)$. Formally, let $\mathcal{B}_\psi$ be the automaton in Figure 4, where the guards and the invariants are as follows.

$$
\begin{array}{ll}
I_1, I_3 & : \forall t \in (-\mathsf{fut}(\psi), -\mathsf{fut}(\psi) + a + 1).C_{\psi_1}(t) \wedge \\
 & \qquad \forall t \in (-\mathsf{fut}(\psi) + a, -\mathsf{fut}(\psi) + a + 1).\neg C_{\psi_2}(t) \\
I_2 & : \exists t \in (-\mathsf{fut}(\psi) + a, -\mathsf{fut}(\psi) + a + 1). \quad C_{\psi_2}(t) \wedge \forall t' \in (-\mathsf{fut}(\psi), t).C_{\psi_1}(t) \\
I_4 & : \exists t(-\mathsf{fut}(\psi), -\mathsf{fut}(\psi) + a + 1).\neg C_{\psi_1}(t) \ \wedge \forall t' \in (-\mathsf{fut}(\psi) + a, t).\neg C_{\psi_2}(t) \\
g_1, g_3 & : I_1 \wedge \exists t \in [-\mathsf{fut}(\psi) + a + 1, -\mathsf{fut}(\psi) + a + 2). \\
 & \qquad \forall t' \in [-\mathsf{fut}(\psi) + a + 1, t).C_{\psi_1}(t') \wedge \neg C_{\psi_2}(t') \\
g_2 & : I_2 \vee (I_1 \wedge \exists t \in (-\mathsf{fut}(\psi), -\mathsf{fut}(\psi) + 1). \\
 & \quad \forall t' \in (-\mathsf{fut}(\psi), t).\exists t'' \in (t' + a, t' + a + 1).C_{\psi_2}(t'') \wedge \forall t''' \in (t, t'')C_{\psi_1}(t''') \\
g_4 & : I_4 \vee (I_3 \wedge \exists t \in (-\mathsf{fut}(\psi), -\mathsf{fut}(\psi) + 1).\forall t' \in (-\mathsf{fut}(\psi), t). \\
 & \qquad \exists t'' \in (t', t' + a + 1).\neg C_{\psi_1}(t'') \wedge \forall t''' \in (t' + a, t'')\neg C_{\psi_2}(t''')
\end{array}
$$

    States s_1, s_2 and their incoming transitions are labeled by p_ψ, all other states and transitions by $\neg p_\psi$, and s_1 is the only unfair state.

[2] We note that replacing $a + 2$ by $a + 2\epsilon$ for every $\epsilon > 0$ would not affect the correctness of the construction. We choose integer values for the lookaheads to avoid normalizing the largest constant in the guards and invariants of the timed automaton

Intuitively, the invariants realize the intuitive meaning of the states as explained in Figure 4. The transition guards, in addition, use the extra 1 in order to make sure that when going to the next state it is possible to *stay* in it (eliminate 0-duration errors). This is required in order to be able to apply determinization. In case that determinization is not pursued, the extra lookahead can be removed and the guards are I_1 for transitions into s_1 and s_3, $I_2 \vee I_1$ for transitions into s_2, and $I_4 \vee I_3$ for transitions into s_4.

- For every $t \in [-\mathsf{fut}(\varphi), -\mathsf{fut}(\psi)]$ the constraint $C_\psi(t)$ that describes the truth value of ψ is formally defined as follows:

$$(t = -\mathsf{fut}(\psi) \wedge p_\psi) \vee (t < -\mathsf{fut}(\psi) \wedge p_\psi \wedge \forall t' \in (t, -\mathsf{fut}(\psi)].C_{\psi_1}(t)) \vee$$
$$(t < -\mathsf{fut}(\psi) \wedge \exists t' \in (t + a, -\mathsf{fut}(\psi)).C_{\psi_2}(t') \wedge \forall t'' \in (t, t').C_{\psi_1}(t))$$

This completes the inductive part of the construction[3]. Let $\psi_1, \ldots, \psi_n$ be all the unbounded temporal operators in φ such that if ψ_i is a subformula of ψ_j then $i < j$. Let $\mathcal{B}_{\psi_1}, \ldots, \mathcal{B}_{\psi_n}$ be the DTA constructed in the inductive part, $AP' = \{p_{\psi_1}, \ldots, p_{\psi_n}\}$, and $\Gamma = 2^{AP'}$. That is, Γ includes the output of all DTA constructed by induction. Note that for every i we have $\mathcal{B}_{\psi_i}$ is merely informative. That is, $\mathcal{B}_{\psi_i}$ accepts all inputs and adorns them with the proposition p_{ψ_i}.

We now construct a final DTA $\mathcal{B}$ for φ itself. Let $\mathcal{B}$ be $\langle \Sigma \times \Gamma, \{o\}, \{q_0, q_1, q_2\}, \mathcal{C}, \gamma, I, \Delta, q_0, \{\{q_1\}\}\rangle$, where $\gamma(q_0) = \gamma(q_1) = \gamma(q_2) = o$, $I(q_0) = z < f$, and $I(q_1) = I(q_2) = \mathit{True}$. The transition relation is $\Delta = \{(q_0, g_1, o, q_1), (q_0, g_2, o, q_2)\}$, where $g_1 = C_\varphi(-f) \wedge z = f$ and $g_2 = \neg C_\varphi(-f) \wedge z = f$. That is, $\mathcal{B}$ enters state q_1 if φ is true at time 0 and state q_2 if φ is false at time 0. State q_1 is an accepting sink state and state q_2 is a rejecting sink state. Let $\mathcal{B}_\varphi = \mathcal{B}_{\psi_1} \otimes \cdots \otimes \mathcal{B}_{\psi_n} \otimes \mathcal{B}$. Finally, the TA for φ, denoted by A_φ, is $\mathcal{A}_m \otimes \mathcal{B}_\varphi$.

Lemma 4. $L(\mathcal{A}_\varphi) = L(\varphi)$

Corollary 1. *For every* MTL *formula φ with m propositions, n unbounded temporal operators, and inputs of bounded variability k, there exists a nondeterministic timed automaton with $2m\lceil\frac{k\cdot\mathsf{fut}(\varphi)}{2}\rceil + 1$ clocks and $((2\lceil\frac{k\cdot\mathsf{fut}(\varphi)}{2}\rceil + 1)^m + 1)(2 \cdot 4^n + 1)$ states that accepts the language of φ.*

5 Deterministic Timed Automata

We show that the automata constructed in Section 4 can be determinized. The separation of the construction to deterministic proposition monitors and nondeterministic DTA makes this part possible. We use a variant of Piterman's version of Safra's determinization [23] to determinize DTA. The differences are mostly syntactic, taking into account the 'asynchronicity' of transitions from a set of states. We assume that every transition (q, g, o, q') of a DTA the intersection of g and $I(q)$ is either empty or isolated, i.e., there does not exist an open interval (t, t') such that $(t, t') \subseteq I(q)$ and $(t, t') \cap g \neq \emptyset$ and if (q, g, o, q') is enabled at time t then there is a small interval (t, t')

[3] Due to the lack of space, we restrict attention to intervals of the form $[b, b]$, (a, b) and (a, ∞) and include a full treatment of all other types of intervals in the full version of the paper.

such that $(t, t') \subseteq I(q')$. The DTA from Section 4 satisfy this condition. We also assume that the DTA takes infinitely many steps and that its acceptance set $\mathcal{F}$ contains exactly one set. It is simple to modify the automaton to an automaton that takes infinitely many steps by adding (or reusing) a clock that keeps resetting itself and taking a transition whenever this clock reaches some value. Converting an automaton where $|\mathcal{F}| > 1$ to an automaton where $|\mathcal{F}| = 1$ is standard. We make this assumption solely to simplify presentation.

Let $B = \langle \Sigma, \Gamma, Q, \mathcal{C}, \gamma, I, \Delta, q_0, \mathcal{F} \rangle$ be a DTA satisfying these conditions. We construct an 'equivalent' DTA D. The construction is based on the subset construction [25]. Thus, every state of D is associated with a set of states of B. A state of D associated with $Q' \subseteq Q$ follows a set of runs of B ending in the states Q'. For a set $Q' \subseteq Q$ let $I(Q')$ be $\bigwedge_{q \in Q'} I(q)$ and $\overline{I}(Q')$ be $\bigwedge_{q \in Q'} \neg I(q)$. Let $\Delta(Q') = \{(q, g, o, q') \in \Delta \mid q \in Q'\}$. For a set $\Delta' \subseteq \Delta$ let $g(\Delta')$ be $\bigwedge_{(q,g,o,q') \in \Delta'} g$ and $\overline{g}(\Delta')$ be $\bigwedge_{(q,g,o,q') \in \Delta'} \neg g$. A set of runs of B that end in states in Q' can be extended in three different ways: Some runs are extended by staying in the same state, some runs are extended by crossing discrete transitions (and cannot be extended by crossing other discrete transitions), and some runs cannot be extended. We represent such a choice by a set $T \subseteq Q' \cup \Delta(Q')$. Let $stay(T) = T \cap Q'$ be the states whose runs are extended by staying in the same state. In particular, all transitions from $stay(T)$ have to be disabled. Let $\Delta(T) = \Delta(Q') \cap T$ be the discrete transitions taken by states in Q. Let $deadend(Q', T)$ be the set of states $q \in Q'$ such that $q \notin T$ and for every $(q', g, o, q'') \in T$ we have $q' \neq q$. That is, the states whose runs cannot be extended. Let $move(T) = Q' \setminus (stay(T) \cup deadend(T))$, the set of states that have some transitions going out of them in T. Let $succ(Q'', T)$ be $stay(T) \cap Q'' \cup \{q \mid \exists (q', g, o, q) \in T \cap \Delta(Q'')\}$. For every T as above, a state of D associated with Q' can take a T-transition with guard $g(T)$:

$$I(stay(T)) \wedge \overline{g}(\Delta(stay(T))) \wedge g(T \cap \Delta) \wedge \overline{g}(\Delta(move(T)) \setminus T) \wedge \overline{I}(deadend(Q', T)) \wedge \overline{g}(\Delta(deadend(Q', T)))$$

That is, states whose run is extended by staying in the same state contribute their invariants and the negation of transitions exiting them; transitions that are crossed contribute their guards and the negation of transitions that are not crossed; and states whose run is going to end contribute the negation of their invariants and the guards of transitions exiting them. The case when both $deadend(T)$ and $T \cap \Delta$ are empty is not interesting as all runs are extended.

Definition 4. [23] *A* compact Safra tree t *over* Q *is* $\langle N, M, 1, p, l, e, f \rangle$*, where the components of* t *are as follows. Let* $|Q| = n$*. (a)* $N \subseteq [n]$ *is a set of nodes. (b)* $1 \in N$ *is the root node. (c)* $p : N \to N$ *is the parent function. (d)* $l : N \to 2^Q$ *is a labeling of the nodes with subsets of* Q*. The label of every node is a proper superset of the union of the labels of its sons. The labels of two siblings are disjoint. (e)* $e, f \in [n+1]$ *are used to define the parity acceptance conditions. For a tree* t*, let* $set(t) = \bigcup_{i \in [n]} l(i)$ *be the set of states that label at least one node in* t*.*

The deterministic DTA is $D = \langle \Sigma, \{o\}, S, \mathcal{C}, \gamma', I', \Delta', s_0, \alpha \rangle$, where components are defined as follows. Differences from the construction in [23] are in bold.

- S is the set of compact Safra trees over Q.
- s_0 is the tree with a single node 1 labeled $\{q_0\}$, where $e = 2$ and $f = 1$.

- The parity acceptance condition $\alpha = \langle F_0, \ldots, F_{2n-1} \rangle$, where $F_{2i} = \{s \in S \mid f = i+1 \text{ and } e > f\}$ and $F_{2i+1} = \{s \in S \mid e = i+2 \text{ and } f \geq e\}$.
- **For every state $s \in S$ we have $\gamma'(s) = o$ and $I'(s) = I(set(s))$.**
- For every tree $s \in S$ and **every set $T \subseteq set(s) \cup \Delta(set(s))$ we add to Δ' the transition $(s, g(T), o, s')$** where s' is obtained from s using the following transformations.
  1. For every node v with label Q' replace Q' by $succ(Q', T)$.
  2. For every node v with label Q' such that $Q' \cap \alpha \neq \emptyset$, create a new son $v' \notin N$ of v. Set its label to $Q' \cap \alpha$. Set its name to the minimal value greater than all used names. We may have to use temporarily names in the range $[(n+1)..(2n)]$.
  3. For every node v with label Q' and state $q \in Q'$ such that q belongs also to some sibling v' of v such that $M(v') < M(v)$, remove q from the label of v and all its descendants.
  4. For every node v whose label is equal to the union of the labels of its children, remove all descendants of v. Call such nodes *green*. Set f to the minimum of $n+1$ and all green nodes. Notice that no node in $[(n+1)..(2n)]$ can be green.
  5. Remove all nodes with empty labels. Set e to the minimum of $n+1$ and all nodes removed during all stages of the transformation.
  6. Let Z denote the set of nodes removed during all previous stages of the transformation. For every node v let $rem(v)$ be $|\{v' \in Z \mid M(v') < M(v)\}|$. That is, we count how many nodes that are smaller than v are removed during the transformation. For every node v such that $l(v) \neq \emptyset$ we change v to $M(v) - rem(v)$.

Theorem 1. *For every deterministic timed automaton A, we have $A \otimes D$ is deterministic and $L(A \otimes D) = L(A \otimes B)$.*

Corollary 2. *For every* MTL *formula φ with m propositions, n unbounded temporal operators, and inputs of bounded variability k, there exists a deterministic timed automaton with $2m\lceil \frac{k \cdot \mathsf{fut}(\varphi)}{2} \rceil + 1$ clocks and $((2\lceil \frac{k \cdot \mathsf{fut}(\varphi)}{2} \rceil)^m + 1) \cdot 2^{2^{O(n \log n)}}$ states that accepts the language of φ.*

We note that the double exponent in the number of unbounded temporal operators is unavoidable, as follows from the same for LTL.

6 Conclusions

We developed a novel construction for translating full MTL to timed automata, under bounded variability assumptions. Our construction provides a unified framework for model checking, runtime monitoring, and controller synthesis, and offers an alternative translation that improves exponentially on the complexity of securing a deterministic timed automaton, avoiding a doubly exponential number of clocks.

In the future, we intend to investigate further improvements of our construction:

- Consider MTL with past operators. This extension does not increase the complexity of the construction as satisfaction of past operators depends only on the observation of memorized events in the proposition monitors.

– Interpret the logic over finite signals, in the context of monitoring timed behaviors.
– Optimize and improve the translation. One straightforward improvement would require a smarter memorization of events in the proposition monitors.
– Implement the translation presented in this paper and evaluate it in the context of model checking, runtime monitoring, and controller synthesis.

Acknowledgements. We would like to thank Dana Fisman and Oded Maler for their insightful suggestions that helped us improve the clarity of the manuscript.

References

1. Alur, R.: Timed automata. In: Halbwachs, N., Peled, D.A. (eds.) CAV 1999. LNCS, vol. 1633, pp. 8–22. Springer, Heidelberg (1999)
2. Asarin, E., Caspi, P., Maler, O.: Timed regular expressions. JACM 49(2), 172–206 (2002)
3. Alur, R., Dill, D.: A theory of timed automata. TCS 126(2), 183–236 (1994)
4. Alur, R., Feder, T., Henzinger, T.A.: The benefits of relaxing punctuality. JACM 43(1), 116–146 (1996)
5. Alur, R., Henzinger, T.A.: Logics and models of real time: a survey. In: Huizing, C., de Bakker, J.W., Rozenberg, G., de Roever, W.-P. (eds.) REX 1991. LNCS, vol. 600, pp. 74–106. Springer, Heidelberg (1992)
6. Asarin, E., Maler, O., Pnueli, A., Sifakis, J.: Controller synthesis for timed automata. In: IFAC SSSC, pp. 469–474. Elsevier, Amsterdam (1998)
7. Baier, C., Bertrand, N., Bouyer, P., Brihaye, T.: When are Timed Automata Determinizable? In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009. LNCS, vol. 5556, pp. 43–54. Springer, Heidelberg (2009)
8. Bouyer, P., Bozzelli, L., Chevalier, F.: Controller Synthesis for MTL Specifications. In: Baier, C., Hermanns, H. (eds.) CONCUR 2006. LNCS, vol. 4137, pp. 450–464. Springer, Heidelberg (2006)
9. Behrmann, G., Cougnard, A., David, A., Fleury, E., Larsen, K., Lime, D.: UPPAAL-Tiga: Time for playing games! In: Damm, W., Hermanns, H. (eds.) CAV 2007. LNCS, vol. 4590, pp. 121–125. Springer, Heidelberg (2007)
10. Doyen, L., Geeraerts, G., Raskin, J.-F., Reichert, J.: Realizability of real-time logics. In: Ouaknine, J., Vaandrager, F.W. (eds.) FORMATS 2009. LNCS, vol. 5813, pp. 133–148. Springer, Heidelberg (2009)
11. Havelund, K., Rosu, G.: Efficient monitoring of safety properties. In: STTT (2004)
12. Henzinger, T.A.: It's about time. In: Sangiorgi, D., de Simone, R. (eds.) CONCUR 1998. LNCS, vol. 1466, pp. 439–454. Springer, Heidelberg (1998)
13. Henzinger, T.A., Nicollin, X., Sifakis, J., Yovine, S.: Symbolic model checking for real-time systems. I&C 111, 193–244 (1994)
14. Koymans, R.: Specifying real-time properties with metric temporal logic. Real-time Systems 2(4), 255–299 (1990)
15. Larsen, K.G., Petterson, P., Yi, W.: UPPAAL: Status & developments. In: Grumberg, O. (ed.) CAV 1997. LNCS, vol. 1254, pp. 456–459. Springer, Heidelberg (1997)
16. Maler, O., Nickovic, D., Pnueli, A.: Real time temporal logic: Past, present, future. In: Pettersson, P., Yi, W. (eds.) FORMATS 2005. LNCS, vol. 3829, pp. 2–16. Springer, Heidelberg (2005)
17. Maler, O., Nickovic, D., Pnueli, A.: From MITL to timed automata. In: Asarin, E., Bouyer, P. (eds.) FORMATS 2006. LNCS, vol. 4202, pp. 274–289. Springer, Heidelberg (2006)

18. Maler, O., Nickovic, D., Pnueli, A.: On synthesizing controllers from bounded-response properties. In: Damm, W., Hermanns, H. (eds.) CAV 2007. LNCS, vol. 4590, pp. 95–107. Springer, Heidelberg (2007)
19. Monniaux, D.: A quantifier elimination algorithm for linear real arithmetic. In: Cervesato, I., Veith, H., Voronkov, A. (eds.) LPAR 2008. LNCS (LNAI), vol. 5330, pp. 243–257. Springer, Heidelberg (2008)
20. Manna, Z., Pnueli, A.: Temporal Verification of Reactive Systems: Specification. Springer, Heidelberg (1991)
21. Maler, O., Pnueli, A., Sifakis, J.: On the synthesis of discrete controllers for timed systems. In: Mayr, E.W., Puech, C. (eds.) STACS 1995. LNCS, vol. 900, Springer, Heidelberg (1995)
22. Ouaknine, J., Worrell, J.: On the decidability of metric temporal logic. In: LICS, pp. 188–197 (2005)
23. Piterman, N.: From nondeterministic Büchi and Streett automata to deterministic parity automata. LMCS 3(3), 5 (2007)
24. Pnueli, A., Rosner, R.: On the Synthesis of a Reactive Module. In: POPL, pp. 179–190 (1989)
25. Rabin, M.O., Scott, D.: Finite automata and their decision problems. IBM Journal of R&D 3, 115–125 (1959)
26. Safra, S.: On the complexity of ω-automata. In: FOCS, pp. 319–327 (1988)
27. Vardi, M.Y., Wolper, P.: An Automata-theoretic Approach to Automatic Program Verification. In: LICS, pp. 322–331 (1986)
28. Wilke, T.: Specifying Timed State Sequences in Powerful Decidable Logics and Timed Automata. In: Langmaack, H., de Roever, W.-P., Vytopil, J. (eds.) FTRTFT 1994 and ProCoS 1994. LNCS, vol. 863, pp. 694–715. Springer, Heidelberg (1994)
29. Yovine, S.: Kronos: A verification tool for real-time systems. STTT 1(1-2), 123–133 (1997)

Unambiguity in Timed Regular Languages: Automata and Logics

Paritosh K. Pandya* and Simoni S. Shah

Tata Institute of Fundamental Research, Colaba, Mumbai *400005*, India
pandya@tifr.res.in

Abstract. Unambiguous languages (UL), originally defined by Schutzenberger using unambiguous polynomials, are a robust subclass of regular languages. They have many diverse characterizations: they are recognized by partially-ordered two-way deterministic automata (*po2dfa*), they are definable by Unary Temporal Logic (UTL) as also by the two variable first-order logic over words ($FO^2[<]$).

In this paper, we consider the timed version of unambiguous languages. A subclass of the two-way deterministic timed automata (*2DTA*) of Alur and Henzinger, called *partially-ordered two-way deterministic automata* (*po2DTA*) are examined and we call the languages accepted by these as *Timed Unambiguous Languages* (TUL). This class has some interesting properties: we show that *po2DTA* are boolean closed and their non-emptiness is NP-Complete. We propose a deterministic and unary variant of MTL called *DUMTL* and show that *DUMTL* formulae can be reduced to language equivalent *po2DTA* in polynomial time, giving NP-complete satisfiability for the logic. Moreover, *DUMTL* is shown to be expressively complete for *po2DTA*. Finally, we consider the unary fragments of well known logics MTL and MITL and we show that neither of these are expressively equivalent to *po2DTA*. Contrast this with the untimed case where unary temporal logic is equivalent to *po2dfa*.

1 Introduction

Regular languages and their automata are a well established formalism which have made considerable impact on techniques for modelling and verification of systems. The connection between temporal logics and finite automata provides the key to algorithmic analysis and reasoning about logical properties of reactive systems.

Attempts to extend the above paradigm to timed logics and timed automata has proved challenging. One impediment is that logics have boolean operations and the timed automata (of Alur-Dill) are not closed under complementation. Thus, one must consider some variant/subclass of the timed automata which are boolean closed. The classical timed logic, MTL (interpreted over timed words), is undecidable and has no automaton characterization. This logic uses time constrained until and since operators U_I and S_I where I is a time interval with

* Corresponding author.

K. Chatterjee and T.A. Henzinger (Eds.): FORMATS 2010, LNCS 6246, pp. 168–182, 2010.

integral endpoints: we denote this by $MTL[\mathsf{U}_I, \mathsf{S}_I]$. Various subclasses of MTL have been investigated for decidablity. Ouaknine and Worrell have shown [OW05] that over finite timed words the logic $MTL[\mathsf{U}_I]$ is decidable by reducing the logic to 1-clock alternating timed automata which are boolean closed. The emptiness of 1-clock ATA is decidable with non-primitive-recursive complexity.

An altogether different approach is followed by Alur, Feder and Henzinger who consider logic MTL with only non-punctual (non-singleton) intervals in the modalities U_I and S_I. The resulting logic is called $MITL$ [AFH96]. In their pioneering work called "Back to the Future", Alur and Henzinger have defined the notion of *two-way deterministic timed automata* (*2DTA*) and shown that the subclass of reversal-bounded *2DTA* is boolean closed. The emptiness of *2DTA* with bounded number of reversals is PSPACE-complete. Alur and Henzinger also give a reduction from logic $MITL[\mathsf{U}_I, \mathsf{S}_I]$ to reversal bounded *2DTA*. Using this they show that the satisfiability of the logic is EXPSPACE-complete. Real-time Temporal Logics with low decision complexities are a rarity.

In this paper, we consider a subclass of the reversal-bounded *2DTA* of Alur and Henzinger called *partially-ordered two-way deterministic timed automata* (*po2DTA*). The automaton is partially ordered in the sense that the underlying graph is acyclic upto self loops. By restricting to this subclass, we show that we can achieve better complexities for the decision problems for the automata and their logics. We call the languages accepted by these automata as *timed unambiguous languages* (TUL). In this nomenclature, we are motivated by the unambiguous languages (UL), originally proposed by Schutzenberger [Sch75]. Schwentick *et al* showed that UL is precisely the subclass of regular languages that can be recognized by partially-ordered two-way deterministic finite automata (*po2dfa*) [STV01]. Moreover, Etessami *et al* [EVW02] showed that UL are precisely the languages definable by the Unary Temporal Logic, UTL, which is linear temporal logic using only the modalities $\overset{\frown}{\Diamond}$ and $\overset{\frown}{\Diamond}$. In our recent work, we proposed an unambiguous interval temporal logic, *UITL*, which also specifies exactly the class UL [LPS08]. One feature of this deterministic (or unambiguous) logic is that every model (word) can be "uniquely parsed" to match the formula. The survey article by Diekert *et al* gives a comprehensive treatment of UL [DGK08].

In this paper, we explore the properties of *po2DTA* and show that these automata are closed under the operations of union, intersection and complementation. In fact, each of these operations only results in a linear blowup in the size of the automaton. We also show that non-emptiness of a *po2DTA* is decidable and is NP-Complete. Membership in NP is based on showing that every *po2DTA* with non-empty language has a "small-sized" timed word in its language which uses only a limited set of integral and fractional values in the time stamps. It is possible to guess this word non-deterministically and to simulate the *po2DTA* in time polynomial in the number of states to check for non-emptiness. Contrast this with k-reversal bounded *2DTA* for which emptiness is PSPACE-complete for fixed k. It follows that the language inclusion of *po2DTA* is Co-NP-Complete.

As our next key exploration, we consider logics for specifying timed unambiguous languages. We propose a variant of MTL called $DUMTL$ (Deterministic Unary MTL) where U_I and S_I are replaced by "deterministic" and "unary" operators $\xi\mathsf{U}^x_\theta$ and $\xi\mathsf{S}^x_\theta$. A guarded event θ has the form (a, g) where guard g puts constraints on the time of occurrence of a relative to other events in the word. The exact form of guards is defined in Section 2. Let ξ be a finite set of guarded events. Then formula $\xi\ \mathsf{U}^x_\theta\ \phi$ holds at position i in a timed word provided j is the *first* position strictly after i with letter a satisfying the time constraint g and the formula ϕ holds at j with a freeze variable x remembering the time stamp of j. Moreover, all the positions strictly inbetween i and j must have events matching the set ξ. Note that ξ is a set of guarded events and not a formula. In this sense, the operator $\xi\mathsf{U}^x_\theta$ is unary and deterministic (requiring match with first occurrence of θ). The operator $\xi\mathsf{S}^x_\theta$ is the past (mirror image) version of $\xi\mathsf{U}^x_\theta$ and defined similarly. We show that every formula of logic $DUMTL$ can be reduced to *po2DTA* in linear time. Hence, satisfiability of $DUMTL$ is NP-complete. We also show that language of every *po2DTA* can be specified by a $DUMTL$ formula. The logic $DUMTL$ is deterministic or unambiguous in the sense that every timed word can be uniquely parsed to match a formula.

Next we explore expressiveness of various timed logics. The contrast between timed and untimed case should be noted. In untimed case, expressively, $UTL \equiv UITL \equiv$ *po2dfa*. In order to complete the picture, we consider the unary version of the well known logic MTL and call it *UMTL*. By an example, we show that there are languages definable by *UMTL* which are not *po2DTA* recognizable. We also consider the unary fragment of logic MITL and call it *UMITL*. Again, by an example we show that there are *po2DTA* recognizable languages which are not definable using *UMITL*. Thus, unambiguity in timed regular languages seems to be captured only by the rather esoteric timed logic $DUMTL$ which is expressively distinct from the other timed logics listed above.

The rest of the paper is organized as follows. Section 2 defines the *po2DTA* and investigates their properties. Section 3 gives the unambiguous unary metric temporal logic, $DUMTL$ and shows its NP-complete satisfiability. Finally, Section 4 compares the expressiveness of unary fragments of MTL and $MITL$ with *po2DTA*. The paper ends with a short discussion.

2 Partially Ordered 2-Way Deterministic Timed Automata

Let Σ be a finite alphabet. A *timed word* w over Σ is a finite sequence $w = (\sigma_1, \tau_1), (\sigma_2, \tau_2)...(\sigma_m, \tau_m)$ of symbols $\sigma_i \in \Sigma$ paired with non-negative real numbers $\tau_i \in R$, such that the sequence $\tau_1, ...\tau_m$ is weakly monotonically increasing. We shall often represent w as $(\overline{\sigma}, \overline{\tau})$ with $\overline{\sigma} = \sigma_1, ...\sigma_m$ and $\overline{\tau} = \tau_1, ...\tau_m$ and let $untime((\overline{\sigma}, \overline{\tau})) = \overline{\sigma}$. For any real number τ let $Int(\tau)$ and $Fra(\tau)$ be the integral and fractional parts of τ. The set of all finite timed words over an alphabet Σ is denoted by $T\Sigma^*$. A timed language is a subset of $T\Sigma^*$. For 2-way automata it is convenient to enclose a timed word w between end markers. Let

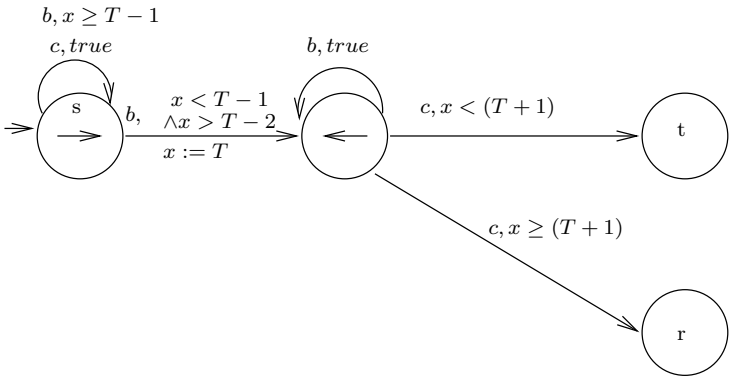


Fig. 1. Example of po2DTA

$\hat{w} = (\triangleright, 0)w(\triangleleft, \tau_{|w|})$ and let $\Sigma' = \Sigma \cup \{\triangleright, \triangleleft\}$. Also, let $dom(\hat{w}) = 0, 1, ...(|w| + 1)$ be the set of all positions in the word $\hat{w}$. When clear from context, we shall abbreviate $\hat{w}$ by w.

A *two way finite state automaton* has a read-only head which scans the input word by reading the letter at its current head position i and the head can move in both directions (as in Turing machines). The textbook by Kozen [Koz97] provides a readable account of two way deterministic finite automata and their equivalence to 1-way DFA. Alur and Henzinger [AH92] generalized this to *2-way determinstic timed automata*, *2DTA*, which work over timed words. These timed automata are equipped with a finite set C of clocks (or registers) and in each transition a subset X of these can be reset to the value of the current time stamp under the head. The clocks retain their value till reset, thus they are like registers. As in timed automata, each transition is labelled with a letter and a guard. A guard g specifies a constraint on the values of the clocks and the current timestamp T. We label each state with a direction (left or right) and the head moves in direction specified by the *target* state of the transition. This form is more convenient for reduction to logics. (It can be shown that this form of automata are equivalent with a linear blowup in size to automata where the transitions carry the direction of head movement as in [AH92].) Each automaton is total, deterministic (see the formal definition below) and partially ordered. By *partial ordering of states* we mean that the only loops allowed are self-loops and once the automaton exits a state it can never return to it.

Example 1. Figure 1 shows an example po2DTA. Notation $x := T$ denotes that clock x is reset to current time stamp. This automaton accepts timed words with the following property: There is b in the interval $(1, 2)$ and c occurs before it. Moreover, if j is the position of the first b in the interval $(1, 2)$ and k is the position of the last c before it then $\tau_j - \tau_k < 1$.

We now give a formal definition of the syntax and semantics of *po2DTA*. Let G_C denote the set of guards over the clock set C. Let x range over C and c over

integer constants. Let $\sim \in \{<, >, \leq, \geq, =\}$. The distinguished variable T denotes the current time. A guard g has the syntax:

$$g \in G_C \ ::= \ x \sim T + c \ \mid \ g_1 \wedge g_2 \ \mid \ \neg g$$

As usual, a clock valuation $\nu : C \rightarrow \Re_0$ assigns to each clock a non-negative real number. Let $\nu, \tau \models g$ denote that ν satisfies the guard g when T is assigned the real value τ. Also, let $\nu' = \nu \otimes (x \rightarrow \tau)$ denotes a valuation such that $\forall y \in C \ . \ y \neq x \Rightarrow \nu'(y) = \nu(y)$ and $\nu'(x) = \tau$. Two guards g_1 and g_2 are said to be disjoint if for all valuations ν and all τ, we have $\nu, r \models \neg(g_1 \wedge g_2)$. A special valuation ν_{init} maps all clocks to 0.

Definition 1 (Syntax of *po2DTA*). *A po2DTA over alphabet Σ is a tuple $\mathcal{M} = (Q, \leq, \delta, s, t, r, C)$ where $(Q, \leq)$ is a partially ordered and finite set of states such that r, t are the only minimal elements and s is the only maximal element. Here, s is the initial state, t the accept state and r the reject state. Set C is a finite set of clocks. The set $Q \setminus \{t, r\}$ is partitioned into Q_l and Q_r (making the head move resp. left and the right on transitions leading into them) with $s \in Q_r$. Function $\delta : ((Q_l \cup Q_r) \times \Sigma' \times G_C) \rightarrow Q \times 2^C)$ is the transition function which satisfies the following conditions: Let $\delta(q, a, g) = (q', X)$. Then, $X \subseteq 2^C$ specifies the subset of clocks to be reset to the current time stamp. Moreover,*

- *(Partial-order) $q' \leq q$.*
- *(Reset on progress) If $q' < q$ (i.e. $q \neq q'$) then the transition is called a* progress transition *(or progress edge). A transition with $q = q'$ is called a self-loop. We allow resets only on progress edges, i.e. $X = \emptyset$ if $q = q'$.*
- *If $a = \triangleleft$ then $q' \in Q_l$ and if $a = \triangleright$ then $q \in Q_r$. This prevents the head from falling off the end-markers.*
- *(Determinism) For all $q \in Q$ and $a \in \Sigma'$, if there exist distinct transitions $\delta(q, a, g_1) = (q_1, X_1)$ and $\delta(q, a, g_2) = (q_2, X_2)$, then g_1 and g_2 are disjoint.*

Definition 2 (Run). *Let $w = (\sigma_1, \tau_1), (\sigma_2, \tau_2)...(\sigma_m, \tau_m)$ be a given timed word. The automaton actually runs on the word $\hat{w}$ which is w enclosed in end markers, as defined before. The configuration of a po2DTA at any instant is given by (q, ν, l) where q is the current state, ν is the the current clock valuation and $l \in dom(\hat{w})$ is the current head position. In this configuration, the head reads the letter σ_l and the time stamp τ_l.*

The run ρ of a po2DTA on the timed word w with starting head position k and starting clock valuation ν is the (unique) sequence of configurations $(q_1, \nu_1, l_1), ..., (q_n, \nu_n, l_n)$ such that

- *Initialization: $q_1 = s$, $l_1 = k$ and $\nu_1 = \nu$. The automaton always starts in the initial state s.*
- *If the automaton is in a configuration (q_i, ν_i, l_i) and there exists a (unique) transition $\delta(q_i, a, g) = (p, X)$ such that $\sigma_{l_i} = a$ and $\nu_i, \tau_{l_i} \models g$. Then,*
  - *$q_{i+1} = p$*
  - *$\nu_{i+1}(x) = \tau_{l_i}$ for all clocks $x \in X$, and $\nu_{i+1}(x) = \nu_i(x)$ otherwise.*

- $l_{i+1} = l_i - 1$ *if* $p \in Q_l$
  $l_{i+1} = l_i + 1$ *if* $p \in Q_r$
  $l_{i+1} = l_i$ *if* $p \in \{t, r\}$

– *Termination:* $q_n \in \{t, r\}$. *The run is accepting if* $q_n = t$ *and rejecting if* $q_n = r$. *The configuration* (q_n, l_n, ν_n) *is called the final configuration and the head position* l_n *is called the accepting/rejecting position.*

Let $\mathcal{F}_{\mathcal{M}}$ *be a function such that* $\mathcal{F}_{\mathcal{M}}(w, i, \nu)$ *gives the final configuration* (q_n, l_n, ν_n) *of the unique run of* $\mathcal{M}$ *on* w *starting with the configuration* (s, i, ν). *The language accepted by an automaton* $\mathcal{M}$ *is given by:*

$$\mathcal{L}(\mathcal{M}) = \{w \mid \mathcal{F}_{\mathcal{M}}(w, 1, \nu_{init}) = (t, i, \nu), \textit{for some } i, \nu\}.$$

2.1 Properties of *po2DTA*

We now discuss the properties of *po2DTA*. One nice property of our *po2DTA* is that all runs on a given word are of bounded length and the automaton cannot loop for ever.

Reversal Bounding. A reversal refers to the change in the direction of the head movement within a run of the automaton. Recall that in *po2DTA*, the head movement direction is determined by the direction of the target state of a transition. Hence, for a *po2DTA*, a reversal corresponds to a progress transition from a Q_l to a Q_r state, or vice versa: reversal can occur only on progress transitions. Due to partial ordering, we can have at most $(n-1)$ progress transitions during the run where $|Q| = n$, the number of states. Hence the number of reversals is also at most $(n-1)$. Moreover, the run on a word $\hat{w}$ can be at most of length $|w| \times (n-1)$.

Number of Clocks. Due to the partial order on the states, each progress edge in a *po2DTA* is traversed at most once in a given run of the automaton. Since a clock may be reset only on progress edges, there can be only $n-1$ number of resets on any run of the automaton. Hence, the number of clocks of a *po2DTA* can upper-bounded by $n-1$. Any automaton with more clocks will have clocks which are always equal and these can be removed. We now assume that the automaton has at most $n-1$ clocks.

Composition of Automata. We define some useful constructions on *po2DTA*. These also establish the closure properties of the language class *po2DTA*.

– Let *Acc* be a *po2DTA* that immediately accepts without moving the head. Thus, $\mathcal{F}_{Acc}(w, i, \nu) = (t, i, \nu)$. Similarly, let *Rej* be a *po2DTA* that immediately rejects without moving the head. Hence, $\mathcal{F}_{Rej}(w, i, \nu) = (r, i, \nu)$.
– Let *Start* be a *po2DTA* that scans leftward for the start end marker and then accepts after moving one step to the right. Thus, $\forall i \in dom(w)$ we have $\mathcal{F}_{Start}(w, i, \nu) = (t, 1, \nu)$. Similarly, let *End* be the automaton that accepts at the end of the word. Thus, $\forall i \in dom(w)$, we have $\mathcal{F}_{End}(w, i, \nu) = (t, |w|, \nu)$.

- Let $Freeze(x)$ be a *po2DTA* which accepts after assigning the current time to the clock x. Hence, $\mathcal{F}_{Freeze(x)}(w,i,\nu) = (t,i,\nu')$, where $\nu' = \nu \otimes (x \rightarrow \tau_i)$.
- Let $\mathcal{M}_1?\mathcal{M}_2 : \mathcal{M}_3$ be an automaton that executes M_1 first. If M_1 accepts then the execution continues with the run of M_2 from the final configuration of M_1. If M_1 rejects then the execution continues with the run of M_3. We omit the details of this simple construction. Let $\mathcal{M} = \mathcal{M}_1?\mathcal{M}_2 : \mathcal{M}_3$. Then the size (number of states) of $\mathcal{M}$ is equal to the sum of the sizes of $\mathcal{M}_1$, $\mathcal{M}_2$ and $\mathcal{M}_3$. Also, $\mathcal{F}_{\mathcal{M}}(w,i,\nu) = \mathcal{F}_{\mathcal{M}_2}(w,j,\nu')$ if $\mathcal{F}_{\mathcal{M}_1}(w,i,\nu) = (t,j,\nu')$ and $\mathcal{F}_{\mathcal{M}}(w,i,\nu) = \mathcal{F}_{\mathcal{M}_3}(w,j,\nu')$ if $\mathcal{F}_{\mathcal{M}_1}(w,i,\nu) = (r,j,\nu')$.
- Let $\mathcal{M}_1;\mathcal{M}_2 = \mathcal{M}_1?\mathcal{M}_2 : Rej$.

We state the following lemma without proof.

Lemma 1 (Boolean Closure). *Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be po2DTA with disjoint clock sets.*

- *Let $\mathcal{M}_1 \vee \mathcal{M}_2 = \mathcal{M}_1?Acc : (Start;\mathcal{M}_2)$.*
  Then, $\mathcal{L}(\mathcal{M}_1 \vee \mathcal{M}_2) = \mathcal{L}(\mathcal{M}_1) \cup \mathcal{L}(\mathcal{M}_2)$.
- *Let $\mathcal{M}_1 \wedge \mathcal{M}_2 = \mathcal{M}_1?(Start;\mathcal{M}_2) : Rej$.*
  Then, $\mathcal{L}(\mathcal{M}_1 \wedge \mathcal{M}_2) = \mathcal{L}(\mathcal{M}_1) \cap \mathcal{L}(\mathcal{M}_2)$.
- *Let $\neg\mathcal{M}_1 = \mathcal{M}_1?Rej : Acc$. Then, $\mathcal{L}(\neg\mathcal{M}_1) = T\Sigma^* \setminus \mathcal{L}(\mathcal{M}_1)$.*

Note that in each case, the size of the constructed automaton exceeds the sum of sizes the component automata by only a constant factor. The construction can be carried out in time polynomial in the input. □

2.2 po2DTA: Decision Problems

Membership Since *po2DTA* are total and deterministic, there is a unique run ρ of the automaton on any word w. As stated before, the size of the run is upperbounded by $|w| \times (n-1)$ where n is the number of states. For every timed word w with rational timestamps, we may simulate this run on the automaton in time polynomial in the size of the word and number of states of the automaton, and decide whether or not the automaton accepts w. Thus, the membership problem for *po2DTA* is in P.

Non-emptiness. As discussed earlier, recollect that a *po2DTA* with n states has at most $(n-1)$ clocks. Given positive integers n, k, we define an equivalence $\approx^n_k$, using ideas similar to the region equivalence found in literature. Consider two sequences of timestamps $\overline{u} = u_1...u_n$ and $\overline{t} = t_1...t_n$. Define $\overline{u} \approx^n_k \overline{t}$ iff $\forall 0 \leq i,j < n$

- $t_i - t_j > k \Leftrightarrow u_i - u_j > k$
- $(u_i - u_j) \leq k \Rightarrow Int(u_i - u_j) = Int(t_i - t_j)$
- $Fra(u_i) < Fra(u_j) \Leftrightarrow Fra(t_i) < Fra(t_j)$
  $Fra(u_i) = Fra(u_j) \Leftrightarrow Fra(t_i) = Fra(t_j)$

Lemma 2. *Consider po2DTA $\mathcal{M}$ k_{max} as the maximum integer constant appearing on the guards. Consider two words $w = (\overline{\sigma}, \overline{u})$ and $w' = (\overline{\sigma}, \overline{t})$ of length p, with the same untimed word but possibly different timestamps. If $\overline{u} \approx^{p}_{k_{max}} \overline{t}$, then $w \in \mathcal{L}(\mathcal{M})$ iff $w' \in \mathcal{L}(\mathcal{M})$.*

Proof. (Outline) We can inductively observe that the run of $\mathcal{M}$ on both w and w' takes the same sequence of edges (transitions). Essentially, the guards compare the time distance between two timestamps with integer constants $\leq k_{max}$. Hence, if the configurations after i steps in runs on w and w' are (q, ν, l) and (q, ν', l) respectively, then for any guard g we have $\nu, u_l \models g$ iff $\nu', t_l \models g$ due to $\overline{u} \approx^{p}_{k_{max}} \overline{t}$. Hence, the same edge e is taken as $i+1$'th transition in both runs. □

Lemma 3. *Given positive integers n, k_{max}, for all $\overline{u}$ of length $n-1$, there exists $\overline{t}$ of length $n-1$ such that $\overline{u} \approx^{n-1}_{k_{max}} \overline{t}$ and $\forall i$*

- *$Int(t_i) \leq (k_{max}+1)(n-1)$*
- *$Fra(t_i)$ is a multiple of $1/n$*

Proof. (Outline) The equivalence definition states that once difference between successive numbers in list $\overline{u}$ is greater than k_{max} their magnitude does not affect the equivalence. Hence, we can construct an equivalent sequence $\overline{t}$ where difference in timestamps between successive elements is at most $k_{max}+1$. Also, only the relative ordering of fractional parts is important for equivalence. Since in each sequence there are at most $n-1$ different numbers, the relative ordering between their fractional parts can be faithfully recorded using fractions which are multiple of $1/n$. □

Theorem 1 (Small-model property). *Given a po2DTA $\mathcal{M}$ with n number of states and k_{max} being the maximum integer appearing on the guards, if $\mathcal{L}(\mathcal{M})$ is non-empty, then there exists a timed word $w = (\sigma_1, \tau_1)...(\sigma_m, \tau_m) \in \mathcal{L}(\mathcal{M})$ such that*

- *$m < n$*
- *$\tau_m \leq (n-1)(k_{max}+1)$ and*
- *$\forall i \in dom(w)$, the fractional part of τ_i is a multiple of $1/n$*

Proof. Consider any word $w \in \mathcal{L}(\mathcal{M})$ (since $\mathcal{L}(\mathcal{M})$ is non-empty). Let w_1 be the subword of w consisting of only those positions which correspond to the progress transitions of $\mathcal{M}$. Since resets only occur at the progress transitions, it is easy to see that $w \in \mathcal{L}(\mathcal{M}) \Leftrightarrow w_1 \in \mathcal{L}(\mathcal{M})$. Since the number of progress transitions on any path from start to accept state is $< n$, we know that length of w_1 is $< n$. Let $w_1 = (\overline{\sigma}, \overline{u})$ and $w_2 = (\overline{\sigma}, \overline{t})$ such that $u \approx^{n-1}_{k_{max}} \overline{t}$ and $\forall i$

- $Int(t_i) \leq (k_{max}+1)(n-1)$
- $Fra(t_i)$ is a multiple of $1/n$

From Lemma 2, we know that such a word w_2 exists, and from Lemma 3, we know that $w_1 \in \mathcal{L}(\mathcal{M}) \Leftrightarrow w_2 \in \mathcal{L}(\mathcal{M})$. Hence, we have the word w_2 with the desired properties. □

Corollary 1 (Non-emptiness and Language Inclusion)

- *Non-emptiness of po2DTA is decidable with NP-Complete complexity.*
- *Language inclusion of po2DTA is decidable with Co-NP-Complete complexity.*

Proof. From Theorem 1, we know that a polynomial-sized word is included in the language of any *po2DTA* with non-empty language. This word can be guessed non-deterministically and checked for membership using the PTIME membership checking. Hence, non-emptiness of *po2DTA* is in NP. The non-emptiness of the (untimed) subclass *po2dfa* of *po2DTA* is already shown to be NP-complete [SP09] by giving a log-space reduction of satisfiability of CNF boolean formulae to non-emptiness of *po2dfa*. This immediately implies the NP-hardness of *po2DTA* non-emptiness.

Note that $\mathcal{L}(\mathcal{M}_1) \subseteq \mathcal{L}(\mathcal{M}_2)$ iff $\mathcal{L}(\neg(\mathcal{M}_2) \wedge \mathcal{M}_1) = \emptyset$. Since boolean operations on *po2DTA* automata are achieved in PTIME with linear increase in size by Lemma 1, and non-emptiness checking is NP-Complete, we have that language inclusion of *po2DTA* is in co-NP-Complete. □

3 *DUMTL*

In real-time logics, Freeze quantifiers are typically used in order to bind the time of occurrence of some events to variables[AH91]. Let Σ be the finite alphabet and X the finite set of freeze variables. We use the same syntax of guards G_X as in *po2DTA* to express time constraints. A *Guarded Event* over Σ, X is a pair $\theta = (a, g)$ such that $a \in \Sigma$ and $g \in G_X$ and a Guarded-Event-Set ξ over Σ, X is a finite set of guarded events. (A guarded event is just a singleton Guarded-Event-Set). A Guarded-Event-Function is a function in $\Sigma \rightarrow G_X$. For each Guarded-Event-Set ξ we define equivalent Guarded-Event-Function χ_ξ such that $\chi_\xi(a) = \vee\{g_i \mid (a, g_i) \in \xi\}$. Also, for a Guarded-Event-Function χ we can define a Guarded-Event-Set $\xi_\chi = \{(a, \chi(a)) \mid a \in \Sigma\}$. Thus, Guarded-Event-Set and Guarded-Event-Function are equivalent representations. Let $\top$ denote Guarded-Event-Set such that $\chi_\top(a) = true$. Given two Guarded-Event-Function χ_1 and χ_2, we define boolean operation on them in pointwise manner. E.g. $\forall a \in \Sigma.\ (\chi_1 \wedge \chi_2)(a) = \chi_1(a) \wedge \chi_2(a)$. This also allows us to carry out boolean operations on Guarded-Event-Set. This notational facility will be useful in constructing automata.

The logic *DUMTL* over Σ, X specifies properties of finite timed words using guarded events in its formulae. Let ϕ, ϕ_1, ϕ_2 range over *DUMTL* formulas, θ be any guarded event and ξ be any Guarded-Event-Set. The syntax of *DUMTL* formulas is as follows:

$$\phi := true \mid \theta \mid \xi \mathsf{U}^x_\theta \phi_1 \mid \xi \mathsf{S}^x_\theta \phi_1 \mid \neg\phi_1 \mid \phi_1 \vee \phi_2$$

Semantics. Let $\theta = (a, g)$, then $w, i \models_\nu \theta$ iff $(\sigma_i = a) \wedge (\nu, \tau_i \models g)$. Similarly, $w, i \models_\nu \xi$ iff $\nu, \tau_i \models \chi_\xi(\sigma_i)$. The remaining cases are defined inductively:

$w, i \models_\nu \xi \mathsf{U}^x_\theta \phi$ iff $\exists j > i. w, j \models \theta \wedge \forall i < k < j. w, k \models (\xi \wedge \neg\theta) \wedge w, j \models_{\nu'} \phi$
where $\nu' = \nu \otimes x \to \tau_j$.
$w, i \models_\nu \xi \mathsf{S}^x_\theta \phi$ iff $\exists j < i. w, j \models \theta \wedge \forall j < k < i. w, k \models (\xi \wedge \neg\theta) \wedge w, j \models_{\nu'} \phi$
where $\nu' = \nu \otimes x \to \tau_j$.
$w, i \models_\nu \phi_1 \vee \phi_2$ iff $w, i \models_\nu \phi_1$ or $w, i \models_\nu \phi_2$
$w, i \models_\nu \neg\phi_1$ iff $w, i \not\models_\nu \phi_1$

Notice that the U and S modalities are strict and no next/previous modalities are needed. The language of formula ϕ is given by $\mathcal{L}_\phi = \{w \mid w, 0 \models_{\nu_{init}} \phi\}$ where ν_{init} assigns the value 0 to each freeze variable.

Example 2. Let $\theta_1 = (b, x < (T-1))$ and $\theta_2 = (c, y = T+1)$. Then the formula $\top \mathsf{U}^y_{\theta_1}(\top \mathsf{S}_{\theta_2} true)$ defines the language of all timed words where there is a b after time 1, and there is a c exactly one time unit before the first occurrence of b after time 1.

3.1 From *DUMTL* to *po2DTA*

Consider a *DUMTL* formula ϕ. Our aim is to construct a *po2DTA* $\mathcal{M}(\phi)$ recognizing the language $L(\phi)$. The logic is deterministic. Hence, given a timed word w, in evaluating ϕ any subformula ψ of ϕ can be uniquely determined to be relevant or irrelevant. Moreover, if relevant, its value is needed only at a determined position in the word. We denote this position by pos^ψ_w. We construct a *po2dfa* $\mathcal{R}(\psi)$, called the ranker of ψ such that, starting with any position, $\mathcal{R}(\psi)$ accepts at position pos^ψ_w if the formula is relevant and rejects if it is irrelevant. The ranker actually depends not on the subformula ψ but the context λ such that $\phi = \lambda(\psi)$. Its inductive construction is given below. Let $\mathcal{A}_{\mathsf{U}}(\xi, \theta, x)$ and $\mathcal{A}_{\mathsf{S}}(\xi, \theta, x)$ be *po2DTA* as shown in Figure 2 which scan forwards (or backwards respectively) accepting at the first (or last) θ provided all intermediate positions are events in ξ.

- $\mathcal{R}(\phi) = Start$
- If $\psi = \psi_1 \vee \psi_2$ then $\mathcal{R}(\psi_1) = \mathcal{R}(\psi_2) = \mathcal{R}(\psi)$.
  Also, if $\psi = \neg\psi_1$ then $\mathcal{R}(\psi_1) = \mathcal{R}(\psi)$.
- $\psi = \xi \mathsf{U}^x_\theta \psi_1$ then $\mathcal{R}(\psi_1) = \mathcal{R}(\psi); \mathcal{A}_{\mathsf{U}}(\xi, \theta, x)$
- $\psi = \xi \mathsf{S}^x_\theta \psi_1$ then $\mathcal{R}(\psi_1) = \mathcal{R}(\psi); \mathcal{A}_{\mathsf{S}}(\xi, \theta, x)$

Given rankers $R(\psi)$ for each subformula ψ, we can construct *po2DTA* automaton $\mathcal{M}(\psi)$ in a bottom-up fashion so that the automaton accepts if ψ is relevant and evaluates to true at its relevant position. A fresh clock is introduced each time we assign a freeze quantifier.

- If $\psi = \theta$ then $\mathcal{M}(\psi) = \mathcal{R}(\psi); \mathcal{A}(\theta)$ where $\mathcal{A}(\theta)$ accepts immediately if θ evaluates to true at current position and rejects immediately otherwise.
- If $\psi = \xi \mathsf{U}^x_\theta \psi'$ then $\mathcal{M}(\psi) = \mathcal{R}(\psi); \mathcal{A}_{\mathsf{U}}(\xi, \theta, x); \mathcal{M}(\psi')$
- If $\psi = \xi \mathsf{S}^x_\theta \psi'$ then $\mathcal{M}(\psi) = \mathcal{R}(\psi); \mathcal{A}_{\mathsf{S}}(\xi, \theta, x); \mathcal{M}(\psi')$
- If $\psi = \psi_1 \vee \psi_2$ then $\mathcal{M}(\psi) = \mathcal{M}(\psi_1) \vee \mathcal{M}(\psi_2)$
- If $\psi = \neg\phi$ then $\mathcal{M}(\psi) = \neg(\mathcal{M}(\phi))$

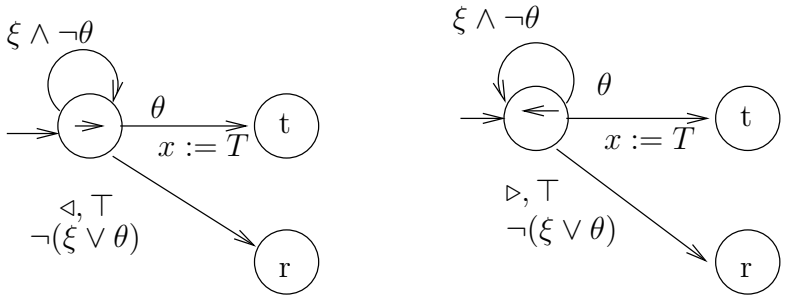


Fig. 2. Automata for $\mathcal{A}_{\mathsf{U}}(\xi, \theta, x)$ and $\mathcal{A}_{\mathsf{S}}(\xi, \theta, x)$

We state the following lemma without proof.

Lemma 4. *Given a DUMTL formula ϕ, the above construction gives an equivalent po2DTA $\mathcal{M}(\phi)$ such that $\forall w \in T\Sigma^* \;.\; w \in \mathcal{L}(\phi)$ iff $w \in \mathcal{L}(\mathcal{M}(\phi))$.*

3.2 From *po2DTA* to *DUMTL*

We now give a translation from an automaton $\mathcal{M}$ to a language equivalent *DUMTL* formula $\phi_{\mathcal{M}}$. Since the automaton works on words $\hat{w} = (\rhd, 0)w(\lhd, \tau_{|w|})$, we shall extend the alphabet of the *DUMTL* formulas to include end-markers: $\Sigma' = \Sigma \cup \{\rhd, \lhd\}$. For each clock in the automaton, we have a freeze variable in

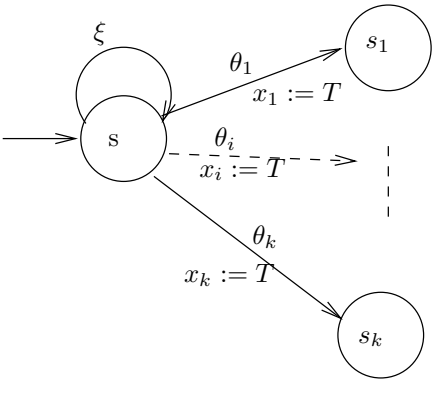


Fig. 3. A typical state of *po2DTA*with incoming and outgoing transitions

the formula. Consider a state s in $\mathcal{M}$ as shown in Figure 3. We will construct a formula ψ_s satisfying the following lemma which we give here without proof.

Lemma 5. *For any state s of $\mathcal{M}$ and for any word w if $\mathcal{M}$ in its partial run executes a progress transition e with head at position p and enters the configuration (s, p', ν) then $w, p \models_\nu \psi_s$ iff there is an accepting run of $\mathcal{M}$ on w from the configuration (s, p', ν)*

We inductively construct ψ_s for each state as follows:

- $\psi_t = \top$ and $\psi_r = \bot$
- Given ψ_{s_i} for all $1 \leq i \leq k$ (as in the figure), then
  - If $s \in Q_r$, $\psi_s := \vee_{1 \leq i \leq k}(\xi \mathsf{U}_{\theta_i}^{x_i}(\psi_{s_i}))$
  - If $s \in Q_l$, $\psi_s := \vee_{1 \leq i \leq k}(\xi \mathsf{S}_{\theta_i}^{x_i}(\psi_{s_i}))$

For the formula ψ_s corresponding to the start state s, we assume that there is a dummy incoming edge on the left end-marker, for which s is the target state. Hence, the *DUMTL* formula ψ_s must be evaluated from the position 0.

Theorem 2. *Given a po2DTA $\mathcal{M}$, there is an equivalent DUMTL formula $\psi_{\mathcal{M}} = \psi_s$ (where s is the start state) such that $\forall w \in T\Sigma^*$, $\mathcal{M}$ accepts w iff $\hat{w}, 0 \models_{\nu_{init}} \psi_{\mathcal{M}}$, where ν_{init} assigns 0 to each freeze variable.*

4 Unary Metric Temporal Logics and Expressiveness

For untimed words, it was shown by [EVW02] that Unary Temporal Logic (UTL) is expressively equivalent to po2DFA. Here, we add timing constraints to the temporal modalities of UTL to get logic *UMTL*.

Consider finite timed words over a finite alphabet, Σ. Let ϕ, ϕ_1, ϕ_2 range over *UMTL* formulae and $a \in \Sigma$. An interval I is a convex subset of non-negative reals with integral end-points. It may be open, half-open or closed. Such an interval may be represented as $I =< i, j >$ in general. Here j can even be ∞ specifying infinite right open interval. If $i = j$, then the interval is said to be singular. It is non-singular if $i < j$. The syntax of *UMTL* is as follows:
$\phi := a \mid \overset{\rightarrow}{\Diamond}_I \phi \mid \overset{\leftarrow}{\Diamond}_I \phi \mid \phi_1 \vee \phi_2 \mid \neg\phi$
Let $w = (\sigma_1, \tau_1), ...(\sigma_m, \tau_m)$ be a finite timed word over Σ. *UMTL* formulas are interpreted over a position in the timed word. We give the semantics of *UMTL* formulas as follows.

$w, i \models a$ iff $\sigma_i = a$
$w, i \models \overset{\rightarrow}{\Diamond}_I \phi$ iff $\exists j > i . \tau_j \in \tau_i + I \wedge w, j \models \phi$
$w, i \models \overset{\leftarrow}{\Diamond}_I \phi$ iff $\exists j < i . \tau_j \in \tau_i - I \wedge w, j \models \phi$
$w, i \models \phi_1 \vee \phi_2$ iff $w, i \models \phi_1 \vee w, i \models \phi_2$
$w, i \models \neg\phi$ iff $w, i \not\models \phi$

4.1 Punctuality and Expressiveness

The logic *UMTL* allows singular or punctual intervals in its formulae. Let *UMITL* be the subclass of *UMTL* where all intervals are syntactically required to be non-punctual. In context of MITL, Alur and Henziger [AH92] have shown with examples that punctuality is a property that cannot be expressed by two-way deterministic TA even without any reversal bounds.

Theorem 3. *There is no po2DTA which recognizes the language defined by the UMTL formula $\phi := \overset{\rightarrow}{\Diamond}_{(0,1)}(a \wedge \overset{\rightarrow}{\Diamond}_{[3,3]} c)$.*

Proof. Let us assume $\mathcal{M}_\phi$ is a *po2DTA* which recognizes $\mathcal{L}(\phi)$, and $\mathcal{M}_\phi$ has n states.
Consider a word w such that $untime(w) = a^{2n+1}c^{2n+1}$ such that all the a's are in the interval $(0,1)$ and $\overset{\triangledown}{}_{[3,3]}c$ holds only for the $(n+1)^{th}$ a. In order to recognize the timestamp of this a, there must be a progress transition on this a (because resets are allowed only on progress transitions). However, if the $(n+1)^{th}$ a within the unit interval $(0,1)$ is on a progress transition, it must imply that the other n a's before (or after) are also on progress transitions since the automaton is deterministic. But the automaton $\mathcal{M}_\phi$ may have at most $(n-1)$ progress transitions on a given run. Hence, the assumption that $\mathcal{M}_\phi$ recognizes the language is false. □

Theorem 4. *The automaton in the figure below does not have an equivalent UMITL formula.*

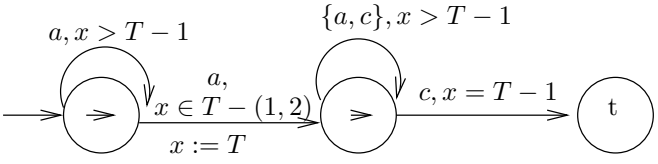


Fig. 4. Expressiveness of *UMITL*

Proof (Outline). The following property holds for *UMITL* formulas. We omit its detailed proof. Given a *UMITL* formula ϕ there exists $\delta > 0$ such that for any word $w = (\overline{\sigma}, \overline{t}) \in \mathcal{L}(\phi)$, there exists a sequence of open intervals $\overline{I}$ such that each I_i is of width δ and $t_i \in I_i$ and for any sequence $\overline{u}$ formed with $u_i \in I_i$, we have $w' = (\overline{\sigma}, \overline{u}) \in \mathcal{L}(\phi)$. Informally, it states that time stamps of models of a *UMITL* formula can be perturbed in a small neighbourhood preserving truth. Now, the automaton $(\mathcal{M})$ above accepts all words such that for the first a in the interval $(1,2)$, there is a c exactly one time unit after it. A word in $\mathcal{L}(\mathcal{M})$ with only one c in it, will not satisfy the perturbation property of *UMITL* formulas stated above. Hence, there is no *UMITL* formula equivalent to $\mathcal{M}$. □

5 Discussion

Unambiguous languages [Sch75] are a robust subclass of regular languages with diverse characterizations. The *po2dfa* give an automaton based characterization of this class[STV01] and the unary temporal logic, UTL, which is a temporal logic using only the modalities $\overset{\triangledown}{}$ and $\overset{\triangledown}{}$ provides a logical characterization of this class [EVW02]. In our past work [LPS08, LPS10] we have shown that deterministic temporal logics admitting unique parsability can be defined for unambiguous languages. The deterministic logics have intimate connections with the automata for unambiguous languages and this results into efficient satisfiability checking of determinstic logic formulae. In this paper, we carry over the same approach to much more challenging setting of timed languages.

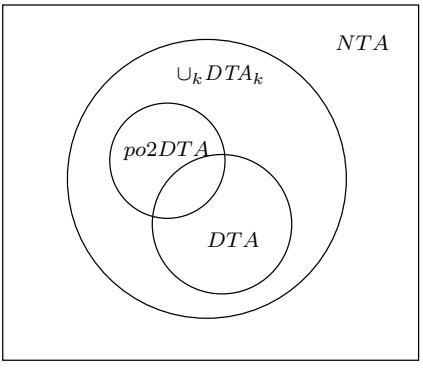


Fig. 5. Showing Expressiveness of po2DTA

In this paper, we have initiated the study of timed unambiguous languages (TUL). We have chosen to specify this class by defining partially ordered two way deterministic timed automata, *po2DTA*, which recognize TUL. We have also come up with a temporal logic called deterministic, unary metric temporal logic, *DUMTL*, which is expressvily complete for *po2DTA*. Using the reduction from logic to automata we have shown that the satisfiability of *DUMTL* is NP-complete.

Alur and Henzinger [AH92] defined *2DTA*. The subclass of *2DTA* with at most k reversals is called DTA_k. The subclass $\cup_k DTA_k$ of reversal bounded 2DTA is boolean closed and the emptiness of DTA_k with fixed reversal bound k is PSPACE-complete. Alur and Henzinger also showed that logic $MITL[\mathsf{U}_I, \mathsf{S}_I]$ can be reduced to $\cup_k DTA_k$ giving EXPSPACE satisfiability for the logic.

Our *po2DTA* are a subclass of $\cup_k DTA_k$. They have many pleasant properties: they are boolean closed (with only polynomial blowup), their non-emptiness is NP-Complete and the language inclusion between automata is Co-NP-Complete. Moreover, an automaton with n states and arbitrarily many clocks can be reduced to an automaton with at most n-1 clocks. Expressively, *po2DTA* are a strict subset of $\cup_k DTA_k$ and unrelated to the deterministic timed automata (DTA). The exact expressiveness is depicted diagrammatically in Figure 5.

For untimed case, unambiguous languages correspond to unary temporal logic [EVW02]. The generalization of this to timed case turns out to be tricky. The unary temporal logic is obtained by replacing the U and the S by unary operators $\diamondsuit$ and $\overleftarrow{\diamondsuit}$. Unfortunately, this method does not yield timed unambiguous languages. In the paper, we have considered the unary version of logic MTL and called it *UMTL*. We also considered the unary fragment of MITL and called it *UMITL*. In Section 4, we were able to show with examples that *UMTL* is not contained within *po2DTA* and that *po2DTA* are not contained within *UMITL*. Hence, neither of these logics characterize the class of timed unambiguous languages. Our deterministic and unary logic *DUMTL* seems to adequately capture

the essence of unambiguity in timed regular languages. In our opinion, the efficient satisfiability checking of *DUMTL* is a direct consequence of this connection. The exact relationship between the expressive powers of logics *DUMTL*, *UMTL* and *UMITL* is a topic of our ongoing investigation.

References

[AD91] Alur, R., Dill, D.: A Theory of Timed Automata. Theo. Comp. Sc. 126(2), 183–235 (1994)

[AH91] Alur, R., Henzinger, T.A.: Logics and Models of Real Time: A Survey. In: Huizing, C., de Bakker, J.W., Rozenberg, G., de Roever, W.-P. (eds.) REX 1991. LNCS, vol. 600, pp. 74–106. Springer, Heidelberg (1992)

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

[AH92] Alur, R., Henzinger, T.: Back to the Future: Towards a Theory of Timed Regular Languages. In: FOCS 1992, pp. 177–186 (1992)

[Koz97] Kozen, D.: Automata and Computability. Springer, Heidelberg (1997)

[DGK08] Diekert, V., Gastin, P., Kufleitner, M.: A survey on small fragments of first-order logic over finite words. Int. J. Found. Comp. Sci. 19(3), 513–548 (2008)

[EVW02] Etessami, K., Vardi, M.Y., Wilke, T.: First-order logic with two variables and unary temporal logic. Inf. Comput. 179, 279–295 (2002)

[LPS08] Lodaya, K., Pandya, P.K., Shah, S.S.: Marking the chops: an unambiguous temporal logic. In: Ausiello, G., Karhumäki, J., Mauri, G., Ong, L. (eds.) Proc. 5th IFIP TCS, Milano. IFIP Series, vol. 273, pp. 461–476 (2008)

[LPS10] Lodaya, K., Pandya, P.K., Shah, S.S.: Around dot-depth two. In: Yu, S. (ed.) Proc. 14th DLT, London, Canada. LNCS (2010)

[OW05] Ouaknine, J., Worrell, J.: On the Decidability of Metric Temporal Logic over finite words. LMCS 3(1) (2007)

[Sch75] Schützenberger, M.-P.: Sur le produit de concaténation non ambigu. Semigroup Forum 13, 47–75 (1976)

[STV01] Schwentick, T., Thérien, D., Vollmer, H.: Partially-ordered two-way automata: a new characterization of DA. In: Kuich, W., Rozenberg, G., Salomaa, A. (eds.) DLT 2001. LNCS, vol. 2295, pp. 239–250. Springer, Heidelberg (2002)

[SP09] Shah, S.S., Pandya, P.K.: An automaton normal form for UITL, Technical Report STCS-TR-SP-2009/1. Computer Science Group, TIFR (2009)

[TW98] Thérien, D., Wilke, T.: Over words, two variables are as powerful as one quantifier alternation: $FO^2 = \Sigma_2 \cap \Pi_2$. In: Proc. STOC, Dallas, pp. 41–47 (1998)

A Process Algebraic Framework for Modeling Resource Demand and Supply*

Anna Philippou[1], Insup Lee[2], Oleg Sokolsky[2], and Jin-Young Choi[3]

[1] Department of Computer Science, University of Cyprus, Nicosia, Cyprus
[2] Department of Computer and Info. Science, University of Pennsylvania, Philadelphia, PA, U.S.A.
[3] Department of Computer Science, Korea University, Seoul, Korea

Abstract. As real-time embedded systems become more complex, resource partitioning is increasingly used to guarantee real-time performance. Recently, several compositional frameworks of resource partitioning have been proposed using real-time scheduling theory with various notions of real-time tasks running under restricted resource supply environments. However, these real-time scheduling-based approaches are limited in their expressiveness in that, although capable of describing resource-demand tasks, they are unable to model resource supply. This paper describes a process algebraic framework for reasoning about resource demand *and* supply inspired by the timed process algebra ACSR. In ACSR, real-time tasks are specified by enunciating their consumption needs for resources. To also accommodate resource-supply processes we define PADS where, given a resource CPU, the complimented resource $\overline{CPU}$ denotes for availability of CPU for the corresponding demand process. Using PADS, we define a supply-demand relation where a pair (S, T) belongs to the relation if the demand process T can be scheduled under supply S. We develop a theory of compositional schedulability analysis as well as a technique for synthesizing an optimal supply process for a set of tasks. We illustrate our technique via a number of examples.

1 Introduction

The increasing complexity of real-time embedded systems demands compositional design and analysis methods for the assurance of timing requirements. Component-based design has been widely accepted as a compositional approach to facilitate the design of complex systems. It provides means for decomposing a complex system into simpler components and for composing the components using interfaces that abstract component complexities. Such approaches are increasingly used in practice for real-time systems. For example, ARINC-653 standard by the Engineering Standards for Avionics and Cabin Systems committee specifies partition-based design of avionics applications. Also, hypervisors for real-time virtual machines provide temporal partitions to guarantee real-time performance [15,11].

To take advantage of the component-based design of real-time systems, schedulability analysis should support compositional analysis using component interfaces.

* This research was supported in part by NSF grants CNS-0834524 and CNS-0720703.

K. Chatterjee and T.A. Henzinger (Eds.): FORMATS 2010, LNCS 6246, pp. 183–197, 2010.

These interfaces should abstract the timing requirements of a component with a minimum resource supply that is needed to meet the resource demand of the component. Component-based real-time systems often involve hierarchical scheduling frameworks that support resource sharing among components as well as associated scheduling algorithms [5,20]. To facilitate the analysis of such systems, resource component interfaces and their compositional analysis have been proposed [16,21,22,8,23,12]

This paper presents a formal treatment of the problem of compositional hierarchical scheduling by introducing a process algebraic framework, PADS, for modeling resource demand and supply inspired by the timed process algebra ACSR [13,14]. The notions of resource demand and supply are fundamental in defining the meaning of compositional real-time scheduling analysis. Our proposed algebraic framework formally defines both of these notions. As in ACSR, a task in our formalism is specified by describing its consumption needs for resources. To also accommodate resource-supply processes, we extend the notion of a *resource* and given a resource cpu we use $\overline{cpu}$ to denote the availability of the resource for consumption by a requesting task. Our formalism then addresses the following issues:

1. *Schedulability*: We define a *supply simulation relation* $\models$ that captures when a task T is schedulable by a supply S, $S \models T$.
2. *Compositionality*: We explore conditions under which we may safely compose schedulable systems. Specifically, we are interested to define functions on supplies, $\circ$, and appropriate conditions, f, such that if T_1 is schedulable by S_1 and T_2 by S_2 then the parallel composition of T_1 and T_2 is schedulable by $S_1 \circ S_2$, assuming that condition f holds:
$$\frac{S_1 \models T_1, S_2 \models T_2}{S_1 \circ S_2 \models T_1 \| T_2}, \quad f(S_1, S_2)$$
3. *Supply Synthesis*: We propose a method by which we can generate a supply process to schedule a set of tasks, assuming that such a scheduler exists. Our method is based on the notion of a *demand* of a task which is a supply that can schedule the task and, at the same time, it is optimal in the sense that (1) it does not reserve more resources than those required and (2) it captures all possibilities in which a task can be scheduled. We then prove that two or more tasks are schedulable if and only if they can be scheduled by the composition of their demands.

Related work. As mentioned above, this work brings together two long-standing lines of research. On the one hand, there has been much work on compositional hierarchical scheduling based on real-time scheduling theory [16,21,22,8,6,7]. Typically, such approaches to schedulability analysis rely on overapproximations of task demand using, for example, demand bound functions and underapproximations of resource supply using, supply bound functions. Efficient algorithms are developed to ensure that demand never exceeds supply. On the other hand, several formal approaches to scheduling based on process algebras [3,14,13,19,18], task automata [10,9], preemptive Petri nets [4], etc., have been developed. To the best of our knowledge, none of these approaches consider the problem of modeling resource supply explicitly. Instead, sharing of a continuously available processing resource between a set of tasks has been considered.

Our approach to supply synthesis is conceptually similar to the work of Altisen *et al.* on applying controller synthesis to scheduling problems [1,2]. The difference is that we are not aiming to generate schedulers, but rather an *interface* for a task set, an abstraction that can be used in a component-based approach to real-time system design.

The rest of the paper is structured as follows. Section 2 presents our process algebra and its semantics. Section 3 contains our results on compositional schedulability analysis and interface construction, followed by examples illustrating the application of the theory in Section 4. Section 5 concludes the paper.

2 The Language

In our calculus, PADS (Process Algebra for Demand and Supply), we consider a system to be a set of processes operating on a set of serially reusable resources denoted by R. These processes are (1) the *tasks* of the system, which require the use of resources in order to complete their jobs, and (2) the *supplies*, that specify when each resource is available to the tasks. Based on this, each resource $r \in \mathsf{R}$ can be requested by a task, r, granted by a supply, $\overline{r}$, or consumed, $\overleftrightarrow{r}$, when a supply and a request for the resource are simultaneously available. An action in the formalism is a set containing resource requests, grants and consumptions, where each resource may be represented at most once. For example, the action $\{r_1, r_2\}$ represents a request for the resources r_1 and r_2 whereas the action $\{\overline{r_1}, \overleftrightarrow{r_2}, r_3\}$ involves the granting of resource r_1, consumption of resource r_2 and request for resource r_3. We take a discrete time approach: we assume that all actions require one unit of time to complete measured on a global clock with action $\emptyset$ representing idling for one time unit since no resource is being employed.

We write *Act*, ranged over by α and β, for the set of all actions and distinguish Act_R, the set of actions involving only resource requests, ranged over by ρ, and Act_G, the set of actions involving only resource grants, ranged over by γ. Given $\alpha \in Act$ we use the notation $\overline{\alpha}$ to reverse between resource grants and requests in action α, so, $\overline{\{r_1, \overline{r_2}, \overleftrightarrow{r_3}\}} = \{\overline{r_1}, r_2, \overleftrightarrow{r_3}\}$. Finally, we write $res(\alpha)$ for the set of resources occurring in α: $res(\alpha) = \{r \in \mathsf{R} | r \in \alpha \text{ or } \overline{r} \in \alpha \text{ or } \overleftrightarrow{r} \in \alpha\}$.

2.1 Syntax

The following grammars define the set of tasks, T, the set of supplies S and the set of timed systems P, where I and J are sets of indices, and I is assumed to be nonempty. Furthermore, C ranges over a set of *task constants*, each with an associated definition of the form $C \stackrel{\text{def}}{=} T$, where T may contain occurrences of C as well as other task constants and, D ranges over a similar set of *supply constants*.

$$T ::= \text{FIN} \mid \Sigma_{i \in I} \rho_i : T_i \mid C$$
$$S ::= \text{FIN} \mid \Sigma_{i \in I} \gamma_i : S_i \mid D$$
$$P ::= \text{FIN} \mid T \mid S \mid P \| P \mid \Sigma_{j \in J} \alpha_j : P_j$$

We consider FIN to be the terminated process. Then a task process can be a terminated process, a task constant, or a nondeterministic choice $\Sigma_{i \in I} \rho_i : T_i$. The latter offers the

choice of executing each of the actions ρ_i and then proceeding as T_i, where $I \neq \emptyset$. Similarly, a supply process can be a terminated process, a supply constant, or a nondeterministic choice. Finally, a process can be an arbitrary composition of tasks and supplies or a nondeterministic choice between processes $\Sigma_{j\in J}\alpha_j : P_j$.

2.2 Semantics

The semantics of PADS are given in two steps. First, we develop a transition system in which nondeterminism is resolved in all possible ways, $\twoheadrightarrow$. Then, we refine $\twoheadrightarrow$ into $\longrightarrow$ by implementing a type of "angelic" behavior in the way in which tasks resolve their nondeterminism by choosing the best possible outcome given the available supply. The rules for the first-level transition relation can be found in Table 1, whereas the second-level transition relation is subsequently defined on the basis of a preemption relation.

We proceed to consider relation $\twoheadrightarrow$ defined in Table 1. FIN being a well-terminated (and not a deadlocked) process, it allows time to pass (axiom (IDLE)). Nondeterministic choice in tasks and supplies can be resolved by executing an action and then proceeding as its continuation ((SumT) and (SumS)). A constant behaves as the process in its defining equation (Const). Finally, rule (Par) specifies the way in which a parallel system evolves. To begin with, note that the components of a parallel composition evolve synchronously in that the composition advances only if both of the constituent processes are willing to take a step. Furthermore, the rule enunciates the outcome of the synchronization between two parallel processes, the most important aspect being that a request within the one component is satisfied by an available grant in the other. The condition of rule (PAR) imposes a restriction on when two actions may take place simultaneously within a system. Specifically, we say that actions α_1 and α_2 are *compatible* with each other if, whenever r occurs in both actions then one occurrence must be a request and the other a supply of the resource. So, for example, it is not possible to simultaneously offer a resource in one component and consume or offer it in another, nor to request it by two different tasks. We capture this requirement as follows:

$$\mathsf{compatible}(\alpha_1, \alpha_2) = \bigwedge_{r\in res(\alpha_1)\cap res(\alpha_2)} (r \in \alpha_1 \wedge \overline{r} \in \alpha_2) \vee (r \in \alpha_2 \wedge \overline{r} \in \alpha_1)$$

We may now combine compatible actions by transforming a simultaneous request and supply of the same resource into a consumption:

$$\begin{aligned}\alpha_1 \oplus \alpha_2 = {} & \{r \in \alpha_1, \alpha_2 | \overline{r} \notin \alpha_1 \cup \alpha_2\} \cup \{\overline{r} \in \alpha_1, \alpha_2 | r \notin \alpha_1 \cup \alpha_2\} \\ & \cup \{\overleftrightarrow{r} \mid r \in \alpha_i, \overline{r} \in \alpha_{(i+1)mod2}, i \in \{1,2\}\} \cup \{\overleftrightarrow{r} \mid \overleftrightarrow{r} \in \alpha_1 \cup \alpha_2\}\end{aligned}$$

Note that, the associativity of the parallel composition operator with respect to $\twoheadrightarrow$ follows by the associativity of $\oplus$ which, in turn, is easy to prove by its definition.

Example 1. Consider the supply $S \stackrel{\text{def}}{=} \{\overline{r_1}, \overline{r_2}\} : S$ and the following task processes:

$$\begin{aligned} T_1 &\stackrel{\text{def}}{=} \{r_1, r_2\} : \text{FIN} + \{r_1\} : T_1 & T_2 &= \{r_2\} : \text{FIN} + \{r_1, r_3\} : T_2 \\ T_3 &\stackrel{\text{def}}{=} \{r_2\} : \text{FIN} & T_4 &= \{r_1\} : \text{FIN} \end{aligned}$$

Table 1. Transition rules for tasks, supplies and systems

(Idle)	$\mathrm{FIN} \overset{\emptyset}{\twoheadrightarrow} \mathrm{FIN}$	
(SumT)	$\Sigma_{i\in I}\rho_i : T_i \overset{\rho_i}{\twoheadrightarrow} T_i$	$I \neq \emptyset$
(SumS)	$\Sigma_{i\in I}\gamma_i : S_i \overset{\gamma_i}{\twoheadrightarrow} S_i$	$I \neq \emptyset$
(Const)	$\dfrac{P \overset{\alpha}{\twoheadrightarrow} P' \quad C \overset{\mathrm{def}}{=} P}{C \overset{\alpha}{\twoheadrightarrow} P'}$	
(Par)	$\dfrac{P_1 \overset{\alpha_1}{\twoheadrightarrow} P_1' \quad P_2 \overset{\alpha_2}{\twoheadrightarrow} P_2'}{P_1 \| P_2 \overset{\alpha_1 \oplus \alpha_2}{\twoheadrightarrow} P_1' \| P_2'}$	$\mathsf{compatible}(\alpha_1, \alpha_2)$
(SumP)	$\Sigma_{j\in J}\alpha_j : P_j \overset{\alpha_j}{\twoheadrightarrow} P_j$	$J \neq \emptyset$

We have:

$$T_1 \| S \overset{\{\overleftrightarrow{r_1}, \overleftrightarrow{r_2}\}}{\twoheadrightarrow} \mathrm{FIN} \| S \qquad (1) \qquad\qquad T_1 \| S \overset{\{\overleftrightarrow{r_1}, \overline{r_2}\}}{\twoheadrightarrow} T_1 \| S \qquad (2)$$

$$T_2 \| S \overset{\{\overline{r_1}, \overleftrightarrow{r_2}\}}{\twoheadrightarrow} \mathrm{FIN} \| S \qquad (3) \qquad\qquad T_2 \| S \overset{\{\overleftrightarrow{r_1}, \overline{r_2}, r_3\}}{\twoheadrightarrow} T_2 \| S \qquad (4)$$

$$(T_1 \| S) \| T_3 \overset{\{\overleftrightarrow{r_1}, \overleftrightarrow{r_2}\}}{\twoheadrightarrow} (T_1 \| S) \| \mathrm{FIN} \qquad (5)$$

Note that $(T_1 \| S) \| T_3$ has no transition other than (5) above, while $(T_1 \| S) \| T_4$ has no transitions altogether since both T_1 and T_4 require r_1 during the first time unit. □

Moving on to the second level of the semantics, we employ a preemption relation on actions to prune away all transitions that do not represent correctly the behavior of a system, as we would expect it. In particular, we make the following two assumptions:

1. Given a supply, a task should respond angelically and, given a nondeterministic set of transitions by which it can evolve, it should choose only between the ones that are satisfied by the available suply, assuming that such options exist. For example, $T_2 \| S$ above should retain only transition (3) out of the available (3) and (4).
2. In addition, we assume that a task behaves greedily and, at each step, it employs as many of the supplied resources as possible. For example, the composition $T_1 \| S$ in Example 1 should only retain transition (1) out of transitions (1) and (2).

Given the above, we define the *preemption relation* $\prec$ so that $\alpha \prec \beta$ if one of the following hold:

1. $\{r | \overline{r} \in \alpha, \overleftrightarrow{r} \in \alpha\} = \{r | \overline{r} \in \beta, \overleftrightarrow{r} \in \beta\}$, $\alpha \cap \mathsf{R} \neq \emptyset$ and $\beta \cap \mathsf{R} = \emptyset$,
2. $\alpha \cap \mathsf{R} = \beta \cap \mathsf{R} = \emptyset$, $\{r | \overline{r} \in \alpha, \overleftrightarrow{r} \in \alpha\} = \{r | \overline{r} \in \beta, \overleftrightarrow{r} \in \beta\}$ and $\{r | \overleftrightarrow{r} \in \alpha\} \subseteq \{r | \overleftrightarrow{r} \in \beta\}$.

Intuitively, an action precludes another if it makes better usage of the same offered resources. In particular, an action β preempts an action α, if, either α and β concern

the same offered resources (granted or consumed) but α, unlike β, also contains some unsatisfied resource requests (condition (1)), or α and β contain only the same granted and consumed resources but β consumes more resources than α (condition (2)).

We may now define the relation $\stackrel{\alpha}{\longrightarrow}$ by the following rule:

$$\frac{P \stackrel{\alpha}{\twoheadrightarrow} Q}{P \stackrel{\alpha}{\longrightarrow} Q}, \quad \text{there is no } P \stackrel{\beta}{\twoheadrightarrow}, \alpha \prec \beta$$

We conclude this section by introducing some notations. We write $P \longrightarrow$ if there exists action α such that $P \stackrel{\alpha}{\longrightarrow}$. If $P \stackrel{\alpha}{\not\longrightarrow}$ for all actions α, we write $P = \delta$, where δ is the deadlocked process. We write $P \Longrightarrow P'$ if there exist actions $\alpha_1 \ldots \alpha_n$ and processes $P_1, \ldots, P_{n-1}$ such that $P \stackrel{\alpha_1}{\longrightarrow} P_1 \stackrel{\alpha_2}{\longrightarrow} \ldots P_{n-1} \stackrel{\alpha_n}{\longrightarrow} P'$. The set of traces of P, $\mathsf{traces}(P)$, is defined to be the set of all infinite sequences of actions $\alpha_1\alpha_2 \ldots$ such that $P \stackrel{\alpha_1}{\longrightarrow} P_1 \stackrel{\alpha_2}{\longrightarrow} \ldots$. Finally, for $w = \alpha_1\alpha_2 \ldots$, we write $\overline{w}$ for $\overline{\alpha_1}\,\overline{\alpha_2} \ldots$

3 Schedulability

In this section we present a theory of schedulability for our calculus. We begin by defining when a set of tasks is considered to be schedulable by a supply. Then we present an alternative characterization based on a type of simulation relations and we prove the two definitions to be equivalent. In what follows we write T^* for the set containing all processes of the form $T_1 \| \ldots \| T_n$, $n \geq 1$, and S^* for the set containing all processes of the form $S_1 \| \ldots \| S_n$, $n \geq 1$. For simplicity, we refer to elements of T^* and S^* simply as tasks and supplies, respectively.

Definition 1. *A task $T \in \mathsf{T}^*$ is* schedulable under *supply $S \in \mathsf{S}^*$ if whenever $T \| S \Longrightarrow P$ then $P \neq \delta$ and for all $P \stackrel{\alpha}{\longrightarrow}$ we have $\alpha \cap \mathsf{R} = \emptyset$.*

According to this definition, a task T is schedulable under supply S if at no point during their interaction does the system deadlock and, moreover, no request for a resource remains unsatisfied.

Example 2. Here follow some examples relating to the above definition.

- $T \stackrel{\text{def}}{=} \{r\}$:FIN is not schedulable under $S \stackrel{\text{def}}{=} \{\overline{r'}\}$:FIN. We have $T \| S \stackrel{\{r,\overline{r'}\}}{\longrightarrow}$ and the definition is violated.
- $T \stackrel{\text{def}}{=} \{r\}$:FIN is schedulable under $S \stackrel{\text{def}}{=} \{\overline{r}, \overline{r'}\}$:FIN. We have $T \| S \stackrel{\{\leftrightarrow_{r}, \overline{r'}\}}{\longrightarrow}$ FIN$\|$FIN which satisfies the definition.
- $T \stackrel{\text{def}}{=} \{r\}$:FIN $+ \emptyset$:$\{r\}$:FIN is schedulable under $S \stackrel{\text{def}}{=} \{\overline{r}\}$:FIN. The composition of the two processes has only one possible transition $T \| S \stackrel{\{\leftrightarrow_{r}\}}{\longrightarrow}$ FIN$\|$FIN. Note that the transition $T \| S \stackrel{\{\overline{r}\}}{\twoheadrightarrow} \{r\}$:FIN$\|$FIN at the lower-level of the semantics is pruned by the preemption relation. Thus, the definition is satisfied. The same holds for $T \stackrel{\text{def}}{=} \{r\}$:FIN$+\{r'\}$:FIN and $S \stackrel{\text{def}}{=} \{\overline{r}\}$:FIN since $\{r', \overline{r}\} \prec \{\leftrightarrow_{r}\}$. This illustrates that as long as there is some possible way of scheduling a task's requirements by an available supply, the task is considered to be schedulable by the supply. □

Before moving on to our alternative schedulability definition we introduce the following useful notation: For $T \in \mathsf{T}^*$ and $\alpha, \beta \in Act$, we write, $\beta \trianglelefteq_T \alpha$, if there exists no γ such that $T \xrightarrow{\gamma} T'$ and $\beta \subset \gamma \subseteq \alpha$.

Definition 2. *A relation $\mathcal{S} \subseteq \mathsf{T}^* \times \mathsf{S}^*$ is a* supply simulation relation *if for all $(T, S) \in \mathcal{S}$, $S \longrightarrow$, and, if $S \xrightarrow{\alpha} S'$ then*

1. *there exists $T \xrightarrow{\beta} T'$ with $\beta \subseteq \overline{\alpha}$ and $(T', S') \in \mathcal{S}$, and*
2. *if $T \xrightarrow{\beta} T'$ with $\beta \trianglelefteq_T \overline{\alpha}$, then $(T', S') \in \mathcal{S}$.*

If there exists a supply relation between T and S, then we write $S \models T$ and we say that S schedules *T.*

That is, a task and a supply are related by a supply simulation relation if (i) the supply is able to offer resources to the task ($S \longrightarrow$), (ii) if a supply offers a set of resources then the task will be able to respond by employing some (or all) of these resources and remain schedulable by the resulting state of the supply (clause 1), and (iii) given a set of resources offered by the supply, any maximal transition by which the task can accept the offered supply will result in a state that remains schedulable by the remaining supply (clause 2). Here, by a *maximal response* of the task, we mean all greedy transitions β by which the task can employ the offered resources α, that is, where $\beta \trianglelefteq_T \overline{\alpha}$. Note that any non-maximal transition of T taking place as a response to $S \xrightarrow{\alpha}$ would be subsequently pruned in the composition $S \| T$ as it would be preempted by greedier responses of T. Therefore, such transitions can be ignored.

We may now prove that the two alternative ways of defining schedulability of a task by a supply coincide.

Lemma 1. *A task $T \in \mathsf{T}^*$ is schedulable under supply S if and only if $S \models T$.*

Proof: To begin with, suppose there exists a supply simulation relation $\mathcal{R}$ between T and S. We will show that if $S \| T \xrightarrow{\alpha} S' \| T'$ then $S' \| T' \neq \delta$, $\alpha \cap \mathsf{R} = \emptyset$ and $(S', T') \in \mathcal{R}$. So suppose that $S \| T \xrightarrow{\alpha} S' \| T'$, $S \xrightarrow{\alpha_1} S'$ and $T \xrightarrow{\alpha_2} T'$, $\alpha = \alpha_1 \oplus \alpha_2$. We know that for some $\beta \subseteq \overline{\alpha_1}$, $T \xrightarrow{\beta} T''$ (Definition 2(1)). This implies that $\alpha_2 \subseteq \overline{\alpha_1}$ (otherwise $\alpha_1 \oplus \alpha_2 \prec \alpha_1 \oplus \beta$ and $S \| T \not\xrightarrow{\alpha}$). Consequently, we deduce that $\alpha \cap \mathsf{R} = \emptyset$. In addition, since T' is schedulable by S', by Definition 2 we have that $S' \longrightarrow$ and for each $S' \xrightarrow{\beta_1}$ there exists $T' \xrightarrow{\beta_2}$ such that $S' \| T' \longrightarrow$, that is, $S' \| T' \neq \delta$. And, finally, we may observe that there is no $T \xrightarrow{\gamma}$, $\alpha_2 \subset \gamma \subseteq \overline{\alpha_1}$ (otherwise $\alpha_1 \oplus \alpha_2 \prec \gamma \oplus \alpha_2$), which, by Definition 2(2), implies that $(S', T') \in \mathcal{S}$.

Conversely, suppose that task T is schedulable by supply S. We will show that $\mathcal{R} = \{(S, T) | S \text{ is schedulable by } T\}$ is a supply simulation relation. Suppose $(S, T) \in \mathcal{R}$. Since $S \| T \neq \delta$, $S \longrightarrow$. Furthermore, if $S \xrightarrow{\alpha} S'$ then $T \longrightarrow$. Since T is schedulable by S, there exists $T \xrightarrow{\beta} T'$, $\beta \subseteq \overline{\alpha}$. If not, that is for all $T \xrightarrow{\gamma} T''$, $\gamma \cap \overline{\alpha} \neq \gamma$, then $S \| T \xrightarrow{\alpha'}$, $\alpha' \cap \mathsf{R} \neq \emptyset$ which contradicts our assumption of T being schedulable by S. Next, suppose that $T \xrightarrow{\beta} T'$, $\beta \subseteq \alpha$ and for no $\beta \subset \gamma \subseteq \overline{\alpha}$. Then, clearly, $S \| T \xrightarrow{\alpha \oplus \beta} S' \| T'$, where T' is schedulable by S', which implies that $(S', T') \in \mathcal{R}$, as required. □

We define when a task is schedulable and this is done in the following obvious way.

Definition 3. *A task $T \in \mathsf{T}^*$ is* schedulable *if there exists a supply S with $S \models T$.*

We observe that the crux of the schedulability of a task by a supply lies in the capability of the task to operate acceptably for all possible behaviors of the supply. Furthermore, at each point during its execution and for each supply provision of the supplier, the task must behave well in all its nondeterministic executions that can take place by employing the resources available. The notion of a cylinder, defined below is intended to capture the relevant executions of the task given a behavior of the supply.

Definition 4. *Given a task $T \in \mathsf{T}^*$ and an infinite trace $w = \alpha_1\alpha_2\ldots$, we define the* w-cylinder *of T to be the set $A = \cup_{i\geq 1} A_i$, where*

$$A_1 = \{(T, \alpha_1, P_1) | T \xrightarrow{\alpha_1} P_1\}$$
$$A_i = \{(P_i, \beta_i, P_i') | P_i \xrightarrow{\beta_i} P_i', \beta_i \trianglelefteq_{P_i} \alpha_i, \exists (Q, \gamma, P_i) \in A_{i-1}\}, \quad i > 1$$

Furthermore, we say that an w-cylinder $A = \cup_{i\geq 1} A_i$ is live *if (i) A contains no triple of the form (Q, α, δ), (ii) $A_i \neq \emptyset$ for all i and (3) $\bigcup_{(P,\beta,Q)\in A_i} \beta = \alpha_i$.*

Lemma 2. *A task $T \in \mathsf{T}^*$ is schedulable if and only if it possesses a live cylinder.*

Proof: Suppose T has a live w-cylinder where $w = \alpha_1\alpha_2\ldots$. Consider supply S_0 defined by the following set of equations $S_i \stackrel{\text{def}}{=} \overline{\alpha_{i+1}}{:}S_{i+1}$. Then, we may confirm that $S_0 \models T$. The details are omitted. On the other hand, if T is schedulable, then there exists a supply S that schedules it. If we consider a trace $w = \overline{\alpha_1}\,\overline{\alpha_2}\ldots$ of S, we may construct an associated cylinder of T and confirm that it is live. □

3.1 Matching Supplies to Tasks

In this section we focus our attention to the problem of collecting the resource requirements of a task into a matching supply. Specifically, given a task, we would like to generate a supply process which schedules the task and at the same time is optimal in that (1) it does not reserve more resources than those required by the task and (2) it provides all the alternative resource assignments in which the task can be scheduled. Both of these properties are important during the compositional scheduling of real-time tasks. The first property is clearly desirable since conservation of resources becomes critical when real-time components are composed. For the second property, we observe that capturing all possible ways of scheduling a task gives flexibility when one tries to compositionally schedule a set of tasks where the challenge is to share the resources between the tasks in ways that are acceptable to each one of them.

We begin by defining a function on combining supplies. This is helpful for a subsequent definition that considers matching supplies to tasks.

Definition 5. *Given supplies S_1 and S_2 we define $S_1 \otimes S_2 =$*

$$S_1 \otimes S_2 = \begin{cases} S_1 & \text{if } S_2 = \text{FIN} \\ S_2 & \text{if } S_1 = \text{FIN} \\ \Sigma_{i\in I}\Sigma_{j\in J}\ \alpha_i \cup \beta_j{:}(\bigotimes_{k\in I, \alpha_k \trianglelefteq_{S_1} \alpha_i\cup\beta_j} P_k \otimes \bigotimes_{l\in J, \beta_l \trianglelefteq_{S_2} \alpha_i\cup\beta_j} Q_l) & \\ & \text{if } S_1 \stackrel{\text{def}}{=} \sum_{i\in I} \alpha_i{:}P_i \text{ and } S_2 \stackrel{\text{def}}{=} \sum_{j\in J} \beta_i{:}Q_i \end{cases}$$

Essentially, the joined supply $S_1 \otimes S_2$, joins together the various summands of the individual supplies as follows: in its topmost summand it unites all available grants of S_1 with all available grants of S_2, while the continuation process consists of the join of those continuations of S_1 and S_2 which appear after "maximal" subsets of the initial action in question. For example we have:

$$\begin{aligned}
&\emptyset : \{\overline{cpu}\} : \emptyset : \mathrm{FIN} \otimes \emptyset : \emptyset : \{\overline{cpu}\} : \mathrm{FIN} \quad = \quad \emptyset : \{\overline{cpu}\} : \{\overline{cpu}\} : \mathrm{FIN} \\
&\emptyset : \{\overline{cpu}\} : \emptyset : \mathrm{FIN} \otimes (\emptyset : \emptyset : \{\overline{cpu}\} : \mathrm{FIN} + \{\overline{cpu}\} : \emptyset : \emptyset : \mathrm{FIN}) \\
&\qquad\qquad = \quad \emptyset : \{\overline{cpu}\} : \{\overline{cpu}\} : \mathrm{FIN} + \{\overline{cpu}\} : \{\overline{cpu}\} : \emptyset : \mathrm{FIN}
\end{aligned}$$

Using this definition we now move to define the *demand* of a task. The demand of a task is intended to capture the optimal supply that can schedule a task in the sense we have already discussed. The main point to note in this definition is that we combine all same-prefixed nondeterministic choices of a task by a singly-prefixed supply.

Definition 6. *Given a task* $T \stackrel{\mathrm{def}}{=} \sum_{i \in I} \alpha_i{:}T_i$*, we define its* demand *as follows:*

$$\mathsf{demand}(T) \stackrel{\mathrm{def}}{=} \Sigma_{i \in I} \overline{\alpha_i}{:} \bigotimes_{j \in I, \alpha_i = \alpha_j} \mathsf{demand}(T_j)$$

Example 3. Consider tasks

$$\begin{aligned}
T_1 &= \{cpu\} : \emptyset : T_1 + \emptyset : \{cpu\} : T_1 \\
T_2 &= \{cpu\} : \emptyset : \emptyset : T_2 + \emptyset : \{cpu\} : \emptyset : T_2 + \emptyset : \emptyset : \{cpu\} : T_2 \\
T_3 &= \{cpu\} : \emptyset : \emptyset : T_3 + \emptyset : (\{cpu\} : \emptyset : T_3 + \emptyset : \{cpu\} : T_3)
\end{aligned}$$

Their demands are given by X_1, X_2 and X_3 below, respectively.

$$\begin{aligned}
X_1 &= \{\overline{cpu}\} : \emptyset : X_1 + \emptyset : \{\overline{cpu}\} : X_1 \\
X_2 &= \{\overline{cpu}\} : \emptyset : \emptyset : X_2 + \emptyset : \{\overline{cpu}\} : \{\overline{cpu}\} : X_2 \\
X_3 &= \{\overline{cpu}\} : \emptyset : \emptyset : X_3 + \emptyset : (\{\overline{cpu}\} : \emptyset : X_3 + \emptyset : \{\overline{cpu}\} : X_3)
\end{aligned}$$

□

The next lemma considers the optimality of $\mathsf{demand}(T)$ following the requirements posed at the beginning of this section. We write $w' \leq w$ for the infinite traces $w' = \alpha_1\alpha_2\ldots$ and $w = \beta_1\beta_2\ldots$, if either $w' = w$, or there exists j such that $\alpha_j \subset \beta_j$ and $\alpha_i = \beta_i$ for all $1 \leq i < j$.

Lemma 3. *A task T possesses a live w-cylinder if and only if there exists $w' \leq \overline{w}$ such that $w' \in \mathsf{traces}(\mathsf{demand}(T))$.*

Proof: Suppose $\mathsf{demand}(T) \xrightarrow{\overline{\alpha_1}} T_1 \xrightarrow{\overline{\alpha_2}} T_2 \xrightarrow{\overline{\alpha_3}} \ldots$. We may show that for the w-cylinder $A = \cup_{i \geq} A_i$, where $w = \alpha_1\alpha_2\ldots$ we have $T_i = \bigotimes_{(P,\beta,Q) \in A_i} \mathsf{demand}(Q)$ and A is live. The details are omitted.

To establish the opposite direction suppose $A = \cup_{i \geq} A_i$ is a live w-cylinder of T. Now consider the w'-cylinder of T, $B = \cup_{i \geq} B_i$, where $w' = \beta_1\beta_2\ldots$ is defined such

that $\beta_1 = \alpha_1$, $B_1 = A_1$ and $\beta_i \in \{\gamma_1 \cup \ldots \cup \gamma_n | P_i \xrightarrow{\gamma_i}, P_i \in \{P|(P,\alpha,Q) \in B_{i-1}\}\}$, $B_i = \{(P,\alpha,Q)|P \xrightarrow{\alpha} Q, \alpha \trianglelefteq_P \beta_i, \exists (Q,\gamma,P) \in B_{i-1}\}$. We may now prove, that $w' \in \mathsf{traces}(\mathsf{demand}(T))$ and, therefore, that it is a live cylinder of T. □

As a consequence of the result we conclude that a task T is schedulable by its demand. Furthermore, $\mathsf{demand}(T)$ may schedule all cylinders of T and it schedules them exactly, i.e. it offers exactly the resources that are necessary for the scheduling.

3.2 Compositional Theory

We proceed to consider the schedulability problem of a set of tasks. The first issue we tackle is the compositionality problem: If T_1 is schedulable by S_1 and T_2 by S_2 can we combine S_1 and S_2 into a supply that schedules $T_1 \| T_2$? We begin by noting a certain subtlety in this problem which we need to consider while answering it.

Consider the tasks

$$T_1 = \{r\}{:}\emptyset{:}\text{FIN} + \emptyset{:}\{r\}{:}\text{FIN and } T_2 = \{r\}{:}\emptyset{:}\text{FIN} + \emptyset{:}\{r\}{:}\{r\}{:}\text{FIN}.$$

These tasks are schedulable under supplies $S_1 = \emptyset{:}\{\overline{r}\}{:}\text{FIN}$ and $S_2 = \{\overline{r}\}{:}\emptyset{:}\text{FIN}$, respectively. That is, it is sufficient for task T_1 to obtain resource r during the second time unit and for task T_2 during the first time unit. However, a supply $S = \{\overline{r}\}{:}\{\overline{r}\}{:}\text{FIN}$, offering r during both time units, fails to schedule $T_1 \| T_2$. This is due to the fact that the supply for resource r during the first time unit is intended for task T_2 but may be consumed by task T_1 leading to a deadlock of the system during the third time unit.

To resolve this issue, we associate tasks with their matching supplies by annotating each resource reference by a number which distinguishes the task in which the resource is employed/supplied. Precisely, we assume that each task is associated with a resource identity and if resource r is requested by a task with identifier i we write $r[i]$ for the request and, similarly, if a supply of r is intended for the task with identifier i we write $\overline{r[i]}$ for the supply. So, we say that task $\{r[1]\}$:FIN is schedulable by supply $\{\overline{r[1]}\}$:FIN and task $\{r[2]\}$:FIN by supply $\{\overline{r[2]}\}$:FIN. However, note that resources $r[1]$ and $r[2]$ do refer to the same resource and for all other purposes should be treated as the same. So, for example, $\{r[1]\} \cap \{r[2]\} \neq \emptyset$. To model this precisely we write:

- $P[i]$ for the process P with all its resources r renamed as $r[i]$.
- $\alpha \cap_R \beta$ for $\{r \in R | r[i] \in \alpha, r[j] \in \beta, \text{ or } \overline{r[i]} \in \alpha, \overline{r[j]} \in \beta\}$

Furthermore, we use the notation $\alpha[i] = \{r|r[i] \in \alpha\}$ and, if $w = \alpha_1\alpha_2 \ldots$, $w[i] = \alpha_1[i]\alpha_2[i] \ldots$. We have the following result:

Lemma 4. *If T_1 is schedulable by S_1, T_2 is schedulable by S_2 and $S_1 \| S_2$ does not deadlock, then $T_1[1] \| T_2[2]$ is schedulable by $S_1[1] \| S_2[2]$.*

Proof: We will show that $\mathcal{R}$, below, is a supply simulation relation.

$$\mathcal{R} = \{(T_1[1] \| T_2[2], S_1[1] \| S_2[2]) | S_1 \models T_1, S_2 \models T_2, S_1[1] \| S_2[2] \text{ does not deadlock}\}$$

Let $(T_1[1] \| T_2[2], S_1[1] \| S_2[2]) \in \mathcal{R}$. By the definition of $\mathcal{R}$, $S_1[1] \| S_2[2] \longrightarrow$. So consider $S_1[1] \| S_2[2] \xrightarrow{\alpha} S_1'[1] \| S_2'[2]$. It must be that $\alpha = \alpha_1[1] \oplus \alpha_2[2]$, where $S_1 \xrightarrow{\alpha_1} S_1'$, $S_2 \xrightarrow{\alpha_2} S_2'$ and $\alpha_1 \cap \alpha_2 = \emptyset$. Since $S_1 \models T_1$, $S_2 \models T_2$, we have $T_1 \xrightarrow{\beta_1} T_1'$, $S_1' \models T_1'$,

and similarly $T_2 \xrightarrow{\beta_2} T_2'$, $S_2' \models T_2'$. In fact, for all $T_1 \xrightarrow{\beta_1} T_1'$, $\beta_1 \trianglelefteq_{T_1} \overline{\alpha_1}$, it holds that $S_1' \models T_1'$, and for all $T_2 \xrightarrow{\beta_2} T_2'$, $\beta_2 \trianglelefteq_{T_2} \overline{\alpha_2}$, it holds that $S_2' \models T_2'$. This implies that for all $T_1[1]\|T_2[2] \xrightarrow{\beta} T_1'[1]\|T_2'[2]$, $\beta \trianglelefteq_{T_1[1]\|T_2[2]} \overline{\alpha}$, $(T_1'[1]\|T_2'[2], S_1'[1]\|S_2'[2]) \in \mathcal{R}$ and there exists at least one such α-transition. This completes the proof. □

However, note that even if $S_1\|S_2$ deadlocks, it is still possible that the schedules S_1 and S_2 can be combined to produce a schedule for $T_1\|T_2$. In particular, we may suspect that every infinite trace of $S_1\|S_2$ is capable of scheduling $T_1\|T_2$, and in fact we can show that the part of the transition system that pertains to non-deadlocking behavior achieves exactly that. The following operator on supplies extracts this type of behavior.

$$S_1 \times S_2 = \begin{cases} S_1 & \text{if } S_2 = \text{FIN} \\ S_2 & \text{if } S_1 = \text{FIN} \\ (\alpha \cup \beta){:}(S_1' \times S_2') & \text{if } S_1 = \alpha{:}S_1', S_2 = \beta{:}S_2', \alpha \cap \beta = \emptyset, S_1' \times S_2' \neq \delta \\ \delta & \text{if } S_1 = \alpha{:}S_1', S_2 = \beta{:}S_2', \alpha \cap \beta \neq \emptyset \text{ or } S_1' \times S_2' = \delta \\ \Sigma_{i\in I, j\in J}(S_1^i \times S_2^j) & \text{if } S_1 = \Sigma_{i\in I} S_1^i, S_2 = \Sigma_{j\in J} S_2^j \end{cases}$$

Note that the set of recursive equations used in the definition of $S_1 \times S_2$ may allow more than one solution. Consider, for example, $S_1 = \{\overline{r_1}\} : S_1$ and $S_2 = \{\overline{r_2}\} : S_2$. It is easy to see that $S_1 \times S_2 = \delta$ is a trivial solution. However, we are interested in the maximal solution to this set of equations, which in this case is $S_1 \times S_2 = \{\overline{r_1}, \overline{r_2}\} : S_1 \times S_2$. For finite-state processes, the maximal solution can be computed iteratively. Due to space restrictions the proof is omitted.

It is easy to see that, if $S_1\|S_2$ does not deadlock then $S_1 \times S_2 \neq \delta$. However, the opposite is not true. By the construction of $\times$, $S_1 \times S_2$ selects the part of the transition system of $S_1\|S_2$ that does not lead to deadlocked states. For example, consider

$$S_1 \stackrel{\text{def}}{=} \{r\}{:}\{r\}{:}\text{FIN} + \emptyset{:}\{r\}\{r\}{:}\text{FIN} \quad \text{and} \quad S_2 \stackrel{\text{def}}{=} \emptyset{:}\{r\}{:}\text{FIN} + \{r\}{:}\emptyset{:}\text{FIN}$$

Then, although $S_1\|S_2 \xrightarrow{\{r\}} \{r\}{:}\text{FIN}\|\{r\}{:}\text{FIN} = \delta$, $S_1 \times S_2 = \{r\}{:}(\{r\} : \{r\}{:}\text{FIN} \times \emptyset{:}\text{FIN})$, and $(\{r\}\{r\}{:}\text{FIN} \times \emptyset{:}\text{FIN}) = \{r\}\{r\}{:}\text{FIN}$.

Lemma 5. *If T_1 is schedulable by S_1, T_2 is schedulable by S_2 and $S_1 \times S_2 \neq \delta$, then $T_1[1]\|T_2[2]$ is schedulable by $S_1[1] \times S_2[2]$.*

Proof: The proof is similar to that of the previous lemma. □

At this point we turn our attention to the problem of constructing an interface for a set of mutually schedulable tasks. To do this, we employ the notion of demands and we prove the following:

Lemma 6. *If $T_1[1]\|T_2[2]$ has a live w-cylinder then there exists a trace $w' \leq \overline{w}$ such that $w' \in \mathsf{traces}(\mathsf{demand}(T_1[1]) \times \mathsf{demand}(T_2[2]))$.*

Proof: Suppose that the w-cylinder of $T_1[1]\|T_2[2]$ is live. It is easy to see that $w[1]$ and $w[2]$ give rise to live cylinders in $T_1[1]$ and $T_2[2]$. Then, by Lemma 3, there exist $w_1 \leq \overline{w[1]}$ and $w_2 \leq \overline{w[2]}$ such that $w_1 \in \mathsf{traces}(\mathsf{demand}(T_1[1]))$ and $w_2 \in \mathsf{traces}(\mathsf{demand}(T_2[2]))$. This implies that, $w_1 \cup w_2 \leq \overline{w}$ is a trace of $\mathsf{demand}(T_1[1]) \times \mathsf{demand}(T_2[2])$, as required. □

This result implies that all alternatives of scheduling $T_1[1] \| T_2[2]$ will be explored by $\mathsf{demand}(T_1[1]) \times \mathsf{demand}(T_2[2])$. It can be extended to the composition of an arbitrary number of tasks. We are now ready to present our main theorem:

Theorem 1. *$T_1 \| T_2$ is schedulable if and only if* $\mathsf{demand}(T_1[1]) \times \mathsf{demand}(T_2[2]) \neq \delta$. *Moreover, if it is schedulable, then it is schedulable by* $\mathsf{demand}(T_1[1]) \times \mathsf{demand}(T_2[2])$.

Proof: Suppose $T_1[1] \| T_2[2]$ is schedulable. Then, by Lemma 2 it has a live w-cylinder. Consequently, by Lemma 6 there is a trace $w' \leq w$ such that the w' is a trace of $\mathsf{demand}(T_1[1]) \times \mathsf{demand}(T_2[2])$. This implies that $\mathsf{demand}(T_1[1]) \times \mathsf{demand}(T_2[2]) \neq \delta$. On the other hand, if $\mathsf{demand}(T_1[1]) \times \mathsf{demand}(T_2[2]) \neq \delta$, then, since, additionally, $\mathsf{demand}(T_1[1])$ schedules $T_1[1]$ and $(T_2[2])$ schedules $T_2[2]$, then, by Lemma 5, $T_1[1] \| T_2[2]$ is schedulable by $\mathsf{demand}(T_1[1]) \times \mathsf{demand}(T_2[2])$. □

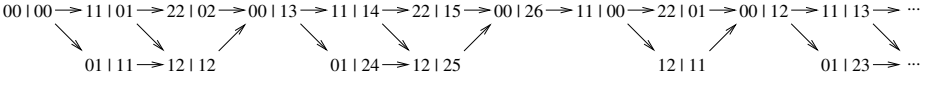
Based on this result we may determine the schedulability and a related scheduler for a set of tasks $T_1, \ldots, T_n$, as follows: For each task, extract its demand and compute the combinations $D_1 = \mathsf{demand}(T_1) \times \mathsf{demand}(T_2)$, $D_2 = D_2 \times \mathsf{demand}(T_3), \ldots$. If this process does not reduce to some $D_i = \delta$ then the tasks are schedulable by D_{n-1}. Furthermore, according to Theorem 1, if they are indeed schedulable then $D_{n-1} \neq \delta$. Thus, this method is guaranteed to produce a schedule if one exists.

4 Examples

Example 4. We first define a simple periodic task with period p and execution time w, $Task_{w,p} = T_{0,0,w,p}$, as follows:

$$T_{e,t,w,p} = \begin{cases} \emptyset : T_{e,t+1,w,p} & \text{if } e = w, t < p \\ T_{0,0,w,p} & \text{if } e = w, t = p \\ \emptyset : T_{e,t+1,w,p} + \{r\} : T_{e+1,t+1,w,p} & \text{if } e < w, w - e < p - t \\ \{r\} : T_{e+1,t+1,w,p} & \text{if } e < w, w - e = p - t \end{cases}$$

Note that in our definition, the task cannot idle if idling will make it miss the deadline. If the supply can avoid giving the resource to the task in this case, the system will have an unmet resource request transition that signals non-schedulability (by Definition 1). Let us consider an instance of a classical scheduling problem for a set of periodic tasks running on a single processor resource: $Task_{2,3} \| Task_{2,7} \| S$, where $S = \{\overline{r}\} : S$. In the figure below, we show the initial part of the state space of the example. Each state is represented as a tuple $ij|km$, where i and j are the first two parameters of the first task and k and m are the first two parameters of the second task. The other two parameters do not change and are omitted to avoid cluttering the figure. We also omit labels on the transitions: all transitions are labeled by $\{\overleftrightarrow{r}\}$.



The tasks are schedulable according to the Definition 1 and the transition system of the composite process, shown above, can be seen as the specification of feasible schedulers for the task set. Non-determinism in the transition system represent different

decisions that a scheduler can make. For example, the trace along the top of the figure corresponds to the rate-monotonic scheduling policy, which gives priority to $Task_{2,3}$ as it has the smallest period.

We now consider the demand of a periodic task defined above. It is easy to see that the task process is *resource-deterministic*, that is, its behavior is determined by the availability of resources. For a resource-deterministic task, the demand is obtained by a straightforward replacement of requested resources by matching offered resources. Thus, $\mathsf{demand}(Task_{w,p}) = X_{0,0,w.p}$ is defined below:

$$X_{e,t,w,p} = \begin{cases} \emptyset : X_{e,t+1,w,p} & \text{if } e = w, t < p \\ X_{0,0,w,p} & \text{if } e = w, t = p \\ \emptyset : X_{e,t+1,w,p} + \{\overline{r}\} : X_{e+1,t+1,w,p} & \text{if } e < w, w - e < p - t \\ \{\overline{r}\} : X_{e+1,t+1,w,p} & \text{if } e < w, w - e = p - t \end{cases}$$

It is easy to check that $\mathsf{demand}(Task_{2,3}) \| \mathsf{demand}(Task_{2,7})$ does not deadlock and thus can schedule the two tasks according to Lemma 4.

Let us now consider a task with variable execution time which takes between b and w time units to complete: $Task^v_{b,w,p} = Task_{b,p} + Task_{b+1,p} + \ldots + Task_{w,p}$. One can see that $\mathsf{demand}(Task^v_{b,w,p}) = \mathsf{demand}(Task_{w,p})$. This observation matches the well-known fact from the real-time systems theory that for independent periodic tasks it is sufficient to consider worst-case execution time of each task [17].

Example 5. To illustrate compositional analysis with partial supplies, we begin with a simple example of time-partitioned supplies that are widely used in practice. Consider a periodic time partition with period P, duration $D \leq P$, and relative start time t_0, which essentially offers a resource r for the interval $[t, t + D)$ during each period: $Part_{t_0,D,P} = P_{0,t_0,D,P}$ is defined as follows where, again, addition is modulo P:

$$P_{t,t_0,D,P} = \begin{cases} \{\overline{r}\} : P_{t+1,t_0,D,P} & \text{if } t_0 \leq t < t_0 + D \\ \emptyset : P_{t+1,t_0,D,P} & \text{otherwise} \end{cases}$$

It is clear that partitions with the same period and non-overlapping service intervals $[t, t + D)$ do not conflict. We can now analyze schedulability of tasks allocated to a partition separately from any other task in the system. It is, for example, trivial to see that partition $Part_{t_0,D,P}$ can schedule a task $Task_{D,P}$ for any t_0.

We can similarly define more complex partial supplies. Consider, for example, compositional scheduling based on periodic resource models [21,22]. A periodic resource model is a supply that guarantees w units of resource execution within a period P, however, the availability of the resource within the period is unknown *a priori*. We can straightforwardly model a periodic resource model as $PRM_{w,P} = \mathsf{demand}(Task_{w,P})$. We can then analyze whether a set of tasks is schedulable with respect to this supply. This analysis will not be limited to independent periodic or sporadic tasks, unlike existing approaches in the literature.

As an example, consider the system $T_1 = Task_{1,3} \| Task_{1,5} \| PRM_{3,5}$. Figure 1 shows the initial state space using the same notation as above, except now the state tuple also includes the state of the supply. Note that, in this transition system we have actions pertaining to resource consumption, abbreviated by $\overleftrightarrow{r}$, actions pertaining to resource

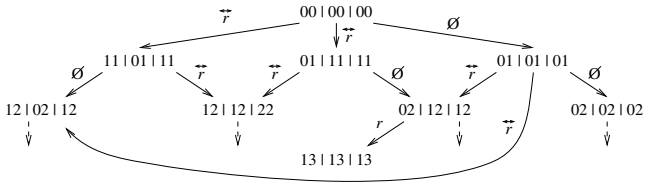


Fig. 1. Scheduling with a periodic resource

requests, abbreviated by $\overline{r}$, and idling actions. Recall that idling and consumed resource actions are incomparable in the preemption relation, while idling preempts unsatisfied resource requests. We see that a poor scheduling decision can make $Task_{1,3}$ miss its deadline. The scenario is seen on the right side of the figure: in the first two time units, one unit of resource goes to $T_{1,5}$ and the other unit of resource is denied to both tasks (this can happen in any order). If on the third step the supply denies access to the resource again, the first task cannot idle, thus we reach a transition labeled by $\{r\}$, which implies that the task misses its deadline, leading to a violation of Definition 1.

5 Conclusions

In this paper, we have presented PADS, a process algebra for resource demand and supply. The algebra can be used to describe a process and its demand on resources necessary for the execution of a real-time task as well as a supply process that describes the behavior of a resource allocator. We have defined precisely the notion of schedulability using demand and supply, that is, when a process can be scheduled under a supply process, and provided a compositional theory of demand-supply schedulability. We believe that PADS is the first process algebra that can describe the behavior of demand and supply processes and compositional schedulability between them.

There are several directions in which the current work can be extended. We are currently adding priorities to resource requests in the same way as in [13]. This allows us to represent schedulability with respect to particular schedulers, which is often a more practical question to analyze. We plan to extend the framework with the notion of order between supplies. This notion will capture the "generosity" of a supply, that is, a more generous supply will be able to schedule any task that the less generous supply can. With this notion, we will be able to formally represent the hierarchical scheduling approaches based on resource models [21] that rely on approximating the necessary supply, making it more generous than necessary, in exchange for a simple representation. It would also be interesting to explore how to extend the notion of schedulability to the notion of *resource satisfiability* between demand and supply of arbitrary resources that are not shared mutually exclusively. Another extension is to explore demand and supply processes in the presence of probabilistic behavior.

References

1. Altisen, K., Gößler, G., Pnueli, A., Sifakis, J., Yovine, Y.: A framework for scheduler synthesis. In: Proceedings of RTSS'99, pp. 154–163. IEEE Computer Society, Los Alamitos (1999)

2. Altisen, K., Gößler, G., Sifakis, J.: Scheduler modeling based on the controller synthesis paradigm. Real-Time Systems 23(1-2), 55–84 (2002)
3. Ben-Abdallah, H., Choi, J.-Y., Clarke, D., Kim, Y.S., Lee, I., Xie, H.-L.: A Process Algebraic Approach to the Schedulability Analysis of Real-Time Systems. Real-Time Systems 15, 189–219 (1998)
4. Bucci, G., Fedeli, A., Sassoli, L., Vicario, E.: Modeling flexible real time systems with preemptive time Petri nets. In: Proceedings of ECRTS'03, pp. 279–286. IEEE Computer Society, Los Alamitos (2003)
5. Deng, Z., Liu, J.W.-S.: Scheduling real-time applications in an open environment. In: Proceedings of RTSS'97, pp. 308–319. IEEE Computer Society Press, Los Alamitos (1997)
6. Easwaran, A., Anand, M., Lee, I.: Compositional analysis framework using edp resource models. In: Proceedings of RTSS'07, pp. 129–138. IEEE Computer Society, Los Alamitos (2007)
7. Easwaran, A., Lee, I., Sokolsky, O.: Interface algebra for analysis of hierarchical real-time systems. In: Proceedings of FIT (2008)
8. Feng, X., Mok, A.: A model of hierarchical real-time virtual resources. In: Proceedings of RTSS'02, pp. 26–35. IEEE Computer Society, Los Alamitos (2002)
9. Fersman, E., Krcál, P., Pettersson, P., Yi, W.: Task automata: Schedulability, decidability and undecidability. Information and Computation 205(8), 1149–1172 (2007)
10. Fersman, E., Pettersson, P., Yi, W.: Timed automata with asynchronous processes: Schedulability and decidability. In: Katoen, J.-P., Stevens, P. (eds.) TACAS 2002. LNCS, vol. 2280, pp. 67–82. Springer, Heidelberg (2002)
11. GmbH, R.-T.S.: Real-Time Hybervisor (2010), `http://www.real-time-systems.com`
12. Henzinger, T.A., Matic, S.: An interface algebra for real-time components. In: Proceedings of RTAS'06, pp. 253–263. IEEE Computer Society, Los Alamitos (2006)
13. Lee, I., Brémond-Grégoire, P., Gerber, R.: A Process Algebraic Approach to the Specification and Analysis of Resource-Bound Real-Time Systems. Proceedings of the IEEE, 158–171 (1994)
14. Lee, I., Philippou, A., Sokolsky, O.: Resources in process algebra. Journal of Logic and Algebraic Programming 72, 98–122 (2007)
15. Linuxworks. LynxSecure Embedded Hypervisor and Separation Kernel (2010), `http://www.lynuxworks.com/virtualization/hypervisor.php`
16. Lipari, G., Bini, E.: Resource partitioning among real-time applications. In: Proceedings of ECRTS'03, pp. 151–160. IEEE Computer Society, Los Alamitos (2003)
17. Liu, J.: Real-time systems. Prentice Hall, Englewood Cliffs (2000)
18. Mousavi, M., Reniers, M., Basten, T., Chaudron, M.: PARS: a process algebra with resources and schedulers. In: Larsen, K.G., Niebert, P. (eds.) FORMATS 2003. LNCS, vol. 2791, pp. 134–150. Springer, Heidelberg (2004)
19. Nunez, M., Rodriguez, I.: PAMR: A process algebra for the management of resources in concurrent systems. In: Proceedings of FORTE'01, pp. 169–184 (2001)
20. Saewong, S., Rajkumar, R., Lehoczky, J., Klein, M.: Analysis of hierarchical fixed-priority scheduling. In: Proceedings of ECRTS'02, pp. 173–181. IEEE Computer Society, Los Alamitos (2002)
21. Shin, I., Lee, I.: Periodic resource model for compositional real-time guarantees. In: Proceedings of RTSS'03, pp. 2–13. IEEE Computer Society, Los Alamitos (2003)
22. Shin, I., Lee, I.: Compositional real-time scheduling framework. In: Proceedings of RTSS'04, pp. 57–67. IEEE Computer Society, Los Alamitos (2004)
23. Thiele, L., Wandeler, E., Stoimenov, N.: Real-time interfaces for composing real-time systems. In: Proceedings of EMSOFT'06. ACM, New York (2006)

Memory Event Clocks

James Jerson Ortiz[1], Axel Legay[2,3], and Pierre-Yves Schobbens[1]

[1] Computer Science Faculty, University of Namur
[2] INRIA/IRISA, Rennes
[3] Institut Montefiore, University of Liège
{james.ortizvega,pierre-yves.schobbens}@fundp.ac.be, alegay@irisa.fr

Abstract. We introduce logics and automata based on memory event clocks. A memory clock is not really reset: instead, a new clock is created, while the old one is still accessible by indexing. We can thus constrain not only the time since the last reset (which was the main limitation in event clocks), but also since previous resets. When we introduce these clocks in the linear temporal logic of the reals, we create Recursive Memory Event Clocks Temporal Logic (RMECTL). It turns out to have the same expressiveness as the Temporal Logic with Counting (TLC) of Hirshfeld and Rabinovich. We then examine automata with recursive memory event clocks (RMECA). Recursive event clocks are reset by simpler RMECA, hence the name "recursive". In contrast, we show that for RMECA, memory clocks do not add expressiveness, but only concision. The original RECA define thus a fully decidable, robust and expressive level of real-time expressiveness.

1 Introduction

Finite automata is a widely used computational model to capture and analyse the behavior of possibly concurrent systems. The main question is checking whether an automaton satisfies a given specification, which can be represented either by some temporal logic formula or by another automaton. The first case, called model checking, is usually reduced to the second. In the second case, the problem is called language inclusion between automata, which models step-wise refinement.

Nowadays, real-time plays a crucial role in system design, especially in the area of embedded systems. To capture the behavior of a real-time system, one needs to augment the computational model with a notion of time. An important model is timed automata (TA) [1], that are automata augmented with clocks used to monitor the evolution of time. Timed automata offer tools [21,9,6] for many real-time problems. Unfortunately, TA have an undecidable language inclusion problem [1]. Around the same time, the satisfiability of natural real-time logics such as Metric Temporal Logic (MTL) and Temporal Propositional Timed Logic (TPTL) were also proved undecidable [5]. In fact, one of the central problems is that TA are not closed under determinization (see [16,15,7] for discussions). The situation contrasts strongly with the one of automata without real time, where

K. Chatterjee and T.A. Henzinger (Eds.): FORMATS 2010, LNCS 6246, pp. 198–212, 2010.

the problems of complementation, language inclusion, emptiness, union and intersection are decidable, as well as the satisfiability and validity of propositional linear temporal logic (LTL). When all these problems are decidable, we call the formalism (automata or logic) fully decidable. These negative results spurred a quest for expressive but still fully decidable formalisms.

To overcome the problem, [3] proposed to restrict the behavior of TA clocks in such a way that language inclusion becomes decidable. The key idea is that the problematic clocks of TA are reset by non-deterministic, internal transitions, that prevent determinization. In contrast, an event clock (EC) x_p is reset when the atomic proposition p occurs. The event clock resets and values are determined by the input and thus Event Clock Automata (ECA) are determinizable, making language inclusion decidable and thus enabling step-wise refinement.

Event clocks can also be introduced in temporal logics [20]. An event clock constraint is naturally translated into a proposition $\lhd_I p$, that means "the last time that a p occurred was a time d ago, where d lies in I". However, the expressiveness of ECA is rather weak. Indeed, events are just the last or next occurrence of an atomic proposition. For instance, the property "p is continuously true in interval $(0, 1)$" cannot be expressed by such an event clock formula: any model where the distances between p's are below 1 will see the clocks always below 1, whether or not it satisfies this property. Therefore [11] introduced the notion of "recursive" event. In a recursive event model, the reset of a clock is decided by a lower-level automaton or formula. This automaton cannot read the clock that it is resetting. Clock resets are thus still deterministic, but the concept of "event" is now much more expressive. For instance, the property above can be expressed as $\neg \rhd_{(0,1)} \neg p$: $\rhd_I$ is now a modality that can contain any subformula, and can be nested. The temporal logic of recursive event clocks (variously called SCL [20] or EventClockTL [11]) has the same expressiveness as Metric Interval Temporal Logic MITL [2] (a decidable fragment of MTL where punctual constraints are forbidden) in the interval semantics. First-and second-order monadic logics with matching expressiveness have been provided [11], yielding a natural, robust, fully decidable level of real-time expressiveness. However, the expressiveness of event clock models has still been criticized, because event clocks can only constrain the time since the last (or next) event. For instance, EventClockTL cannot express the assumption that no more than 3 requests per second will arrive, or the requirement that these all requests will be treated within the next second, when the treatment requires several steps.

In this paper, we address the above limitation and introduce memory clocks, already sketched in [3]: "We could employ a clock x_a^i that records the time since the i-th-to-last occurrence of a". A memory clock x is not really reset: instead, a new clock is created, while the old one is still accessible by indexing: x^1 will be the usual value of the clock x, i.e. the time since last reset, while x^2 will be the time since the last but one reset. In general, x^i will be the time since the last but i reset. Said otherwise, a reset will save a copy of the clock of index i in the clock of index $i+1$. It can be seen as a series of clock updates: $x^3 := x^2; x^2 := x^1; x^1 := 0$. Here, we will study the recursive variant, as explained above.

Our first contribution is to extend the EventClockTL logic with memory clocks. This gives us the memory event clocks logic (RMECTL), which we show to be PSPACE-complete if the indices of the clocks are encoded in unary and EXPSPACE-complete for the binary case. RMECTL is strictly more expressive than EventClockTL. To obtain these results, we show that the expressiveness of RMECTL is equivalent to the one of the Temporal Logic with Counting (TLC) [13]. It is worth observing that TLC was inspired by a TPTL formula (see Section 3.3). TPTL is a "really temporal" but highly undecidable logic [5]. We isolate a decidable fragment of TPTL, which we call TPTL1R. It also has the same expressive power, showing the robustness of this level of expressiveness. Our second contribution is to extend RECA with memory clocks. Surprisingly, RMECA are as expressive as the original Recursive Event Clock Automata RECA [11]. However, in the binary case, they may be exponentially more succinct.

Structure of the paper. The rest of the paper is organized as follows. Sections 2 recalls preliminary notions. Section 3 examines real-time temporal logics. First, it recalls TLC [18], then introduce RMECTL and shows its equivalence with TLC. Then, it defines a fragment of TPTL that also has the same expressiveness. Section 4 defines Recursive Memory Event Clock Automata (RMECA), studies their properties, and concludes that they can be reduced to good old RECA [11].

2 Preliminaries

We briefly recall the various models of time that are used in the literature [4]. We present our results in the interval semantics, that is the richest and most natural (but also most difficult) model. We also recall clocks and their constraints.

2.1 Models of Time

Models of time can be linear, considering a single future, or branching, considering several alternative futures. We only consider linear time in this paper. Classical automata and LTL also use a linear discrete model of time. The *point semantics* adds a time stamp to each event of this discrete model.

Our goal here is to model real-time reactive systems, and thus we will use the real numbers as our model of time. This avoid a premature commitment to a discretization of time: even if computer systems are often discrete, their discretization grain (e.g. clock speed) should not appear at requirements level.

Let $\mathbb{P}$ be a set of propositional symbols. A state over $\mathbb{P}$ is an element of $2^{\mathbb{P}}$. Let $\mathbb{N}$ the set of nonnegative integers, $\mathbb{R}$ denote the set of reals, $\mathbb{R}^+$ the set of nonnegative reals.

In this paper, we use the *interval semantics*. An interval is a convex subset of $\mathbb{R}^+$. An interval is *singular* if it is a singleton. Two intervals I and I' are said to be *adjacent* when $I \cap I' = \emptyset$ and $I \cup I'$ is an interval. We denote by $\mathcal{I}_{\mathbb{R}^+}$ the set of intervals whose bounds are in $\mathbb{R}^+$. An interval sequence over $\mathbb{R}^+$ is an infinite sequence $I = I_0 I_1 \cdots$ of non-empty intervals of $\mathcal{I}_{\mathbb{R}^+}$ where

1. successive intervals I_j and I_{j+1} are adjacent and $I_j < I_{j+1}$, for all $j \geq 0$
2. I is *covering*, i.e., for every $t \in \mathbb{R}^+$, there exists $j \in \mathbb{N}$ such that $t \in I_j$.

An interval state sequence (ISS) is a pair $\rho = (\sigma, I)$ where $\sigma = \sigma_0\sigma_1 \cdots$ is an infinite sequence of states and $I = I_0 I_1 \cdots$ is an *interval sequence*. A interval state sequence ρ can equivalently be seen as an sequence of elements in $2^{\mathbb{P}} \times \mathcal{I}_{\mathbb{R}^+}$. It can also be seen as a *signal*, i.e. a function from $\mathbb{R}^+$ to states: Let $\rho = (\sigma, I)$ be a interval state sequence and given $t \in \mathbb{R}^+$, let $i \in \mathbb{N}$ be the interval such that $t \in I_i$. We define $\rho(t)$ as the state σ_i. A signal derived from an ISS will always have finite variability. Below, our automata will consider two ISS that define the same signal as equivalent, even if the intervals might be split differently. Our automata assume finite variability. In contrast, our logics will admit infinite variability.

Given two intervals I_1, I_2, we define the interval between I_1 and I_2 by $BetwI(I_1, I_2) = \{x \mid I_1 < x < I_2\}$.

Given a set S and an interval I, we define S *Begins During* I by $\exists t \in (S \cap I)$, and $\nexists t' \in S$ such that $t' < I$. Symmetrically, we define S *Ends During* I iff $\exists\ t$, $t \in (S \cap I)$, and $\nexists\ t' \in S$ such that $t' > I$.

2.2 Clocks

The value of a clock is the time elapsed since its last reset. When we use real numbers, there is not always a "last" reset but just a limit, e.g. when the reset holds in an open interval. For this case, we will use non-standard clock values of the form v^+. The set of non-standard reals, noted $\mathbb{R}^+_{ns}$, is the set of $\{v, v^+ \mid v \in \mathbb{R}^+\}$, ordered by $<_{ns}$ as following: $v_1 <_{ns} v_2^+$ iff $v_1 \leq v_2$. $\mathbb{R}^+_\perp$ is $\mathbb{R}^+_{ns}$ plus a special value $\perp$ for uninitialized clocks. $\perp$ is not comparable to other values.

Let X be a finite set of clock names. A clock valuation over X is a mapping $\nu : X \to \mathbb{R}^+_\perp$. The *constraints* over X, noted $\Phi(X)$, are defined by the following grammar, where ϕ ranges over $\Phi(X)$, $x \in X$, $c \in \mathbb{N}$, and $\sim \in \{<, \leq, =, >, \geq\}$:

$$\phi ::= true \mid x \sim c \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \neg\phi$$

We write $\nu \models \phi$ when the valuation ν satisfies the constraint ϕ. By convention, the value $\perp$ does not satisfy any constraint except *true*.

3 Temporal Logics

As we said in the Introduction, the goal of our quest is to construct two levels of expressiveness: (i) fully decidable real-time logics to specify requirements on systems; we examine them in this section; (ii) fully decidable real-time automata, that can express these logics, and model systems (see Section 4). In this way, specifications and systems can be handled uniformly, and verification can be automated. We first recall Temporal Logic with Counting (TLC) [13], since our new logics will turn out have the same expressiveness. Then we define the new logics: RMECTL, that includes memory event clocks, and TPTL1R, a decidable fragment of the logic TPTL with the same expressiveness.

3.1 Temporal Logic with Counting

The Temporal Logic with Counting [13] is an extension of the Temporal Logic of the Reals with Past by *counting modalities*. Here we use a slight variant, called $TLCI_0$ [18], where the counting modalities are $C_k^{(0,b)}(\phi)$ and $\overleftarrow{C}_k^{(0,b)}(\phi)$. The modality $C_k^{(0,b)}(\phi)$ says that ϕ will be true at least at k points in the interval $(t, t+b)$, and its symmetrical $\overleftarrow{C}_k^{(0,b)}(\phi)$ says that ϕ has happened k times in the interval $(t-b, t)$. The syntax of $TLCI_0$ formulae is given by:

$$\phi ::= \ p \mid \phi_1 \ \wedge \ \phi_2 \mid \neg \ \phi \mid \phi_1 \ \mathcal{U} \ \phi_2 \mid \phi_1 \ \mathcal{S} \ \phi_2 \mid C_k^{(0,b)}(\phi) \mid \overleftarrow{C}_k^{(0,b)}(\phi)$$

where p is a propositional symbol, $k \in \mathbb{N}, b \in \mathbb{N}$ and $\phi_1, \phi_2 \in TLCI_0$.

Formally, the semantics is as follows:

$$(\rho, t_0) \models C_k^{(0,b)}(\phi) \ \text{ iff } \ \exists t_1 \cdots \exists t_k, t_0 < t_1 < \cdots < t_k < t_0 + b \quad \wedge \bigwedge_{0<i\leq k} \phi(t_i)$$

$$(\rho, t_0) \models \overleftarrow{C}_k^{(0,b)}(\phi) \quad \text{iff} \quad \exists t_1 \cdots \exists t_k, t_0 - b < t_1 < \cdots < t_k < t_0 \ \wedge \bigwedge_{0<i\leq k} \phi(t_i)$$

Let us also recall the classical semantics, that we will use throughout the paper:

$$\begin{aligned}
(\rho, t) &\models \quad p \quad && \text{iff } p \in \rho(t) \\
(\rho, t) &\models \quad \neg\phi \quad && \text{iff } (\rho, t) \not\models \phi \\
(\rho, t) &\models \phi_1 \wedge \phi_2 \ && \text{iff } (\rho, t) \models \phi_1 \text{ and } (\rho, t) \models \phi_2 \\
(\rho, t) &\models \ \phi_1 \mathcal{U} \phi_2 \ && \text{iff } \exists t' > t. (\rho, t') \models \phi_2 \wedge \forall t'' \in (t, t'), (\rho, t'') \models \phi_1 \\
(\rho, t) &\models \ \phi_1 \mathcal{S} \phi_2 \ && \text{iff } \exists t' < t. (\rho, t') \models \phi_2 \wedge \forall t'' \in (t', t), (\rho, t'') \models \phi_1
\end{aligned}$$

The satisfiability problem for $TLCI_0$ is PSPACE-complete when the indices k of $C_k^{(0,b)}$ is coded in unary, and EXPSPACE-complete when the indices are coded in binary [18]. TLC is the special case where the upper bound b is 1.

Real-time logics are usually required to be scalable, in the sense that a change of the time scale (e.g. from second to minutes), or said otherwise the multiplication by a rational number, should not affect their definition. The logics presented here are not scalable, but their scalable version can be obtained by replacing the integers by rationals in the definition. This may change the expressiveness results below.

TLC was shown to be strictly more expressive than MITL with past with a simple non-scalable example [14]: It uses a single proposition p, that is true exactly at multiples of 2/3. Every *MITL* formula with past will eventually behave like $p, \neg p, true$, or $false$. In contrast, $C_2^{(0,1)}(p)$ will be true on $\{((1+2i)/3, (2+2i)/3) | i \in \mathbb{N}\}$ indefinitely. Similarly, we see that the event clock y_p for p will always be between 0 and 1, but the memory event clock y_p^2 will periodically go above 1. In general, memory clocks bring the same supplementary expressive power, as we will see in the next section.

From [8] we can show that future TLC can be translated to the scalable MTL. To the best of our knowledge, the question whether TLC is more expressive than scalable MITL with past is open.

3.2 Recursive Memory Event Clocks Temporal Logic

In this section we introduce the Recursive Memory Event Clocks Temporal Logic (RMECTL). RMECTL extends EventClockTL of [20,11]. We generalise its modalities by adding an index k: the recording modality $\lhd_I^k \phi$ means that the kth last time that ϕ was true is in interval $t - I$, and symmetrically the predicting modality $\rhd_I^k \phi$ says the kth next occurrence of ϕ will occur within I. We count only one occurrence for an interval where ϕ is continuously true. Such a modality in fact introduces a memory event clock: $\lhd_I^k \phi$ means that we reset a memory clock each time ϕ is true, and we constrain the kth clock value at the time of evaluation. We denote the temporal logic where $k \leq n$ by $RMECTL_n$, for $n \in \mathbb{N}$. If we allow only index 1, we find back EventClockTL.

Definition 1. *The formulas of Recursive Memory Event Clock Temporal Logic (RMECTL) are built from propositional symbols $\mathbb{P}$, boolean connectives, the temporal operators* until *and* since *and two symmetric real-time modalities, the recording modality and predicting modality. The formulas ϕ of RMECTL are defined by the grammar:*

$$\phi ::= \ p \mid \ \rhd_I^n \ \phi \mid \ \lhd_I^n \ \phi \mid \phi_1 \ \wedge \ \phi_2 \mid \neg\phi \mid \phi_1 \ \mathcal{U} \ \phi_2 \mid \phi_1 \ \mathcal{S} \ \phi_2$$

where p is a propositional symbol, $I \in \mathcal{I}_{\mathbb{N}}$ is an interval, and $n \in \mathbb{N}^+$. Let ϕ be a RMECTL formula and let ρ be a signal whose propositional symbols contain all propositions that occur in ϕ. The semantics of the new modalities are:

$(\rho, t) \models \lhd_I^n \phi$ *iff the set* $\{t_n \mid \exists t_1, \ldots, t_{n-1}, s_1, \ldots, s_{n-1} : t_n < s_{n-1} < t_{n-1} < \cdots < t_1 < t, \bigwedge_{i \leq n} (\rho, t_i) \models \phi, \bigwedge_{i<n} (\rho, s_i) \not\models \phi\}$ *Ends During* $t - I$

$(\rho, t) \models \rhd_I^n \phi$ *iff the set* $\{t_n \mid \exists t_1, \ldots, t_{n-1}, s_1, \ldots, s_{n-1} : t_n > s_{n-1} > t_{n-1} > \cdots > t_1 > t, \bigwedge_{i \leq n} (\rho, t_i) \models \phi, \bigwedge_{i<n} (\rho, s_i) \not\models \phi\}$ *Begins During* $t + I$

where "Begins During" and "Ends During" have been defined in Section 2.1. The intuition is that each t_i is a witness of an interval where ϕ was true, that caused a reset of the clock. They must be distinct intervals, i.e. they must be separated by an interval where ϕ is false, as witnessed by s_i. Intuitively, the nth previous reset is the maximum of the candidates t_n, but this maximum might not exist. Hence the indirect definition using "Begins During".

RMECTL turned out to be very close to TLCI_0:

Theorem 1. *RMECTL and $TLCI_0$ are intertranslatable linearly.*

Proof. **From RMECTL to TLCI_0.** We first simplify formulas of RMECTL. First note that the left bound can always be set to 0 as follows: Define the downward closure of an interval I as $\downarrow I = \{t > 0 \mid \exists t' \in I . t \leq t'\}$. We use $\rhd_I^n \phi \equiv \neg \rhd^n_{\downarrow I \setminus I} \phi \wedge \rhd^n_{\downarrow I} \phi$. For instance, $\rhd^n_{(a,b]} \phi \equiv \neg \rhd^n_{(0,a]} \phi \wedge \rhd^n_{(0,b]} \phi$. Second, if the right bound of the interval is closed, we can open it. Define $\mathcal{J}\phi$ as $\phi \mathcal{U}\ true$. Intuitively, it means that ϕ will be true for some time just after the current point of time. Its dual $\mathcal{K}^+$ [10], i.e. $\neg(\neg\phi \mathcal{U}\ true)$, means that ϕ will be true *arbitrarily close* after the current point of time. Symmetrically, $\mathcal{B}\phi$, defined as $\phi \mathcal{S}\ true$, means that ϕ was true for some time just before

now, and $\mathcal{K}^-$ [10], that ϕ will be true arbitrarily close before now. Then we use $\rhd^n_{(0,b]}\phi = \mathcal{J} \rhd^n_{(0,b)} \phi$ if ϕ is left-closed, which is expressed by $\neg\phi\mathcal{U}\phi$. This gives $\rhd^n_{(0,b]}\phi = (\neg\phi\mathcal{U}\phi \wedge \mathcal{J} \rhd^n_{(0,b)} \phi) \vee (\neg(\neg\phi\mathcal{U}\phi) \wedge \rhd^n_{(0,b)}\phi)$. We are only left with operators of the form $\rhd^n_{(0,b)}\phi$, which mean that n resets occur within interval $(0, b)$. A reset for a predicting clock is a rising edge, i.e. ϕ becomes true, and can be described by the formula: $(\mathcal{K}^-\neg\phi \wedge \phi) \vee (\neg\phi \wedge \mathcal{K}^+\phi)$, that we abbreviate $\mathcal{R}\phi$. A special case is when ϕ is true just after now $(\mathcal{K}^+\phi)$, then $\rhd^1_{(0,b]}$ is true, even without a rising edge.Thus we translate $\rhd^n_{(0,b)}\phi$ by

$$(\neg\mathcal{K}^+\phi' \wedge C^{(0,b)}_n(\mathcal{R}\phi')) \vee (\mathcal{K}^+\phi' \wedge C^{(0,b)}_{n-1}(\mathcal{R}\phi'))$$

which is a formula of TLCI_0. $\lhd^n_{(0,b)}\phi$ is translated symmetrically. All other operators appear in both logics, and are translated trivially. This translation is linear in the number of subformulas, i.e. in DAG size which is the relevant measure for logics. It preserves or decreases the indices.

From $\mathbf{TLCI_0}$ to RMECTL. Let I be $(0, b)$. $C^{(0,b)}_n(\phi)$ means that there are at least n points satisfying ϕ in $t + (0, b)$, while $\rhd^n_{(0,b)}\phi$ means that $t + (0, b)$ comprises at least n rising edges (or $n - 1$ if it begins with a ϕ). This is different as soon as ϕ is true on a non-singular ϕ-interval. But then, this interval comprises an infinite number of ϕ points, and thus makes $C^{(0,b)}_n(\phi)$ true. Otherwise, all ϕ-intervals are singular, erasing the difference. Thus we translate $C^{(0,b)}_n(\phi)$ by $(\rhd_{(0,b)}\mathcal{J}\phi') \vee (\rhd^n_{(0,b)}\phi')$. $\overleftarrow{C}^{(0,b)}_n(\phi)$ is translated symmetrically. All other operators appear in both logics, and are translated trivially. This translation is linear in the number of subformulas (DAG size).

Corollary 1. *RMECTL and $TLCI_0$ have the same expressiveness.*

Corollary 2. *RMECTL is more expressive than MITL and EventClockTL.*

Note that MITL, TLC_1, and EventClockTL have the same expressiveness in interval or signal semantics [11].

Corollary 3. *Satisfiability and validity of RMECTL is PSPACE-complete if indices are in unary, EXPSPACE-complete if indices are in binary.*

3.3 The Temporal Logic of One Clock with Right Constraint

In this section, we introduce a fragment of the Timed Propositional Temporal Logic (TPTL) [5] that is expressively equivalent to TLC, but offers a more convenient syntax. TPTL is a temporal logic based on clock variables declared by "freeze quantifiers"; these clock variables can then be used in explicit real-time constraints. It is very natural and expressive, in particular more than MTL; hence it was dubbed "a really temporal logic" by its authors. Alas, satisfiability of full TPTL in most semantics is Σ^1_1-complete, i.e. highly undecidable [5]. The example TPTL formula below, borrowed from [4]:

$$Gx.(p \rightarrow F(q \wedge F(r \wedge x < 1)))$$

where $F\phi$ is $true\,\mathcal{U}\phi$, and G its dual, expresses quite naturally that each time we have p (a request), we will have q then r (it will be processed) within 1 second. This formula spurred further research [12,18,8] to express it in more decidable logics. Here, we propose instead a decidable fragment of TPTL that contains this example. Recently, another fragment was shown decidable, but only for the point semantics [17].

We use an adapted version of TPTL, called Clock Temporal Logic [19, p.46]: We interpret it with a continuous semantics on the non-negative reals (rather than a point semantics on the natural numbers [5]); we add the past modalities; we interpret the quantifiers as clock resets (rather than as a freezed copy of absolute time [5]).

Our fragment is called the One Clock Temporal Logic with constraints on the Right, abbreviated TPTL1R to evoke its link with TPTL. In this logic, only one clock can be active at a time. Furthermore, inside the scope of a clock, the formulae must be positive and until/since can only contain a constraint in the right side. The syntax is rather natural:

$$\begin{array}{c}\phi ::= p \mid x.\phi_x \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \mathcal{U} \phi_2 \mid \phi_1 \mathcal{S} \phi_2 \\ \phi_x ::= \phi \mid x \in I \mid \phi_x \vee \phi_x \mid \phi \wedge \phi_x \mid \phi_1 \mathcal{U} \phi_x \mid \phi_1 \mathcal{S} \phi_x\end{array}$$

where ϕ_x means "a formula with clock variable x free", $p \in AP, I$ is an interval with integer bounds.

The semantics, $(\rho, t) \models \phi$, is defined as usual, plus for the reset quantifier:

$$(\rho, t) \models x.\phi_x \text{ iff } (\rho, t) \models_t \phi_x$$

For formulae ϕ_x, the real value v below is the time of the reset.

$$\begin{array}{lll}(\rho,t) \models_v & \phi & \text{iff } (\rho,t) \models \phi \\ (\rho,t) \models_v & x \in I & \text{iff } t - v \in I \\ (\rho,t) \models_v & \phi'_x \vee \phi_x & \text{iff } (\rho,t) \models_v \phi'_x \text{ or } (\rho,t) \models_v \phi_x \\ (\rho,t) \models_v & \phi' \wedge \phi_x & \text{iff } (\rho,t) \models \phi' \text{ and } (\rho,t) \models_v \phi_x \\ (\rho,t) \models_v & \phi_1 \mathcal{U} \phi_x & \text{iff } \exists t' > t.(\rho,t') \models_v \phi_x \wedge \forall t'' \in (t',t).(\rho,t'') \models \phi_1 \\ (\rho,t) \models_v & \phi_1 \mathcal{S} \phi_x & \text{iff } \exists t' < t.(\rho,t') \models_v \phi_x \wedge \forall t'' \in (t',t), (\rho,t'') \models \phi_1\end{array}$$

In the proof below, we use the fact that Q2MLO (called L_2 in [12]) and $TLCI_0$ have the same expressiveness [13]. Let us recall that Q2MLO is a first-order monadic logic of order. It contains first-order logic, plus a metric quantifier:

$$\phi ::= \forall t.\ \phi_1 \mid \neg\phi_1 \mid \phi_1 \wedge \phi_2 \mid t_1 = t_2 \mid t_1 < t_2 \mid \exists t \in t_0 + I.\ \phi(t, t_0)$$

where I is a non-singular interval with integer bounds. Only two free variables t, t_0 are allowed in the quantified formula $\phi(t, t_0)$.

Theorem 2. *TPTL1R is as expressive as TLC, $TLCI_0$, and Q2MLO.*

Proof. It suffices to translate $TLCI_0$ to TPTL1R, and then TPTL1R to Q2MLO.

1. TPTL1R can express $C_n^{(0,b)}$:

$$C_n^{(0,b)}(\phi) = x.(F(\phi \wedge F(\phi \wedge \ldots F(\phi \wedge x < b))))$$

   And symmetrically for the past operator.
2. The semantics of TPTL1R translates any ϕ_x appearing in $x.\phi_x$ into a first-order formula $\phi_x(v)$. We note that: (i) disjunctions are in the scope of existential quantifiers only, so that we can move the disjunctions out; (ii) each constraint $t_k - v \in I$ is also in the scope of existential quantifiers only, and in particular $\exists t_k$. Therefore the order of these existential quantifications is irrelevant, and we move in front $\exists t_k$ (the quantification whose variable appears in the constraint) then the other variables. The constraints are now part of a conjunction. We move each constraint just after its quantification. We obtain a formula of the form $\exists t_k.t_k - v \in I \wedge \phi(v, t_k)$, that we can express in Q2MLO as $\exists t_k \in v + I.\phi(v, t_k)$.

4 Recursive Memory Event Clocks Automata

As explained in the introduction, our goal is to extend in the realm of real-time the success of classical automata, that can express both specifications and programs uniformly, and are thus the internal data structure used by most model-checkers. Automata can deal with real-time by adding clocks that can be reset or tested. Timed automata allow liberal use of their clocks, making their inclusion problem undecidable. A more disciplined use of clocks is needed. Our proposal follows the idea of ECA. Since ECA reset clocks only on the occurrence of their atomic proposition, which is not very expressive, we proposed Recursive Event Clock Automata (RECA) [11]. "Recursive" refers to the fact that the resets of each clock of an automaton are controlled by a lower-level automaton. When this automaton visits a monitored location, it resets the associated clocks: An event-recording clock x_A and an event-predicting clock y_A can be associated with each monitored automaton. Thus no automaton can reset its own clocks. In particular, an automaton of level 0 has no subautomata, hence no clock.

Here, we examine whether introducing Memory Event Clocks (MEC) will further increase their power. This leads to Recursive Memory Event Clock Automata (RMECA). Event-recording memory clocks $x_{\mathcal{A}}^i$ record the time that has expired since the i_{th} last time at which the automaton $\mathcal{A}$ could pass through a monitored location, and the event-predicting memory clock $y_{\mathcal{A}}^i$ always records the amount of time that will expire until the i_{th} next time at which the automaton $\mathcal{A}$ could pass through a monitored location. Equivalently, a reset does not destroy the previous values of a memory clock: instead, a new clock with value 0 is created, and earlier clocks are still accessible by the indexed notation. We have already used MEC to define RMECTL in Section 3.2.

MEC are determined by the ISS (and not by the run as for TA). To deal with MEC, it is easier to consider them as supplementary propositions. We have then to make them mutually exclusive, so we consider atomic constraints. For a given clock c, let $R_c = \{r_1, \ldots, r_n\}$ be the constants to which it is compared in $\mathcal{A}$, in increasing

order. The atomic constraints for c are $c = \bot, c < r_1, c = r_i, c \in (r_i, r_{i+1}), c > r_n$. An atomic constraint for $\mathcal{A}$ is a conjunction of atomic constraints for each clock. No clock valuation can satisfy two different atomic constraints, and each constraint of $\mathcal{A}$ can be expressed as a disjunction of atomic constraints. This latter property allows us to use only atomic constraints, using more locations if needed.

We now examine the complexities due to continuous time. We want these automata to consider ISS that define the same signal as equivalent, even if the intervals might be split differently. This goal will force the definition of deterministic automata below. To simplify it, we label our locations not only with atomic propositions but also with "limits". Intuitively, a past (resp. future) limit describes what happened just before (resp. just after) the current time. Locations where some limit is different from the current label are called *singular*: only a single instant can be spent there. From a singular location, we can only make a transition to a non-singular location, where the labelling must be as predicted by the limit of the singular location. The past limit of $\top$ is false only in initial locations.

Given a set propositions $\mathbb{P}$, the limit closure ($Limit(\mathbb{P})$) is the set $\{p, \overrightarrow{p}, \overleftarrow{p} \mid p \in \mathbb{P} \cup \{\top\}\}$. $\overrightarrow{p}$ is the future limit of p and $\overleftarrow{p}$ is the past limit of p.

Definition 2. *A Recursive Memory Event Clock Automaton (RMECA) is a tuple $\mathcal{A} = (\mathbb{P}, S, S_0, \rightarrow, C, \gamma, M, F)$, such that:*

1. *$\mathbb{P}$ is a set finite set of propositional symbols.*
2. *S is a finite set of locations and $S_0 \subseteq S$ is the set of starting locations.*
3. *$\rightarrow \subseteq S \times S$ are the transitions.*
4. *A finite set of atomic constraints C, containing clocks x^i_B or y^i_B, with B a lower-level RMECA.*
5. *$\gamma : S \rightarrow 2^{Limit(\mathbb{P} \cup C)}$ is a function which labels each location $s \in S$ with the set of limits of propositions and constraints that are true in that location.*
6. *$M \subseteq S$ is the set of monitored locations: when the automaton visits such a location, it resets the associated clock.*
7. *$F \subseteq S$ is a set of Büchi accepting locations.*

We now define when a RMECA accepts an ISS ρ, thanks to a timed run. This is the time t when the automaton can visit a monitored location.

Definition 3. *A RMECA $\mathcal{A}$ accepts a signal ρ at time t, if there exist an infinite timed run $\theta = (s, I)$ such that following conditions holds:*

1. *the run starts in a starting location $s_0 \in S_0$.*
2. *for all $i > 0$, the run either follows a transition: $s_{i-1} \rightarrow s_i$ or stutters: $s_{i-1} = s_i$.*
3. *It is in a monitored location at time t: $\theta(t) \in M$.*
4. *The labelling of the location corresponds to the ISS and satisfies the limits and clock constraints: $\forall t' \in \mathbb{R}^+, (\rho, t) \models \gamma(\theta(t))$.*
5. *It visits infinitely often a Büchi location.*

The clock valuation function over a lower-level RMECA at $\mathcal{A}$ and time t at ρ, is noted $\nu^\rho_t : \mathcal{C}_\mathcal{A} \rightarrow \mathbb{R}^+_\bot$. It assigns a (non-standard) positive real, or undefined, to each clock variable. The resets are done when $\mathcal{A}$ can visit a monitored location.

Given t and ρ, the reset interval of $y^n_{\mathcal{A}}$ is the interval I_n such that there are non-empty intervals $I_1, \cdots, I_{n-1}$ where:

1. $t < I_1 < \cdots < I_n$,
2. $\forall i \leq n, \forall r \in I_i, \mathcal{A}$ accepts ρ in r
3. $\forall i < n, BetwI(I_i, I_{i+1})$ is not empty and $\forall r \in BetwI(I_i, I_{i+1}), \mathcal{A}$ does not accept ρ in r
4. $\forall r \in BetwI(\{t\}, I_1), \mathcal{A}$ does not accept ρ in r.

Note that I_1 might begin just after t, in which case the last condition is vacuously true. The reset interval for a recording clock is symmetric. The value of a clock is:

$$\nu^\rho_t(x^n_{\mathcal{A}}) = \begin{cases} t - r & \text{if the reset interval of } x^n_{\mathcal{A}} \text{ is } (l, r] \text{ or } [l, r] \\ (t - r)^+ & \text{if the reset interval of } x^n_{\mathcal{A}} \text{ is } (l, r) \text{ or } [l, r) \\ \perp & \text{if } x^n_{\mathcal{A}} \text{ has no reset interval} \end{cases}$$

Symmetrically,

$$\nu^\rho_t(y^n_{\mathcal{A}}) = \begin{cases} l - t & \text{if the reset interval of } y^n_{\mathcal{A}} \text{ is } [l, r) \text{ or } [l, r] \\ (l - t)^+ & \text{if the reset interval of } y^n_{\mathcal{A}} \text{ is } (l, r) \text{ or } (l, r] \\ \perp & \text{if } y^n_{\mathcal{A}} \text{ has no reset interval} \end{cases}$$

The logic RMECTL and RMECA in fact use the same memory clocks:

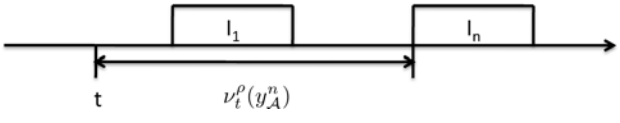


Fig. 1. The value of $y^n_{\mathcal{A}}$ and its reset interval I_n

Theorem 3. *$\nu^\rho_t(x^n_{\mathcal{A}}) \in I$ iff $(\rho, t) \models \lhd^n_I p$*
where p is a proposition such that $(\rho, t) \models p$ iff $\mathcal{A}$ accepts ρ at time t.

4.1 Properties of *RMECA*

We now show here that RMECA inherit all good properties of RECA and ECA: they are determinizable and closed over all boolean operations. The proofs are the same as for RECA [19], replacing clocks by memory clocks. In view of the fact that we later show that they are expressively equivalent, this seems obvious, but (i) we need those properties to prove the equivalence, and (ii) the direct proofs give algorithms that are more efficient, since the translation of the next section is exponential.

We first adapt the definition of determinism to cater for continuous time [19]:

Definition 4. *A RMECA $\mathcal{A} = (\mathbb{P}, S, S_0, \rightarrow, C, \gamma, M, F)$, is deterministic iff:*

1. *Distinct initial locations $s_1 \neq s_2 \in S_0$ have distinct labellings: $\gamma(s_1) \neq \gamma(s_2)$*
2. *Successive locations $s_1 \rightarrow s_2$ have distinct labellings.*
3. *Distinct successor locations $s_2 \neq s_3, s_1 \rightarrow s_2, s_1 \rightarrow s_3$ have distinct labellings.*

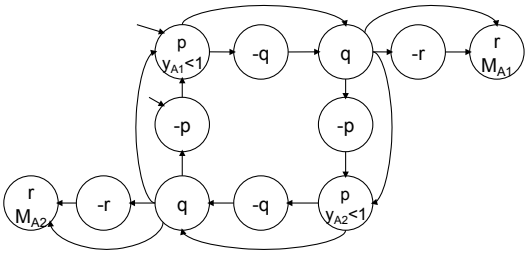


Fig. 2. A RECA for $Gx.(p \rightarrow F(q \wedge F(r \wedge x < 1)))$

The determinism ensures that, at each time t during a run, the choice of the next location is uniquely determined by the current location of the automaton and (ρ, t). Condition (2) ensures that the time at which to leave a location is uniquely given by the signal ρ. Therefore there is at most one (signal) run for each ρ.

Theorem 4. *For any RMECA $\mathcal{A}$, we construct a deterministic Rabin RMECA $\mathcal{A}'$ that accepts the same language. If $\mathcal{A}$ has n locations, $\mathcal{A}'$ has $2^{O(nlogn)}$ locations and the same clocks and subautomata.*

Note that the time of acceptance also preserved by determinization.

Theorem 5. *The class of RMECA-recognizable timed languages is closed under union, intersection and complementation.*

Theorem 6. *The emptiness, universality, and language inclusion problems for RMECA are EXPSPACE-Complete.*

This differs from RECA and ECA, where those problems are PSPACE-complete. The higher complexity is only due to the indices n of the clocks, that can be expressed compactly (in $\log n$) in binary notation, while their implementation requires n clocks. If the indices are expressed in unary notation, these problems are PSPACE-complete.

4.2 From RMECA to RECA

Memory clocks allow to measure distances from several resets. But do they really increase the expressiveness when placed inside automata? The answer is positive for ECA, but negative for RECA and TA. Intuitively, automata can already count resets (though less concisely). For an example, consider again the formula [4]:

$$Gx.(p \rightarrow F(q \wedge F(r \wedge x < 1)))$$

Assuming that p and q do not occur together on a non-singular interval (which is always the case in point semantics), this formula can be translated to the RECA of Fig. 2, that counts modulo 2. To save space, we have drawn the main automaton and its two subautomata $A1, A2$ with the same transitions. All states

are accepting. $A1$ is a copy of the figure where clocks constraints are removed, and the location marked M_{A1} is the only monitored location, and similarly for $A2$. The main automaton has the clock constraints, but no monitored locations. Although the formula conceptually starts an infinite number of clocks x, this automaton shows that two clocks suffice.

More generally, we can use counting to eliminate memory event clocks:

Theorem 7. *For any RMECA, we can construct a RECA that accepts the same language.*

Proof. Let $\mathcal{A}$ be the initial RMECA. To simulate a memory clock $x^n_{\mathcal{C}}$ where $\mathcal{C}$ is a subautomaton of $\mathcal{A}$, we must keep track of the last n events and thus use n clocks. The clocks must be used in a rotating manner, since we cannot copy them. We can assume that $\mathcal{C}$ is deterministic, since RMECA are determinizable. To reset them, we augment $\mathcal{C}$ to count modulo n, and make n copies $\mathcal{C}'(i)$ with monitored states that differ by the count only.

Let $\mathcal{C}'(i)$ (with $i < n$) be the automaton:

1. Its symbols are the same: $\mathbb{P}' = \mathbb{P}$
2. Its states $S' = S \times [0..n)$ are pairs (s, j) where s is a state of $\mathcal{C}$, and $j < n$ counts the number of times we entered a monitored region, modulo n;
3. $S'_0 = S_0 \times \{0\}$;
4. The copy $\mathcal{C}'(i)$ monitors states of count i: $S'_m = S_m \times \{i\}$;
5. The transitions update the count when entering a monitored region: $(s, j) \to (s', j')$ iff $s \to s', j = j', (s \in S_m$ if $s' \in S_m)$ or $s \to s', j' = j + 1 \bmod n, s \notin S_m, s' \in S_m$;
6. The labellings are unchanged: $\gamma'((s, j)) = \gamma(s)$;
7. The accepting states are unchanged: $F' = F \times \{0..n-1\}$.

Each $\mathcal{C}'(i)$ is bisimilar with $\mathcal{C}$. $\mathcal{C}$ visits a monitored state iff (exactly) one of the $\mathcal{C}'(i)$ does. Therefore the set of values of the clocks is the same:

$$\{\nu^\rho_t(x_{\mathcal{C}'(i)}) \mid i < n\} = \{\nu^\rho_t(x^j_{\mathcal{C}}) \mid 0 < j \leq n\}$$

Now we translate the constraints that appear in the upper level automaton: the value of $x^n_{\mathcal{C}}$ is the maximal value of this set. Thus a constraint $x^n_{\mathcal{C}} < c$ is translated by $\bigwedge_{i<n} x_{\mathcal{C}'(i)} < c$ (idem for $\leq$), symmetrically $x^n_{\mathcal{C}} > c$ by $\bigvee_{i<n} x_{\mathcal{C}'(i)} > c$ (idem for $\geq$), and $x^n_{\mathcal{C}} = \bot$ by $\bigvee_{i<n} x_{\mathcal{C}'(i)} = \bot$. $x = c$ is considered as an abbreviation for $x \geq c \wedge x \leq c$.

4.3 From RMECTL to RMECA

We briefly present the construction of a RMECA from a RMECTL formula.

Theorem 8. *For every RMECTL formula ϕ, we can construct a RMECA $\mathcal{A}_\phi$ that accepts ρ at time t iff $(\rho, t) \models \phi$.*

The construction is as in [19], with memory clocks instead of clocks. The transformation is done level by level, where the level of a formula is the nesting depth of real-time modalities. A formula $\rhd^n_I \phi$ is translated as constraint $x^n_{\mathcal{A}_\phi} \in I$. The

formula ϕ is recursively transformed in a tableau automaton for continuous time, where the monitored states are the states containing ϕ. The construction is exponential in the size of the non-real time part of the formula, but linear in the real-time part.

5 Conclusions

Recursive event clocks have been criticized to be too weak as they only record the time to next/previous event. In this paper, we have introduced memory event clocks that are designed to overcome this limitation. We presented them in the interval-based semantics, both to ease comparison with related work and since this setting is more general and more difficult than point semantics.

When we introduce such clocks in a temporal logic, we obtain RMECTL. RMECTL allows punctuality constraints and allows constraints on the n_{th} next (n_{th} last) time a formula will be (was) true. Still, we have shown that RMECTL has the same expressiveness and complexity as TLCI_0.

We also identified a fragment of TPTL that is expressively equivalent and decidable. We conjecture that a larger fragment of TPTL is decidable.

The operational nature of these clocks blends nicely with automata, giving RMECA. They keep all the nice properties of the original event clock automata. They are as expressive as our RECA [11], showing that TLC was already included in RECA, under finite variability. The increase of expressive power is thus modest enough to disappear at automata level. In other words, the criticism was not founded with respect to RECA.

Automata are known to be equivalent to second-order quantification, and this opens the corresponding logical question, whether Q-MITL and Q-TLC, (i.e. with second-order quantification that does not to cross scope with real-time operators) are equivalent. Our results settles this question only under finite variability. Another open question is to characterize the strongest first-order real-time temporal logic included in RECA, beyond TLC perhaps. It is also open, whether TLC is more expressive than scalable MITL with past.

Acknowledgements. This work was funded by the Interuniversity Attraction Poles Programme (IAP) of the Belgian State, Belgian Science Policy (MoVES project), by the Belgian Science Foundation (FNRS) under FRFC project CFV, and by the European Science Foundation (ESF) under EUROCORES project LogiCCC/GASICS.

References

1. Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. 126(2), 183–235 (1994)
2. Alur, R., Feder, T., Henzinger, T.A.: The benefits of relaxing punctuality. J. ACM 43(1), 116–146 (1996)
3. Alur, R., Fix, L., Henzinger, T.A.: A determinizable class of timed automata. In: Dill, D.L. (ed.) CAV 1994. LNCS, vol. 818, pp. 1–13. Springer, Heidelberg (1994)

4. Alur, R., Henzinger, T.A.: Logics and models of real time: A survey. In: Huizing, C., de Bakker, J.W., Rozenberg, G., de Roever, W.-P. (eds.) REX 1991. LNCS, vol. 600, pp. 74–106. Springer, Heidelberg (1992)
5. Alur, R., Henzinger, T.A.: Real-time logics: Complexity and expressiveness. Inf. Comput. 104(1), 35–77 (1994)
6. Annichini, A., Bouajjani, A., Sighireanu, M.: Trex: A tool for reachability analysis of complex systems. In: Berry, G., Comon, H., Finkel, A. (eds.) CAV 2001. LNCS, vol. 2102, pp. 368–372. Springer, Heidelberg (2001)
7. Baier, C., Bertrand, N., Bouyer, P., Brihaye, T.: When are timed automata determinizable? In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009. LNCS, vol. 5556, pp. 43–54. Springer, Heidelberg (2009)
8. Bouyer, P., Chevalier, F., Markey, N.: On the expressiveness of TPTL and MTL. In: Sarukkai, S., Sen, S. (eds.) FSTTCS 2005. LNCS, vol. 3821, pp. 432–443. Springer, Heidelberg (2005)
9. Bozga, M., Daws, C., Maler, O., Olivero, A., Tripakis, S., Yovine, S.: Kronos: A model-checking tool for real-time systems. In: Vardi, M.Y. (ed.) CAV 1998. LNCS, vol. 1427, pp. 546–550. Springer, Heidelberg (1998)
10. Gabbay, D.M., Hodkinson, I.M.: An axiomatization of the temporal logic with until and since over the real numbers. J. Log. Comput. 1(2), 229–259 (1990)
11. Henzinger, T.A., Raskin, J.-F., Schobbens, P.-Y.: The regular real-time languages. In: Larsen, K.G., Skyum, S., Winskel, G. (eds.) ICALP 1998. LNCS, vol. 1443, pp. 580–591. Springer, Heidelberg (1998)
12. Hirshfeld, Y., Rabinovich, A.M.: A framework for decidable metrical logics. In: Wiedermann, J., Van Emde Boas, P., Nielsen, M. (eds.) ICALP 1999. LNCS, vol. 1644, pp. 422–432. Springer, Heidelberg (1999)
13. Hirshfeld, Y., Rabinovich, A.M.: An expressive temporal logic for real time. In: Královič, R., Urzyczyn, P. (eds.) MFCS 2006. LNCS, vol. 4162, pp. 492–504. Springer, Heidelberg (2006)
14. Hirshfeld, Y., Rabinovich, A.M.: Expressiveness of metric modalities for continuous time. In: Grigoriev, D., Harrison, J., Hirsch, E.A. (eds.) CSR 2006. LNCS, vol. 3967, pp. 211–220. Springer, Heidelberg (2006)
15. Ouaknine, J., Worrell, J.: On the language inclusion problem for timed automata: Closing a decidability gap. In: LICS, pp. 54–63. IEEE Computer Society, Los Alamitos (2004)
16. Ouaknine, J., Worrell, J.: Some recent results in metric temporal logic. In: Cassez, F., Jard, C. (eds.) FORMATS 2008. LNCS, vol. 5215, pp. 1–13. Springer, Heidelberg (2008)
17. Parys, P., Walukiewicz, I.: Weak alternating timed automata. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009. LNCS, vol. 5556, pp. 273–284. Springer, Heidelberg (2009)
18. Rabinovich, A.: Complexity of metric temporal logics with counting and the Pnueli modalities. In: Cassez, F., Jard, C. (eds.) FORMATS 2008. LNCS, vol. 5215, pp. 93–108. Springer, Heidelberg (2008)
19. Raskin, J.-F.: Logics, Automata and Classical Theories for Deciding Real Time. Phd thesis, FUNDP University, Belgium (1999)
20. Raskin, J.-F., Schobbens, P.-Y.: State clock logic: A decidable real-time logic. In: Maler, O. (ed.) HART 1997. LNCS, vol. 1201, pp. 33–47. Springer, Heidelberg (1997)
21. The UPPAAL tool, `http://www.uppaal.com/`

Simulation and Bisimulation for Probabilistic Timed Automata*

Jeremy Sproston and Angelo Troina

Dipartimento di Informatica, Università di Torino, 10149 Torino, Italy
{sproston,troina}@di.unito.it

Abstract. Probabilistic timed automata are an extension of timed automata with discrete probability distributions. Simulation and bisimulation relations are widely-studied in the context of the analysis of system models, with applications in the stepwise development of systems and in model reduction. In this paper, we study probabilistic timed simulation and bisimulation relations for probabilistic timed automata. We present an EXPTIME algorithm for deciding whether two probabilistic timed automata are probabilistically timed similar or bisimilar. Furthermore, we consider a logical characterization of probabilistic timed bisimulation.

1 Introduction

The increasing complexity of embedded and networked technologies has lead to a growing demand for formal techniques to reason about their safety, reliability and efficiency. In particular, formal modelling languages for describing systems have been developed, together with associated automatic verification techniques. We consider the case of real-time systems, in which timing information is associated with system behaviour, which can be reflected in system choices (for example, the system times-out if a response has not been received within 30ns) and in measures such as timeliness and efficiency (for example, a system is regarded as being timely if a leader is elected within 1s after a new node joins the network). A widespread example of a system description formalism for real-time systems is timed automata [1]. We also consider probabilistic systems, in which system behaviour is associated with a quantity representing its relative likelihood (for example, a message is lost with probability 0.01). When modelling probabilistic systems, it is often convenient, for representing interleaving between parallel components or for abstraction, to consider formalisms which include both nondeterministic and probabilistic choice, such as those based on Markov decision processes [2] or Segala's probabilistic automata [3,4]. In certain cases, our aim is to model probabilistic real-time systems, for which it is important to model both timed and probabilistic behaviour within the same system model. An example of a formalism for such systems, based on a combination of timed automata

* Supported in part by the MIUR-PRIN project PaCo - Performability-Aware Computing: Logics, Models and Languages.

K. Chatterjee and T.A. Henzinger (Eds.): FORMATS 2010, LNCS 6246, pp. 213–227, 2010.

and Segala's probabilistic automata, is probabilistic timed automata [5]. Probabilistic timed automata have been used previously to model systems such as the IEEE 1394 root contention protocol, the backoff procedure in IEEE 802.11 Wireless LANs, and the IPv4 Zeroconf protocol [6,7].

In the field of formal modelling of systems, reasoning about the same system at different levels of detail using the notions of *refinement* and *abstraction* is well-established. Both notions involve the use of a relation between two system models $\mathcal{A}$ and $\mathcal{B}$: the relation establishes that $\mathcal{A}$ refines $\mathcal{B}$, or, equivalently, that $\mathcal{B}$ is an abstraction of $\mathcal{A}$. These notions can be used in two different ways. Firstly, they offer mechanisms for the stepwise development of system models. That is, the system modeller starts from an abstract description of the system, then refines successively this description to obtain a detailed system model. Secondly, abstraction can be used in the context of system analysis: a system model may be too large to allow its analysis within the resources available, and therefore a smaller model which abstracts the original one can be constructed and analyzed.

An example of a relation for refinement and abstraction of system models is *simulation* [8]. This relation is defined on the states of the two models $\mathcal{A}$ and $\mathcal{B}$. If state $s_{\mathcal{B}}$ of $\mathcal{B}$ simulates state $s_{\mathcal{A}}$ of $\mathcal{A}$, then any single transition from $s_{\mathcal{A}}$ can be mimicked from $s_{\mathcal{B}}$, and the states reached by these transitions are also in the simulation relation. If the converse also holds (that is, also any single transition from $s_{\mathcal{B}}$ can be mimicked from $s_{\mathcal{A}}$), then the relation is a *bisimulation* [9,10]. Simulation and bisimulation relations have been considered for real-time systems: in this paper we consider *timed* (rather than time-abstract) versions of these relations. Deciding timed simulation and bisimulation for timed systems is in EXPTIME [11,12]. Similarly, deciding timed alternating (bi)simulation for timed games, which can be used to model real-time controller synthesis problems, is also in EXPTIME [13]. Simulation and bisimulation have also been considered for Markov decision processes or probabilistic automata models: deciding simulation and bisimulation can be done in polynomial time [14,15]. The relations can also be accompanied by a logical characterization: in the case of bisimulation, this concerns in identifying a logic such that, whenever two states satisfy the same formulas of the logic, then the two states are bisimilar. The logical characterization of timed bisimulation for a subclass of timed systems has been considered in [16], whereas the logical characterization of timed alternating simulation for timed games has been presented in [13] (this result also provides a logical characterization for simulation and bisimulation for timed automata). In both cases a timed modal logic, based on Hennessy-Milner logic [17], or on temporal logic without the until operator, is considered. Instead, for probabilistic automata, a logical characterization of bisimulation has been presented in terms of a probabilistic extension of Hennessy-Milner logic [18].

In this paper we consider timed simulation and bisimulation relations for probabilistic timed automata, both in terms of algorithms for deciding such relations and in terms of a logical characterization of bisimulation. Such timed simulation and bisimulation relations for Segala's probabilistic automata enriched with timing durations have been presented in [4]. Given that probabilistic timed

automata are a generalization of both timed automata and Segala's probabilistic automata, our algorithm is inspired by [12,13] for timing aspects, and by [14,15] for probabilistic aspects. More precisely, a variant of the classical region graph is constructed from two probabilistic timed automata, on which an operator, which can determine whether sets of states are related using certain sub-procedures taken from [14,15], is iterated. We show that, as for timed automata, deciding whether two probabilistic timed automata are related by (bi)simulation is EXPTIME-complete. The logical characterization that we present considers a logic in which the classical diamond operator is replaced with a diamond operator with a time constraint, as in [13], and which features probability thresholds, as in [18]. We also treat *probabilistic* timed (bi)simulation relations in the sense of [3,4], which consider convex combinations of identically labelled transitions in order to represent randomized choice between nondeterministic alternatives.

We briefly discuss related work. Jensen and Gregersen [19,20] presented a model similar to probabilistic timed automata, but which cannot have nondeterministic choice between transitions labelled with the same action. They considered a logical characterization of timed bisimulation for their formalism, and showed that timed bisimulation between acyclic versions of their models is decidable. Yamane [21] studied timed simulation on probabilistic timed automata. However, although introducing a region-graph construction, the possibility of obtaining an algorithm was mentioned only briefly. In particular, the key concept of a finite sampling of timing durations [11,12] was missing, and the definition of how to relate probability distributions at the region-graph level was incomplete. Instead we provide a detailed description of an algorithm. Furthermore, we also establish that our algorithm matches the known lower bound, and consider also probabilistic timed (bi)simulation. Time-abstract bisimulation for probabilistic timed automata was considered in [22].

2 Probabilistic Timed Automata

Notation. We use $\mathbb{R}_{\geq 0}$ to denote the set of non-negative real numbers and $\mathbb{N}$ to denote the set of natural numbers. A discrete probability *distribution* over a countable set Q is a function $\mu : Q \to [0,1]$ such that $\sum_{q \in Q} \mu(q) = 1$. For a function $\mu : Q \to \mathbb{R}_{\geq 0}$ we define $\mathsf{support}(\mu) = \{q \in Q \mid \mu(q) > 0\}$. Then for an uncountable set Q we define $\mathsf{Dist}(Q)$ to be the set of functions $\mu : Q \to [0,1]$, such that $\mathsf{support}(\mu)$ is a countable set and μ restricted to $\mathsf{support}(\mu)$ is a distribution. For $q \in Q$, let $\{q \mapsto 1\}$ be the *point distribution at* q which assigns probability 1 to q. Let $\{\mu_1, ..., \mu_k\}$ be a finite set of distributions over Q, and let $c_1, ..., c_k$ be a sequence of real numbers in $[0,1]$ such that $\sum_{1 \leq i \leq k} c_i = 1$. Then the *convex combination* $\sum_{1 \leq i \leq k} c_i \mu_i$ is the distribution μ defined by $\mu(q) = \sum_{1 \leq i \leq k} c_i \mu_i(q)$ for each $q \in Q$.

Probabilistic Timed Labelled Transition Systems. A *probabilistic timed labelled transition system* (PTLTS) $\mathsf{P} = (S, \bar{S}, Act, \to)$ comprises the following components:

- A possibly uncountable set of *states* S with initial states $\bar{S} \subseteq S$.
- A finite set Act of *actions*.
- A possibly uncountable *timed, probabilistic, nondeterministic transition relation* $\rightarrow \subseteq S \times \mathbb{R}_{\geq 0} \times Act \times \mathsf{Dist}(S)$.

The transitions from state to state of a PTLTS are performed in two steps: given that the current state is s, the first step concerns a nondeterministic selection of $(s, d, a, \mu) \in \rightarrow$, where d and a are the duration and the action of the transition, respectively; the second step comprises a probabilistic choice, made according to the distribution μ, as to which state to make the transition to (that is, we make a transition to a state $s' \in S$ with probability $\mu(s')$).

Syntax of Probabilistic Timed Automata. Let $\mathcal{X}$ be a finite set of real-valued variables called *clocks*, the values of which increase at the same rate as real-time. The set $CC(\mathcal{X})$ of *clock constraints* over $\mathcal{X}$ is defined as the set of conjunctions over atomic formulas of the form $x \sim c$, where $x, y \in \mathcal{X}$, $\sim \in \{<, \leq, >, \geq, =\}$, and $c \in \mathbb{N}$.

A *probabilistic timed automaton* (PTA) $\mathcal{A} = (L, \bar{L}, Act, \mathcal{X}, inv, prob)$ is a tuple consisting of the following components:

- A finite set L of *locations* with the initial locations $\bar{L} \subseteq L$.
- A finite set $\mathcal{X}$ of clocks.
- A finite set Act of actions.
- A function $inv : L \rightarrow CC(\mathcal{X})$ associating an *invariant condition* with each location.
- A finite set $prob \subseteq L \times CC(\mathcal{X}) \times Act \times \mathsf{Dist}(2^{\mathcal{X}} \times L)$ of *probabilistic edges*.

A probabilistic edge $(l, g, a, \mathsf{p}) \in prob$ is a tuple containing (1) a source location l, (2) a clock constraint g, called a *guard*, (3) an action a, and (4) a probability distribution p which assigns probability to pairs of the form (X, l'), where X is a set of clocks to be reset and l' is a location. The behaviour of a PTA takes a similar form to that of a timed automaton [1]: in any location time can advance as long as the invariant holds, and a probabilistic edge can be taken if its guard is satisfied by the current values of the clocks. PTA generalize timed automata in the sense that, once a probabilistic edge is nondeterministically selected, then the choice of which clocks to reset and which target location to make the transition to is *probabilistic*.

The size $|\mathcal{A}|$ of the PTA $\mathcal{A}$ is $|L|+|\mathcal{X}|+|inv|+|prob|$, where $|inv|$ represents the size of the binary encoding of the constants used in the invariant condition, and $|prob|$ includes the size of the binary encoding of the constants used in guards and the probabilities used in probabilistic edges (probabilities are expressed as a ratio between two natural numbers, each written in binary).

Semantics of Probabilistic Timed Automata. We refer to a mapping $v : \mathcal{X} \rightarrow \mathbb{R}_{\geq 0}$ as a *clock valuation*. Let $\mathbb{R}^{\mathcal{X}}_{\geq 0}$ denote the set of clock valuations. Let $\mathbf{0} \in \mathbb{R}^{\mathcal{X}}_{\geq 0}$ be the clock valuation which assigns 0 to all clocks in $\mathcal{X}$. For a clock valuation $v \in \mathbb{R}^{\mathcal{X}}_{\geq 0}$ and a value $d \in \mathbb{R}_{\geq 0}$, we use $v+d$ to denote the clock valuation

obtained by letting $(v+d)(x) = v(x)+d$ for all clocks $x \in \mathcal{X}$. For a clock set $X \subseteq \mathcal{X}$, we let $v[X := 0]$ be the clock valuation obtained from v by resetting all clocks in X to 0; formally, we let $v[X := 0](x) = 0$ for all $x \in X$, and let $v[X := 0](x) = v(x)$ for all $x \in \mathcal{X} \setminus X$. The clock valuation v *satisfies* the clock constraint $\varphi \in CC(\mathcal{X})$, written $v \models \varphi$, if and only if φ resolves to true after substituting each clock $x \in \mathcal{X}$ with the corresponding clock value $v(x)$.

The semantics of the PTA $\mathcal{A} = (L, \bar{L}, Act, \mathcal{X}, inv, prob)$ is the PTLTS $[\![\mathcal{A}]\!] = (S, \bar{S}, Act, \rightarrow)$ where:

- $S = \{(l, v) \mid l \in L \text{ and } v \in \mathbb{R}^{\mathcal{X}}_{\geq 0} \text{ s.t. } v \models inv(l)\}$ and $\bar{S} = \{(l, \mathbf{0}) \mid l \in \bar{L}\}$;
- $\rightarrow$ is the smallest set such that $((l, v), d, a, \mu) \in \rightarrow$ if there exist $d \in \mathbb{R}_{\geq 0}$ and a probabilistic edge $(l, g, a, \mathsf{p}) \in prob$ such that:
  1. $v + d \models g$, and $v + d' \models inv(l)$ for all $0 \leq d' \leq d$;
  2. for any $(X, l') \in 2^{\mathcal{X}} \times L$, if $\mathsf{p}(X, l') > 0$ then $(v + d)[X := 0] \models inv(l')$;
  3. for any $(l', v') \in S$, we have that $\mu(l', v') = \sum_{X \in \mathsf{Reset}(v,d,v')} \mathsf{p}(X, l')$, where $\mathsf{Reset}(v, d, v') = \{X \subseteq \mathcal{X} \mid (v + d)[X := 0] = v'\}$.

Given the state (l, v) and the duration $d \in \mathbb{R}_{\geq 0}$ such that $v + d' \models inv(l)$ for all $0 \leq d' \leq d$, in the sequel we often write $(l, v) + d$ to denote the state $(l, v + d)$. By abuse of notation, we also write $((l, v), d, a, \mathsf{p}) \in \rightarrow$ to denote the existence of $((l, v), d, a, \mu) \in \rightarrow$ such that a probabilistic edge $(l, \_, \_, \mathsf{p}) \in prob$ is used to define $((l, v), d, a, \mu)$ according to the second point in the definition of the semantic PTLTS of the PTA.

Composition of Probabilistic Timed Automata. To aid higher-level modelling, it is often useful to define complex systems as the *parallel composition* of a number of interacting sub-components. The definition of the parallel composition operator $\|$ of PTA uses ideas from the theory of (untimed) probabilistic automata [3] and classical timed automata [1], and was presented in [6]. Let $\mathcal{A}_i = (L_i, \bar{L}_i, Act_i, \mathcal{X}_i, inv_i, prob_i)$ for $i \in \{1, 2\}$ and assume that $\mathcal{X}_1 \cap \mathcal{X}_2 = \emptyset$. Given $\mathsf{p}_1 \in \mathsf{Dist}(2^{\mathcal{X}_1} \times L_1)$ and $\mathsf{p}_2 \in \mathsf{Dist}(2^{\mathcal{X}_2} \times L_2)$, we define the distribution $\mathsf{p}_1 \otimes \mathsf{p}_2 \in \mathsf{Dist}(2^{\mathcal{X}_1 \cup \mathcal{X}_2} \times (L_1 \times L_2))$ in the following way: for each $X_1 \subseteq \mathcal{X}_1$, $X_2 \subseteq \mathcal{X}_2$, $l_1 \in L_1$ and $l_2 \in L_2$, let $\mathsf{p}_1 \otimes \mathsf{p}_2(X_1 \cup X_2, (l_1, l_2)) = \mathsf{p}_1(X_1, l_1) \cdot \mathsf{p}_2(X_2, l_2)$. The *parallel composition* of two PTA $\mathcal{A}_1$ and $\mathcal{A}_2$ is the PTA

$$\mathcal{A}_1 \| \mathcal{A}_2 = (L_1 \times L_2, \bar{L}_1 \times \bar{L}_2, Act_1 \cup Act_2, \mathcal{X}_1 \cup \mathcal{X}_2, inv, prob)$$

such that

- $inv(l_1, l_2) = inv_1(l_1) \wedge inv_2(l_2)$ for all $(l_1, l_2) \in L_1 \times L_2$;
- $((l_1, l_2), g, a, \mathsf{p}) \in prob$ if and only if one of the following conditions holds:
  1. $a \in Act_1 \setminus Act_2$ and there exists $(l_1, g, a, \mathsf{p}_1) \in prob_1$ such that $\mathsf{p} = \mathsf{p}_1 \otimes \{(\emptyset, l_2) \mapsto 1\}$;
  2. $a \in Act_2 \setminus Act_1$ and there exists $(l_2, g, a, \mathsf{p}_2) \in prob_2$ such that $\mathsf{p} = \{(\emptyset, l_1) \mapsto 1\} \otimes \mathsf{p}_2$;
  3. $a \in Act_1 \cap Act_2$ and there exists $(l_i, g_i, a, \mathsf{p}_i) \in prob_i$ for $i = 1, 2$ such that $g = g_1 \wedge g_2$ and $\mathsf{p} = \mathsf{p}_1 \otimes \mathsf{p}_2$.

3 Algorithms for Timed Simulation and Bisimulation

Timed Simulation and Bisimulation. We now define probabilistic timed simulation in the manner of [4,21]. Given two sets Q_1 and Q_2, let $\mathcal{R} \subseteq Q_1 \times Q_2$ be a binary relation. Let $\mu_1 \in \mathsf{Dist}(Q_1)$ and $\mu_2 \in \mathsf{Dist}(Q_2)$ be distributions on Q_1 and Q_2, respectively. A *weight function* [23] for (μ_1, μ_2) with respect to $\mathcal{R}$ is a function $\Delta : Q_1 \times Q_2 \to [0,1]$ such that:

1. $\Delta(q_1, q_2) > 0$ implies $(q_1, q_2) \in \mathcal{R}$;
2. $\sum_{q_2 \in Q_2} \Delta(q_1, q_2) = \mu_1(q_1)$ for each $q_1 \in Q_1$;
3. $\sum_{q_1 \in Q_1} \Delta(q_1, q_2) = \mu_2(q_2)$ for each $q_2 \in Q_2$.

The *lifting* of $\mathcal{R}$ is a relation $\mathcal{L}(\mathcal{R}) \subseteq \mathsf{Dist}(Q_1) \times \mathsf{Dist}(Q_2)$ such that $\mu_1 \mathcal{L}(\mathcal{R}) \mu_2$ if there exists a weight function for (μ_1, μ_2) with respect to $\mathcal{R}$. When clear from the context, we use $\mathcal{R}$ also to refer to the lifting $\mathcal{L}(\mathcal{R})$.

Let $\mathsf{P} = (S, \bar{S}, Act, \to)$ be a PTLTS. A binary relation $\mathcal{R} \subseteq S \times S$ is a *timed simulation* if $s_1 \mathcal{R} s_2$ implies that, for each $(s_1, d, a, \mu_1) \in \to$, there exists $(s_2, d, a, \mu_2) \in \to$ such that $\mu_1 \mathcal{R} \mu_2$. Given two states $s_1, s_2 \in S$, we write $s_1 \preceq s_2$ if there exists a timed simulation $\mathcal{R}$ such that $s_1 \mathcal{R} s_2$. A *timed bisimulation* is a symmetric timed simulation. Given two states $s_1, s_2 \in S$, we write $s_1 \approx s_2$ if there exists a timed bisimulation $\mathcal{R}$ such that $s_1 \mathcal{R} s_2$.

Let $s \in S$, $d \in \mathbb{R}_{\geq 0}$ and $a \in Act$. Consider the largest set $\{\mu_1, ..., \mu_k\}$ of distributions over S such that $(s, d, a, \mu_i) \in \to$ for $1 \leq i \leq k$. Then the tuple (s, d, a, μ) is a *combined transition* if there exists a sequence $c_1, ..., c_k$ of real numbers in $[0,1]$ such that $\sum_{1 \leq i \leq k} c_i = 1$ where $\mu = \sum_{1 \leq i \leq k} c_i \mu_i$. We let $\mathsf{Combined}(s, d, a)$ denote the set of combined transitions associated with s, d and a. A binary relation $\mathcal{R} \subseteq S \times S$ is a *probabilistic timed simulation* if $s_1 \mathcal{R} s_2$ implies that, for each $(s_1, d, a, \mu_1) \in \to$, there exists a combined transition $(s_2, d, a, \mu_2) \in \mathsf{Combined}(s_2, d, a)$ such that $\mu_1 \mathcal{R} \mu_2$. Given two states $s_1, s_2 \in S$, we write $s_1 \preceq^p s_2$ if there exists a probabilistic timed simulation $\mathcal{R}$ such that $s_1 \mathcal{R} s_2$. A *probabilistic timed bisimulation* is a symmetric probabilistic timed simulation. Given two states $s_1, s_2 \in S$, we write $s_1 \approx^p s_2$ if there exists a probabilistic timed bisimulation $\mathcal{R}$ such that $s_1 \mathcal{R} s_2$.

Let $\mathcal{A} = (L_\mathcal{A}, \bar{L}_\mathcal{A}, Act_\mathcal{A}, \mathcal{X}_\mathcal{A}, inv_\mathcal{A}, prob_\mathcal{A})$ and $\mathcal{B} = (L_\mathcal{B}, \bar{L}_\mathcal{B}, Act_\mathcal{B}, \mathcal{X}_\mathcal{B}, inv_\mathcal{B}, prob_\mathcal{B})$ be two PTA. The disjoint composition of $\mathcal{A}$ and $\mathcal{B}$ is the PTA $\mathcal{A} \uplus \mathcal{B} = (L_\mathcal{A} \uplus L_\mathcal{B}, \bar{L}_\mathcal{A} \uplus \bar{L}_\mathcal{B}, Act_\mathcal{A} \uplus Act_\mathcal{B}, \mathcal{X}_\mathcal{A} \uplus \mathcal{X}_\mathcal{B}, inv, prob_\mathcal{A} \uplus prob_\mathcal{B})$, where $inv(l) = inv_\mathcal{A}(l)$ if $l \in L_\mathcal{A}$, and $inv(l) = inv_\mathcal{B}(l)$ if $l \in L_\mathcal{B}$. Given $\mathcal{A} \uplus \mathcal{B}$, a (probabilistic) timed simulation relation $\mathcal{R}$ on $[\![\mathcal{A} \uplus \mathcal{B}]\!]$ is *initialized* if and only if, for every $l_\mathcal{A} \in \bar{L}_\mathcal{A}$, there exists some $l_\mathcal{B} \in \bar{L}_\mathcal{B}$ such that $(l_\mathcal{A}, \mathbf{0}) \mathcal{R} (l_\mathcal{B}, \mathbf{0})$. We write $\mathcal{A} \preceq \mathcal{B}$ if there exists an initialized (probabilistic) timed simulation relation on $[\![\mathcal{A} \uplus \mathcal{B}]\!]$. It follows from [4] that $\preceq$ and $\preceq^p$ are preorders, and, together with [12], that $\preceq$ and $\preceq^p$ are compositional in the following sense: given the PTA $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$, if $\mathcal{A} \preceq \mathcal{B}$ then $\mathcal{A} \| \mathcal{C} \preceq \mathcal{B} \| \mathcal{C}$, and if $\mathcal{A} \preceq^p \mathcal{B}$ then $\mathcal{A} \| \mathcal{C} \preceq^p \mathcal{B} \| \mathcal{C}$.

Example 1. Consider the two PTA fragments $\mathcal{A}$ (left) and $\mathcal{B}$ (right) in Figure 1. We write the invariant conditions within the locations they refer to and we omit them when they are `true`. A probabilistic edge (l, g, a, p) is represented as an

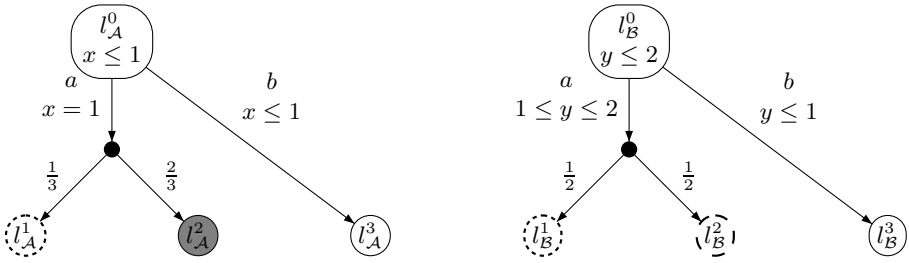


Fig. 1. An example of timed simulation

arc exiting location l, labeled with the action a and guard g. The distribution p is represented by connecting the location l to a dot from which arcs, labeled with a probability, reach the locations indicated by the elements of $\mathsf{support}(\mathsf{p})$. For simplicity, when $\mathsf{p}(\emptyset, l') = 1$, we draw a direct arc from l to l'. The initial locations of $\mathcal{A}$ and $\mathcal{B}$ are $l^0_{\mathcal{A}}$ and $l^0_{\mathcal{B}}$, respectively. Assume that there is subsequent behaviour from the bottom line of locations in the figure such that, for all clock valuations $v \in \mathbb{R}^{\{x\}}_{\ge 0}$ and $v' \in \mathbb{R}^{\{y\}}_{\ge 0}$, we have $(l^1_{\mathcal{A}}, v) \preceq (l^1_{\mathcal{B}}, v')$, $(l^2_{\mathcal{A}}, v) \preceq (l^1_{\mathcal{B}}, v')$, $(l^2_{\mathcal{A}}, v) \preceq (l^2_{\mathcal{B}}, v')$ and $(l^3_{\mathcal{A}}, v) \preceq (l^3_{\mathcal{B}}, v')$ (states in $\preceq$ are indicated by locations of the same color and shape, except for states with the gray location, which are timed simulated by the states of the short and long dashed shape). From $(l^0_{\mathcal{A}}, \mathbf{0})$, there exist transitions $((l^0_{\mathcal{A}}, \mathbf{0}), d, b, \{(l^3_{\mathcal{A}}, v_d) \mapsto 1\}) \in \rightarrow_{\mathcal{A}}$, which can be mimicked by transitions $((l^0_{\mathcal{B}}, \mathbf{0}), d, b, \{(l^3_{\mathcal{B}}, v'_d) \mapsto 1\}) \in \rightarrow_{\mathcal{B}}$ from $(l^0_{\mathcal{B}}, \mathbf{0})$, where $d \le 1$ and $v_d(x) = v'_d(y) = d$. From $(l^0_{\mathcal{A}}, \mathbf{0})$, there exists the single a-labelled transition $((l^0_{\mathcal{A}}, \mathbf{0}), 1, a, \mu_{\mathcal{A}}) \in \rightarrow_{\mathcal{A}}$ such that $\mu_{\mathcal{A}}(l^1_{\mathcal{A}}, v_1) = \frac{1}{3}$ and $\mu_{\mathcal{A}}(l^2_{\mathcal{A}}, v_1) = \frac{2}{3}$. This transition can be mimicked from $(l^0_{\mathcal{B}}, \mathbf{0})$ by the transition $((l^0_{\mathcal{B}}, \mathbf{0}), 1, a, \mu_{\mathcal{B}}) \in \rightarrow_{\mathcal{B}}$ such that $\mu_{\mathcal{B}}(l^1_{\mathcal{B}}, v'_1) = \frac{1}{2}$ and $\mu_{\mathcal{B}}(l^2_{\mathcal{B}}, v'_1) = \frac{1}{2}$. Furthermore, there exists a weight function Δ for $(\mu_{\mathcal{A}}, \mu_{\mathcal{B}})$ with respect to $\preceq$: we can consider $\Delta((l^1_{\mathcal{A}}, v_1), (l^1_{\mathcal{B}}, v_1)) = \frac{1}{3}$, $\Delta((l^2_{\mathcal{A}}, v_1), (l^1_{\mathcal{B}}, v_1)) = \frac{1}{6}$ and $\Delta((l^2_{\mathcal{A}}, v_1), (l^2_{\mathcal{B}}, v_1)) = \frac{1}{2}$. It can be verified that Δ satisfies the conditions of a weight function for $(\mu_{\mathcal{A}}, \mu_{\mathcal{B}})$ with respect to $\preceq$. Hence we have $(l^0_{\mathcal{A}}, \mathbf{0}) \preceq (l^0_{\mathcal{B}}, \mathbf{0})$. From this, we conclude that $\mathcal{A} \preceq \mathcal{B}$.

Example 2. Consider the two PTA fragments $\mathcal{A}$ (left) and $\mathcal{B}$ (right) in Figure 2. Here we suppose that, for all clock valuations $v \in \mathbb{R}^{\{x\}}_{\ge 0}$ and $v' \in \mathbb{R}^{\{y\}}_{\ge 0}$, we have $(l^1_{\mathcal{A}}, v) \preceq (l^1_{\mathcal{B}}, v')$, $(l^2_{\mathcal{A}}, v) \preceq (l^2_{\mathcal{B}}, v')$, $(l^1_{\mathcal{A}}, v) \preceq (l^3_{\mathcal{B}}, v')$ and $(l^2_{\mathcal{A}}, v) \preceq (l^4_{\mathcal{B}}, v')$. It holds that $\mathcal{A} \not\preceq \mathcal{B}$, because $\mathcal{A}$ can reach a location $l^1_{\mathcal{A}}$ in a single step with probability $\frac{1}{2}$, while $\mathcal{B}$ can reach a related location ($l^1_{\mathcal{B}}$ or $l^3_{\mathcal{B}}$) either with probability $\frac{1}{3}$ or $\frac{2}{3}$, but not with probability $\frac{1}{2}$. However, there exists a combined transition for $\mathcal{B}$ obtained by assigning $\frac{1}{2}$ to the two illustrated probabilistic edges from $l^0_{\mathcal{B}}$, and for which it is possible to reach $l^1_{\mathcal{B}}$ or $l^3_{\mathcal{B}}$ with probability $\frac{1}{2}$. Continuing this reasoning also for $l^2_{\mathcal{A}}$, we can verify that $\mathcal{A} \preceq^p \mathcal{B}$.

We now present an algorithm for deciding whether a PTA (probabilistically) timed simulates another PTA. Our approach is to extend the techniques of [12,13], which were applied to non-probabilistic timed automata/timed games, to the case

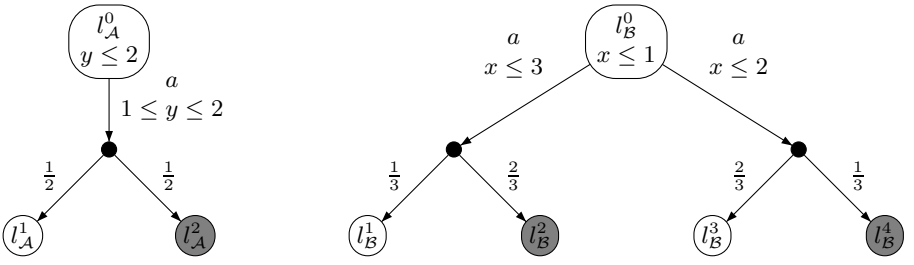


Fig. 2. An example of probabilistic timed simulation

of PTA. We focus our attention on the case of timed simulation. Formally, for two PTA $\mathcal{A}$ and $\mathcal{B}$, our aim is to decide whether $\mathcal{A} \preceq \mathcal{B}$. We comment briefly on how to extend the algorithm to the case of timed bisimulation, and to probabilistic timed (bi)simulation, at the end of this section.

Region Equivalence. We begin by recalling the standard definition of region equivalence [1]. For $r \in \mathbb{R}_{\geq 0}$, we let $\mathsf{frac}(r) = r - \lfloor r \rfloor$. Let $\mathcal{A} = (L, \bar{L}, Act, \mathcal{X}, inv, prob)$ be a PTA, and let c_{max} be the maximal constant to which a clock is compared in any of the guards of probabilistic edges or invariants of $\mathcal{A}$. Two clock valuations $v, v' \in \mathbb{R}^{\mathcal{X}}_{\geq 0}$ are *clock equivalent* if the following conditions are satisfied: (1) for all clocks $x \in \mathcal{X}$, we have $v(x) \leq c_{max}$ if and only if $v'(x) \leq c_{max}$; (2) for all clocks $x \in \mathcal{X}$ with $v(x) \leq c_{max}$, we have $\lfloor v(x) \rfloor = \lfloor v'(x) \rfloor$; (3) for all clocks $x, y \in \mathcal{X}$ with $v(x) \leq c_{max}$ and $v(y) \leq c_{max}$, we have $\mathsf{frac}(v(x)) \leq \mathsf{frac}(v(y))$ if and only if $\mathsf{frac}(v'(x)) \leq \mathsf{frac}(v'(y))$; and (4) for all clocks $x \in \mathcal{X}$ with $v(x) \leq c_{max}$, we have $\mathsf{frac}(v(x)) = 0$ if and only if $\mathsf{frac}(v'(x)) = 0$. Two states $(l, v), (l', v')$ of $[\![\mathcal{A}]\!]$ are *region equivalent*, written $(l, v) \equiv (l', v')$, if (1) $l = l'$, and (2) v and v' are clock equivalent. A *region* is an equivalence class of region equivalence, and let $\mathsf{Regions}_{\mathcal{A}}$ be the set of regions of $\mathcal{A}$. Given a state (l, v), we use $[(l, v)]$ to denote the region to which (l, v) belongs. The number of regions corresponding to the PTA $\mathcal{A}$ is bounded by $|L| \cdot (2c_{max} + 2)^{|\mathcal{X}|} \cdot |\mathcal{X}|! \cdot 2^{|\mathcal{X}|}$.

For deciding timed simulation on two PTA, we consider region equivalence over the state space of the parallel composition of the PTA. The subsequent algorithm for deciding whether $\mathcal{A} \preceq \mathcal{B}$ operates on the set of regions $\mathsf{Regions}_{\mathcal{A}\|\mathcal{B}}$. In the following, given two states $(l_{\mathcal{A}}, v_{\mathcal{A}})$ of $\mathcal{A}$ and $(l_{\mathcal{B}}, v_{\mathcal{B}})$ of $\mathcal{B}$, the unique state of $\mathcal{A}\|\mathcal{B}$ corresponding to these states is written $((l_{\mathcal{A}}, l_{\mathcal{B}}), v_{\mathcal{A}\|\mathcal{B}})$, where $v_{\mathcal{A}\|\mathcal{B}}(x) = v_{\mathcal{A}}(x)$ if $x \in \mathcal{X}_{\mathcal{A}}$, and $v_{\mathcal{A}\|\mathcal{B}}(x) = v_{\mathcal{B}}(x)$ if $x \in \mathcal{X}_{\mathcal{B}}$ (note that it is possible that $v_{\mathcal{A}\|\mathcal{B}} \not\models inv_{\mathcal{A}\|\mathcal{B}}(l_{\mathcal{A}}, l_{\mathcal{B}})$, where $inv_{\mathcal{A}\|\mathcal{B}}$ denotes the invariant condition of $\mathcal{A}\|\mathcal{B}$; it is trivial to decide timed simulation on such states, and henceforth we do not consider states of this form).

Restriction to a Finite Set of Time Durations. Let $((l_{\mathcal{A}}, v_{\mathcal{A}}), (l_{\mathcal{B}}, v_{\mathcal{B}})) \in S_{\mathcal{A}} \times S_{\mathcal{B}}$. Let $x_1, ..., x_n \subseteq \mathcal{X}_{\mathcal{A}} \cup \mathcal{X}_{\mathcal{B}}$ be the clocks such that $v_{\mathcal{A}\|\mathcal{B}}(x_i) < c_{max}$ for each $1 \leq i \leq n$, ordered such that $\tau_1 \leq \tau_2 \leq \cdots \leq \tau_n$, where $\tau_i = \mathsf{frac}(v_{\mathcal{A}\|\mathcal{B}}(x_i))$ for each $1 \leq i \leq n$. Furthermore, let $\tau_0 = 0$ and $\tau_{n+1} = 1$. We also define $\min(v_{\mathcal{A}}, v_{\mathcal{B}}) = \min\{\lfloor v_{\mathcal{A}\|\mathcal{B}}(x_1) \rfloor, \cdots, \lfloor v_{\mathcal{A}\|\mathcal{B}}(x_n) \rfloor\}$. We now recall the definition of

a *finite* set $Times((l_{\mathcal{A}}, v_{\mathcal{A}}), (l_{\mathcal{B}}, v_{\mathcal{B}}))$ of time durations from [12,13]; it will suffice to consider only the time durations in this set in the subsequent algorithm.

$$\begin{array}{l} Times((l_{\mathcal{A}}, v_{\mathcal{A}}), (l_{\mathcal{B}}, v_{\mathcal{B}})) = \\ \quad \{c - \frac{1}{2}(\tau_i + \tau_{i+1}) \mid c \in \mathbb{N} \text{ and } 0 \leq i \leq n \text{ and } 1 \leq c \leq c_{max} - \min(v_{\mathcal{A}}, v_{\mathcal{B}})\} \cup \\ \quad \{c - \tau_i \mid c \in \mathbb{N} \text{ and } 1 \leq i \leq n \text{ and } 1 \leq c \leq c_{max} - \min(v_{\mathcal{A}}, v_{\mathcal{B}})\} \cup \\ \quad \{c \mid c \in \mathbb{N} \text{ and } 0 \leq c \leq c_{max} + 1 - \min(v_{\mathcal{A}}, v_{\mathcal{B}})\} \ . \end{array}$$

Let $I = \{c \mid c \in \mathbb{N} \text{ and } 1 \leq c \leq c_{max} - \min(v_{\mathcal{A}}, v_{\mathcal{B}})\}$. The finite set of durations $Times((l_{\mathcal{A}}, v_{\mathcal{A}}), (l_{\mathcal{B}}, v_{\mathcal{B}}))$ contains: (1) the distances between the mid-points of the intervals (τ_i, τ_{i+1}) and the integers in I, (2) the distances between τ_i and the integers in I, (3) the set of integers in $\{c \mid c \in \mathbb{N} \text{ and } 1 \leq c \leq c_{max} - \min(v_{\mathcal{A}}, v_{\mathcal{B}})\}$. Following [12,13], the distance $d \in Times((l_{\mathcal{A}}, v_{\mathcal{A}}), (l_{\mathcal{B}}, v_{\mathcal{B}}))$ between the mid-point $\frac{1}{2}(\tau_i + \tau_{i+1})$ and an integer $c \in I$ can be used as a representative for all the time delays between $c - \tau_{i+1}$ and $c - \tau_i$.

One-Step Goodness. We now define two notions of "goodness", which we will use subsequently to refer to a single transition step from each of the PTA $\mathcal{A}$ and $\mathcal{B}$. This notion will be presented in two versions: a concrete version, defined on the states of $\mathcal{A}$ and $\mathcal{B}$, and a symbolic version, defined on $\mathsf{Regions}_{\mathcal{A}\|\mathcal{B}}$ and using time durations taken from $Times(\_, \_)$. Analogues of these notions, and their associated results, can be found in [12,13].

Let $\mathcal{R} \subseteq S_{\mathcal{A}} \times S_{\mathcal{B}}$, and let $(s_{\mathcal{A}}, s_{\mathcal{B}}) \in S_{\mathcal{A}} \times S_{\mathcal{B}}$. Then $(s_{\mathcal{A}}, s_{\mathcal{B}})$ *is concretely good in* $\mathcal{R}$ if, for each $(s_{\mathcal{A}}, d, a, \mu_{\mathcal{A}}) \in \rightarrow_{\mathcal{A}}$, there exists $(s_{\mathcal{B}}, d, a, \mu_{\mathcal{B}}) \in \rightarrow_{\mathcal{B}}$ such that $\mu_{\mathcal{A}} \mathcal{R} \mu_{\mathcal{B}}$. The following lemma states that concrete goodness, with respect to a relation described as a union of regions, is invariant over regions.

Lemma 1. *Let* $\Gamma \subseteq \mathsf{Regions}_{\mathcal{A}\|\mathcal{B}}$*, and let* $R \in \mathsf{Regions}_{\mathcal{A}\|\mathcal{B}}$ *be such that there exists* $(s_{\mathcal{A}}, s_{\mathcal{B}}) \in R$ *which is concretely good in* $\bigcup_{R' \in \Gamma} R'$*. Then* each $(s'_{\mathcal{A}}, s'_{\mathcal{B}}) \in R$ *is concretely good in* $\bigcup_{R' \in \Gamma} R'$*.*

Given a relation $\mathcal{R} \subseteq S_{\mathcal{A}} \times S_{\mathcal{B}}$, we let $\Gamma_{\mathcal{R}} = \{R \in \mathsf{Regions}_{\mathcal{A}\|\mathcal{B}} \mid R \cap \mathcal{R} \neq \emptyset\}$.

Proposition 1. *If* $\mathcal{R} \subseteq S_{\mathcal{A}} \times S_{\mathcal{B}}$ *is a timed simulation, then* $\bigcup_{R' \in \Gamma_{\mathcal{R}}} R'$ *is a timed simulation.*

Let $R \in \mathsf{Regions}_{\mathcal{A}\|\mathcal{B}}$. Let $X_{\mathcal{A}} \subseteq \mathcal{X}_{\mathcal{A}}$, $X_{\mathcal{B}} \subseteq \mathcal{X}_{\mathcal{B}}$, $l_{\mathcal{A}} \in L_{\mathcal{A}}$ and $l_{\mathcal{B}} \in L_{\mathcal{B}}$. Then we write $R[X_{\mathcal{A}} \cup X_{\mathcal{B}} := 0, loc := l_{\mathcal{A}}, l_{\mathcal{B}}]$ to denote the region which has the location components $l_{\mathcal{A}}$ and $l_{\mathcal{B}}$, and the clock equivalence class equal to R except that the clocks in $X_{\mathcal{A}}$ and $X_{\mathcal{B}}$ are reset to 0. Now let $\Gamma \subseteq \mathsf{Regions}_{\mathcal{A}\|\mathcal{B}}$. Let $\mathsf{p}_{\mathcal{A}} \in \mathsf{Dist}(2^{\mathcal{X}_{\mathcal{A}}} \times L_{\mathcal{A}})$ and $\mathsf{p}_{\mathcal{B}} \in \mathsf{Dist}(2^{\mathcal{X}_{\mathcal{B}}} \times L_{\mathcal{B}})$. Then $\mathcal{R}_{R,\Gamma} \subseteq \mathsf{support}(\mathsf{p}_{\mathcal{A}}) \times \mathsf{support}(\mathsf{p}_{\mathcal{B}})$ is defined as follows: for each $(X_{\mathcal{A}}, l_{\mathcal{A}}) \in \mathsf{support}(\mathsf{p}_{\mathcal{A}})$ and $(X_{\mathcal{B}}, l_{\mathcal{B}}) \in \mathsf{support}(\mathsf{p}_{\mathcal{B}})$, we have $(X_{\mathcal{A}}, l_{\mathcal{A}}) \mathcal{R}_{R,\Gamma} (X_{\mathcal{B}}, l_{\mathcal{B}})$ if and only if $R[X_{\mathcal{A}} \cup X_{\mathcal{B}} := 0, loc := l_{\mathcal{A}}, l_{\mathcal{B}}] \in \Gamma$.

The region R *is symbolically good in* Γ if there exists $(s_{\mathcal{A}}, s_{\mathcal{B}}) \in R$ such that, for each $(s_{\mathcal{A}}, d, a, \mathsf{p}_{\mathcal{A}}) \in \rightarrow_{\mathcal{A}}$ with $d \in Times(s_{\mathcal{A}}, s_{\mathcal{B}})$, there exists a transition $(s_{\mathcal{B}}, d, a, \mathsf{p}_{\mathcal{B}}) \in \rightarrow_{\mathcal{B}}$ such that $\mathsf{p}_{\mathcal{A}} \mathcal{R}_{R',\Gamma} \mathsf{p}_{\mathcal{B}}$, where $R' = [s_{\mathcal{A}} + d, s_{\mathcal{B}} + d]$. The following lemma establishes a connection between symbolic and concrete goodness.

Lemma 2. *Let $\Gamma \subseteq \mathsf{Regions}_{\mathcal{A}\|\mathcal{B}}$, and let $R \in \mathsf{Regions}_{\mathcal{A}\|\mathcal{B}}$ be symbolically good in Γ. Then each $(s_\mathcal{A}, s_\mathcal{B}) \in R$ is concretely good in $\bigcup_{R' \in \Gamma} R'$.*

The proof of Lemma 2 relies on first showing that a state pair $(s_\mathcal{A}, s_\mathcal{B})$ in R which witnesses symbolic goodness in Γ is also concretely good in $\bigcup_{R' \in \Gamma} R'$, which then, by Lemma 1, implies concrete goodness in $\bigcup_{R' \in \Gamma} R'$ for all state pairs in R.

Algorithm. Let $\Omega : 2^{\mathsf{Regions}_{\mathcal{A}\|\mathcal{B}}} \to 2^{\mathsf{Regions}_{\mathcal{A}\|\mathcal{B}}}$ be the monotone operator defined by $\Omega(\Gamma) = \{R \in \Gamma \mid R \text{ is symbolically good in } \Gamma\}$. By Lemma 2, to decide whether a region $R \in \Gamma$ is such that $R \in \Omega(\Gamma)$, the choice of which representative state pair to consider for R is not significant: hence, an arbitrary state pair can be considered. Note also that $|\mathit{Times}(s_\mathcal{A}, s_\mathcal{B})|$, for any $(s_\mathcal{A}, s_\mathcal{B}) \in S_\mathcal{A} \times S_\mathcal{B}$, is exponential in the sizes of $\mathcal{A}$ and $\mathcal{B}$, as is $|\mathsf{Regions}_{\mathcal{A}\|\mathcal{B}}|$. By the results of [14], we have that, for any $\mathsf{p}_\mathcal{A} \in \mathsf{Dist}(2^{\mathcal{X}_\mathcal{A}} \times L_\mathcal{A}), \mathsf{p}_\mathcal{B} \in \mathsf{Dist}(2^{\mathcal{X}_\mathcal{B}} \times L_\mathcal{B})$, it is possible to decide $\mathsf{p}_\mathcal{A} \mathcal{R}_{R,\Gamma} \mathsf{p}_\mathcal{B}$ in polynomial time. Hence, we can compute $\Omega(\Gamma)$ in exponential time in the sizes of $\mathcal{A}$ and $\mathcal{B}$.

Lemma 3. *If $\mathcal{R} \subseteq S_\mathcal{A} \times S_\mathcal{B}$ is a timed simulation, then $\Gamma_\mathcal{R}$ is a fixpoint of Ω.*

Proposition 2. *Let $\Gamma \subseteq \mathsf{Regions}_{\mathcal{A}\|\mathcal{B}}$ be a set of regions. Then Γ is a fixpoint of Ω if and only if $\bigcup_{R \in \Gamma} R$ is a timed simulation.*

The operator Ω provides the basis of the algorithm for deciding whether $\mathcal{A} \preceq \mathcal{B}$. Our aim is to compute its greatest fixpoint Γ_{max}. Let $\Gamma_0 = \mathsf{Regions}_{\mathcal{A}\|\mathcal{B}}$, and let $\Gamma_{i+1} = \Omega(\Gamma_i)$ for each $i \geq 0$. From the monotonicity of Ω, for some $i \leq |\mathsf{Regions}_{\mathcal{A}\|\mathcal{B}}|$, we have $\Gamma_i = \Omega(\Gamma_i)$. Hence it suffices to apply Ω at most $|\mathsf{Regions}_{\mathcal{A}\|\mathcal{B}}|$ times, and therefore Γ_{max} can be computed in exponential time in the sizes of $\mathcal{A}$ and $\mathcal{B}$.

Let $\mathcal{R}_{max}$ be the maximal timed simulation relation from $\mathcal{A}$ to $\mathcal{B}$. By Proposition 1, we have that $\mathcal{R}_{max}$ is a union of regions. Given the computation of Γ_{max}, by Proposition 2 we have that $\mathcal{R}_{max} = \bigcup_{R \in \Gamma_{max}} R$. We then have that the following are equivalent:

- $\mathcal{A} \preceq \mathcal{B}$;
- for every $l_\mathcal{A} \in \bar{L}_\mathcal{A}$, there exists some $l_\mathcal{B} \in \bar{L}_\mathcal{B}$ such that $(l_\mathcal{A}, \mathbf{0}) \mathcal{R}_{max} (l_\mathcal{B}, \mathbf{0})$;
- for every $l_\mathcal{A} \in \bar{L}_\mathcal{A}$, there exists some $l_\mathcal{B} \in \bar{L}_\mathcal{B}$ such that $[((l_\mathcal{A}, l_\mathcal{B}), \mathbf{0})] \in \Gamma_{max}$.

We can adapt the algorithm above to obtain an algorithm for deciding whether two PTA $\mathcal{A}$ and $\mathcal{B}$ are timed bisimilar. First, we note that concrete and symbolic goodness are required to be redefined to obtain *symmetric* versions; furthermore, concrete goodness is defined with respect to an equivalence relation $\mathcal{R}$, and symbolic goodness is defined with respect to a set Γ of regions which induces an equivalence relation (that is, $\bigcup_{R \in \Gamma} R$ is an equivalence relation). Then it is possible to define a version of the operator Ω which makes reference to the new, symmetric notion of symbolic goodness.

From the results of [24], we have that deciding timed simulation or timed bisimulation is EXPTIME-hard. In combination with the above, this gives us the following theorem.

Theorem 1. *Given two PTA $\mathcal{A}$ and $\mathcal{B}$, the following two problems are EXPTIME-complete: (1) checking whether $\mathcal{A} \preceq \mathcal{B}$; (2) checking whether $\mathcal{A} \approx \mathcal{B}$.*

Probabilistic Timed Simulation. The results of the previous subsection can be adapted to the case of probabilistic timed simulation and bisimulation. As in the previous section, we can obtain an algorithm for probabilistic timed bisimulation from an algorithm for probabilistic timed simulation, and hence we consider the latter. Formally, for two PTA $\mathcal{A}$ and $\mathcal{B}$, our aim is to decide whether $\mathcal{A} \preceq^p \mathcal{B}$.

Firstly, we extend the notions of concrete and symbolic goodness to accommodate the possibility of $\mathcal{B}$ choosing combined transitions. For concrete goodness, this is done simply by replacing the condition $(s_\mathcal{B}, d, a, \mu_\mathcal{B}) \in \rightarrow_\mathcal{B}$ with $(s_\mathcal{B}, d, a, \mu_\mathcal{B}) \in \mathsf{Combined}(s_\mathcal{B}, d, a)$. For symbolic goodness, we first apply the notion of combined transition to the case of distributions featured in probabilistic edges: given the largest set $\{\mathsf{p}_1, ..., \mathsf{p}_k\}$ of distributions such that $(s_\mathcal{B}, d, a, \mathsf{p}_i) \in \rightarrow_\mathcal{B}$ for $1 \leq i \leq k$, we then write $\mathsf{Combined}^p(s_\mathcal{B}, d, a)$ for the set of all tuples $(s_\mathcal{B}, d, a, \mathsf{p})$ such that there exists a sequence $c_1, ..., c_k$ of real numbers in $[0, 1]$ with $\sum_{1 \leq i \leq k} c_i = 1$ and $\mathsf{p} = \sum_{1 \leq i \leq k} c_i \mathsf{p}_i$. Then, to obtain the new notion of symbolic goodness, we replace the condition $(s_\mathcal{B}, d, a, \mathsf{p}_\mathcal{B}) \in \rightarrow_\mathcal{B}$ with $(s_\mathcal{B}, d, a, \mathsf{p}_\mathcal{B}) \in \mathsf{Combined}^p(s_\mathcal{B}, d, a)$. Then the operator Ω is adapted to take into account the new notion of symbolic goodness. Using the results of [15], for a given $\Gamma \subseteq \mathsf{Regions}_{\mathcal{A}\|\mathcal{B}}$, it is possible to compute $\Omega(\Gamma)$ in exponential time in the sizes of $\mathcal{A}$ and $\mathcal{B}$. This reasoning, combined with that concerning the exponential number of iterations of Ω given for timed simulation, then can be used to obtain the following result.

Theorem 2. *Given two PTA $\mathcal{A}$ and $\mathcal{B}$, the following two problems are EXPTIME-complete: (1) checking whether $\mathcal{A} \preceq^p \mathcal{B}$; (2) checking whether $\mathcal{A} \approx^p \mathcal{B}$.*

4 Logical Characterization of Bisimulation

In this section we give a logical characterization of our timed bisimulation and probabilistic timed bisimulation relations. Recall that [18] presents an extension of Hennessy-Milner logic [17] for probabilistic automata. The principal novelty of the logic of [18] is that its semantics is defined over distributions on states, rather than over states. Here we extend the logic of [18] with constraints on the duration of transitions, similarly to [16,13].

We now present the syntax of the logic. The logic PTLogic is syntactically defined by the following formulas:

$$\psi ::= true \mid \neg\psi \mid \psi \wedge \psi \mid \langle a, \sim c\rangle\psi \mid [\psi]_p$$

where $a \in Act$ is an action, $c \in \mathbb{R}_{\geq 0}$ is a constant, and $p \in [0, 1]$ is a probability. Note that we will discuss the sub-logic of PTLogic in which $c \in \mathbb{Q}_{\geq 0}$ (where $\mathbb{Q}_{\geq 0}$ denotes the set of non-negative rationals) at the end of this section.

Let P be a PTLTS. Given a distribution $\mu \in \mathsf{Dist}(S)$ and a set $S' \subseteq S$ of states, we let $\mu(S') = \sum_{s \in S'} \mu(s)$. Let ψ be a formula in PTLogic and μ be

a distribution over the set of states of a PTLTS P. We say that μ *satisfies* ψ, written $\mu \models \psi$, according to the following:

$$
\begin{array}{lll}
\mu \models \mathit{true} & & \\
\mu \models \neg\psi & \text{iff} & \mu \not\models \psi \\
\mu \models \psi_1 \wedge \psi_2 & \text{iff} & \text{both } \mu \models \psi_1 \text{ and } \mu \models \psi_2 \\
\mu \models \langle a, \sim c \rangle \psi & \text{iff} & \text{for all } s \in \mathsf{support}(\mu) \text{ there exists } (s, d, a, \mu') \in \rightarrow \text{ such that} \\
 & & d \sim c \text{ and } \mu' \models \psi \\
\mu \models [\psi]_p & \text{iff} & \mu(\{\!|\psi|\!\}) \geq p
\end{array}
$$

where $\{\!|\psi|\!\} = \{s \in S \mid s \models \psi\}$ denotes the set of all states of P that satisfy the PTLogic formula ψ, and where $s \models \psi$ if and only if $\{s \mapsto 1\} \models \psi$.

We now show that timed bisimilar states of the semantic PTLTS $[\![\mathcal{A}]\!] = (S, \bar{S}, \mathit{Act}, \rightarrow)$ resulting from a PTA $\mathcal{A}$ satisfy the same formulas of PTLogic and, conversely, if there exists a formula of PTLogic that is satisfied in one state and not another, then these two states are not timed bisimilar. We introduce the following notation. Let $\mathcal{F}$ be the set of all PTLogic formulas. Given a set $\mathcal{F}' \subseteq \mathcal{F}$ of PTLogic formulas, we use $\mathcal{F}'(s)$ and $\mathcal{F}'(\mu)$ to denote the subset of formulas of $\mathcal{F}'$ that are satisfied at state $s \in S$ and by distribution $\mu \in \mathsf{Dist}(S)$, respectively. The *depth* of a PTLogic formula ψ is defined as the maximum number of nested $\langle a, \sim c \rangle \psi'$ operators that occur in ψ. Let $\mathcal{F}_n$ be the set of PTLogic formulas of depth n, and let $\bowtie_n \subseteq S \times S$ be the relation such that $s \bowtie_n s'$ if and only if $\mathcal{F}_n(s) = \mathcal{F}_n(s')$. Then, as in [18], we have the following results.

Lemma 4. *1. For each pair $s, s' \in S$ of states, if $\mathcal{F}(s) \neq \mathcal{F}(s')$ then $\mathcal{F}(s) \not\subseteq \mathcal{F}(s')$.*
2. For each pair $s, s' \in S$ of states, $\mathcal{F}_0(s) = \mathcal{F}_0(s')$.
3. Let $\mathcal{R} \subseteq \bowtie_n$ for some $n \in \mathbb{N}$. Then, for each pair $\mu, \mu' \in \mathsf{Dist}(S)$, we have that $\mu \mathcal{R} \mu'$ implies $\mathcal{F}_n(\mu) = \mathcal{F}_n(\mu')$.

The first two points of Lemma 4 follow from the definitions in a straightforward manner. The third point can be shown in a manner similar to the analogous result of [18].

Let $\approx_0 = S \times S$ (that is, the relation $\approx_0$ relates all states). For $n \in \mathbb{N}$, let $\approx_{n+1} \subseteq S \times S$ be the equivalence relation defined as follows: for each $s, s' \in S$, $s \approx_{n+1} s'$ implies that, for each $(s, d, a, \mu) \in \rightarrow$, there exists $(s', d, a, \mu') \in \rightarrow$ such that $\mu \approx_n \mu'$. On semantic PTLTS of PTAs, we have that $\approx = \bigcap_{n \in \mathbb{N}} \approx_n$.

Theorem 3. *Let $[\![\mathcal{A}]\!] = (S, \bar{S}, \mathit{Act}, \rightarrow)$ be the semantic PTLTS of the PTA $\mathcal{A}$. For each pair $s, s' \in S$ of states, we have $s \approx s'$ if and only if $\mathcal{F}(s) = \mathcal{F}(s')$.*

Proof. The proof proceeds along the same lines as that of Theorem 1 of [18]; for completeness, we present the overall structure of the proof. We proceed by induction on $n \in \mathbb{N}$, and show that $s \approx_n s'$ if and only if $\mathcal{F}_n(s) = \mathcal{F}_n(s')$. The base case follows from point 2 of Lemma 4 and the definition of $\approx_0$. We now consider both directions of the inductive step.

($\Rightarrow$) Let $s \approx_{n+1} s'$. We require that $\mathcal{F}_{n+1}(s) = \mathcal{F}_{n+1}(s')$, which requires showing that, for all $\psi \in \mathcal{F}_{n+1}$, we have $s \models \psi$ if and only if $s' \models \psi$. The cases of the Boolean combinators and probabilistic operator $[\psi]_p$ are similar to

the analogous cases in [18]. Consider the case of $\psi = \langle a, \sim c\rangle\phi$. Then, by the semantics of PTLogic, there exists $(s, d, a, \mu) \in\!\!\rightarrow$ such that $d \sim c$ and $\mu \models \phi$. From $s \approx_{n+1} s'$, there exists $(s', d, a, \mu') \in\!\!\rightarrow$ such that $\mu \approx_n \mu'$. From $\mu \approx_n \mu'$ and point 3 of Lemma 4, we have that $\mathcal{F}_n(\mu) = \mathcal{F}_n(\mu')$. Noting that $\phi \in \mathcal{F}_n$, then from $\mu \models \phi$ we have $\mu' \models \phi$. From this fact, and the observation that $d \sim c$, we have that $s' \models \langle a, \sim c\rangle\phi$.

($\Leftarrow$) We proceed by showing that $s \not\approx_{n+1} s'$ implies $\mathcal{F}_{n+1}(s) \neq \mathcal{F}_{n+1}(s')$. Let $\{[t_i]_n\}_{i\in I}$ be an enumeration of the equivalence classes of $\approx_n$ (there will be a finite number of such classes by the results of Section 3, in contrast to possibly countably infinite number in [18]). For each $i \in I$, by induction and point 1 of Lemma 4, we can construct a formula ϕ_i which is satisfied only by states in $[t_i]_n$. We then select some $(s, d, a, \mu) \in\!\!\rightarrow$ such that there does not exist any $(s', d, a, \mu') \in\!\!\rightarrow$ for which $\mu \approx_n \mu'$. Such (s, d, a, μ) exists because $s \not\approx_{n+1} s'$. Let $\phi = \bigwedge_{i\in I}[\phi_i]_{\mu([t_i]_n)}$. Clearly $\mu \models \phi$, and hence $s \models \langle a, = d\rangle\phi$. Aiming for a contradiction, assume that $\mathcal{F}_{n+1}(s) = \mathcal{F}_{n+1}(s')$. Then $s' \models \langle a, = d\rangle\phi$. This implies the existence of $(s', d, a, \mu'') \in\!\!\rightarrow$ such that $\mu'' \models \phi$. This is turn implies that $\mu''([t_i]_n) = \mu([t_i]_n)$ for each $i \in I$, which implies that $\mu \approx_n \mu''$, contradicting $s \not\approx_{n+1} s'$. □

Probabilistic Timed Bisimulation. As in [18], the above material can be adapted to the case of probabilistic timed bisimulation in the following way. First we replace the operator $\langle a, \sim c\rangle\psi$ in PTLogic with the operator $\langle\!\cdot a, \sim c\rangle\psi$, which has the following semantics: given a distribution μ, we have $\mu \models \langle\!\cdot a, \sim c\rangle\psi$ if and only if for all $s \in \mathsf{support}(\mu)$ there exists $(s, d, a, \mu') \in \mathsf{Combined}(s, d, a)$ such that $d \sim c$ and $\mu' \models \psi$. Let $\mathcal{F}^{\bullet}$ denote the set of formulas of the resulting logic. The proof of Theorem 3 can be adapted to the new logic by changing references to transitions to references to combined transitions as necessary, because timing issues are independent of issues concerning combined transitions. This leads to the following result.

Theorem 4. *Let $[\![\mathcal{A}]\!] = (S, \bar{S}, Act, \rightarrow)$ be the semantic PTLTS of the PTA $\mathcal{A}$. For each pair $s, s' \in S$ of states, we have $s \approx^p s'$ if and only if $\mathcal{F}^{\bullet}(s) = \mathcal{F}^{\bullet}(s')$.*

Restriction to Rational Timing Bounds. The logic PTLogic features real values in constraints on timing bounds in order to provide a logical characterization of timed bisimulation for all states of a PTA. However, inspired by [13], we note that a version of PTLogic restricted to non-negative rationals $\mathbb{Q}_{\geq 0}$ provides a logical characterization of timed bisimulation for those states of a PTA with rational values of clocks. Let $\mathcal{F}_{\mathbb{Q}_{\geq 0}}$ denote the set of formulas of the logic obtained from PTLogic by restricting formulas of $\langle a, \sim c\rangle\psi$ to the case of $c \in \mathbb{Q}_{\geq 0}$.

Theorem 5. *Let $[\![\mathcal{A}]\!] = (S, \bar{S}, Act, \rightarrow)$ be the semantic PTLTS of the PTA $\mathcal{A}$ with the set $\mathcal{X}$ of clocks. For each pair $(l, v), (l', v') \in S$ of states such that $v(x) \in \mathbb{Q}_{\geq 0}$ and $v'(x) \in \mathbb{Q}_{\geq 0}$ for all clocks $x \in \mathcal{X}$, $\mathcal{F}_{\mathbb{Q}_{\geq 0}}(s) = \mathcal{F}_{\mathbb{Q}_{\geq 0}}(s')$ implies $(l, v) \approx (l', v')$.*

The proof of Theorem 5 follows that of direction ($\Leftarrow$) of Theorem 3, except that, as in [13], and without loss of generality, only transitions with durations taken from $Times(s, s')$ are considered.

The converse of Theorem 5 (that is, that $(l, v) \approx (l', v')$ implies $\mathcal{F}_{\mathbb{Q}_{\geq 0}}(s) = \mathcal{F}_{\mathbb{Q}_{\geq 0}}(s')$) follows trivially from Theorem 3, because $\mathcal{F}_{\mathbb{Q}_{\geq 0}} \subseteq \mathcal{F}$. Theorem 5 can also be extended to the case of probabilistic timed bisimulation.

Note that, to decide $\mathcal{A} \approx \mathcal{B}$, we consider whether the initial states of the PTA are related by $\approx$; as all clocks have to value 0 initially, clearly the above PTLogic with time constraints restricted to $\mathbb{Q}_{\geq 0}$ characterizes bisimulation between PTA. Finally, we observe that formulas of PTLogic with time constraints restricted to $\mathbb{Q}_{\geq 0}$ can be expressed in the timed modal logic of [25,26] extended with the probabilistic operator of $[\psi]_p$. Hence, such a logic can also provide a logical characterization of states with rational clock values.

5 Conclusions

In this paper we have presented a framework for reasoning about simulation and bisimulation relations for PTA. On the one hand, we have presented an EXPTIME algorithm for deciding such relations, and on the other hand we have shown how a timed extension of the probabilistic model logic of [18] provides a logical characterization of bisimulation. To our knowledge a logical characterization of simulation for Segala's probabilistic automata does not yet exist: if such a characterization is found, it is likely that it can be adapted also to the case of PTA. For specifying properties of probabilistic timed automata, temporal logics such as PTCTL [5], which include constraints on time and probability, have been introduced: we note that timed bisimulation preserves PTCTL properties, and that, for a negation-free fragment of PTCTL, a state s that is timed simulated by another state s' satisfies at least the same properties as s' [3,4,27].

For future work, we intend to study weak extensions of the considered relations, which abstract from non-observable computation (see [28]), and to develop quantitative versions of simulation and bisimulation for PTA, which can quantify how closely two PTA resemble each other.

References

1. Alur, R., Dill, D.L.: A theory of timed automata. TCS 126(2), 183–235 (1994)
2. Puterman, M.L.: Markov Decision Processes. J. Wiley & Sons, Chichester (1994)
3. Segala, R., Lynch, N.A.: Probabilistic simulations for probabilistic processes. Nordic Journal of Computing 2(2), 250–273 (1995)
4. Segala, R.: Modeling and Verification of Randomized Distributed Real-Time Systems. PhD thesis, Massachusetts Institute of Technology (1995)
5. Kwiatkowska, M., Norman, G., Segala, R., Sproston, J.: Automatic verification of real-time systems with discrete probability distributions. TCS 286, 101–150 (2002)
6. Kwiatkowska, M., Norman, G., Sproston, J.: Probabilistic model checking of deadline properties in the IEEE 1394 FireWire root contention protocol. Formal Aspects of Computing 14(3), 295–318 (2003)
7. Kwiatkowska, M., Norman, G., Parker, D., Sproston, J.: Performance analysis of probabilistic timed automata using digital clocks. FMSD 29, 33–78 (2006)
8. Milner, R.: An algebraic definition of simulation between programs. In: Proc. IJCAI'71, pp. 481–489. William Kaufmann, San Francisco (1971)

9. Milner, R.: A Calculus of Communication Systems. LNCS, vol. 92. Springer, Heidelberg (1980)
10. Park, D.: Concurrency and automata on infinite sequences. In: Deussen, P. (ed.) GI-TCS 1981. LNCS, vol. 104, pp. 167–183. Springer, Heidelberg (1981)
11. Čerāns, K.: Decidability of bisimulation equivalences for parallel timer processes. In: Probst, D.K., von Bochmann, G. (eds.) CAV 1992. LNCS, vol. 663, pp. 302–315. Springer, Heidelberg (1993)
12. Taşıran, S., Alur, R., Kurshan, R.P., Brayton, R.K.: Verifying abstractions of timed systems. In: Sassone, V., Montanari, U. (eds.) CONCUR 1996. LNCS, vol. 1119, pp. 546–562. Springer, Heidelberg (1996)
13. Bozzelli, L., Legay, A., Pinchinat, S.: On timed alternating simulation for concurrent timed games. In: Proc. FSTTCS'09. LIPIcs, vol. 4, pp. 85–96. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik (2009)
14. Baier, C., Engelen, B., Majster-Cederbaum, M.E.: Deciding bisimilarity and similarity for probabilistic processes. JCSS 60(1), 187–231 (2000)
15. Zhang, L., Hermanns, H., Eisenbrand, F., Jansen, D.N.: Flow faster: Efficient decision algorithms for probabilistic simulations LMCS 4(4) (2008)
16. Holmer, U., Larsen, K.G., Yi, W.: Deciding properties of regular real time processes. In: Larsen, K.G., Skou, A. (eds.) CAV 1991. LNCS, vol. 575, pp. 443–453. Springer, Heidelberg (1992)
17. Hennessy, M., Milner, R.: Algebraic laws for nondeterminism and concurrency. JACM 32(1), 137–161 (1985)
18. Parma, A., Segala, R.: Logical characterizations of bisimulations for discrete probabilistic systems. In: Seidl, H. (ed.) FOSSACS 2007. LNCS, vol. 4423, pp. 287–301. Springer, Heidelberg (2007)
19. Jensen, H.E., Gregersen, H.: Formal design of reliable real time systems. Master's thesis, Aalborg University (1995)
20. Jensen, H.E.: Model checking probabilistic real time systems. In: Proc. of the 7th Nordic Work. on Progr. Theory, pp. 247–261. Chalmers Institute of Technology (1996)
21. Yamane, S.: Probabilistic timed simulation verification and its application to stepwise refinement of real-time systems. In: Saraswat, V.A. (ed.) ASIAN 2003. LNCS, vol. 2896, pp. 276–290. Springer, Heidelberg (2003)
22. Chen, T., Han, T., Katoen, J.P.: Time-abstracting bisimulation for probabilistic timed automata. In: Proc. TASE'08, pp. 177–184. IEEE, Los Alamitos (2008)
23. Jonsson, B., Larsen, K.G.: Specification and refinement of probabilistic processes. In: Proc. LICS'91, pp. 266–277. IEEE, Los Alamitos (1991)
24. Laroussinie, F., Schnoebelen, P.: The state explosion problem from trace to bisimulation equivalence. In: Tiuryn, J. (ed.) FOSSACS 2000. LNCS, vol. 1784, pp. 192–207. Springer, Heidelberg (2000)
25. Laroussinie, F., Larsen, K.G., Weise, C.: From timed automata to logic – and back. In: Hájek, P., Wiedermann, J. (eds.) MFCS 1995. LNCS, vol. 969, pp. 529–539. Springer, Heidelberg (1995)
26. Aceto, L., Laroussinie, F.: Is your model checker on time? On the complexity of model checking for timed modal logics. JLAP 52-53, 7–51 (2000)
27. Sproston, J.: Model checking for probabilistic timed and hybrid systems. PhD thesis, University of Birmingham (2000)
28. Lanotte, R., Maggiolo-Schettini, A., Troina, A.: Weak bisimulation for probabilistic timed automata and applications to security. In: Proc. SEFM'03, pp. 34–43. IEEE, Los Alamitos (2003)

Layered Composition for Timed Automata*

Ernst-Rüdiger Olderog and Mani Swaminathan

Department of Computing Science
University of Oldenburg, Germany
{olderog,mani.swaminathan}@informatik.uni-oldenburg.de

Abstract. We investigate layered composition for real-time systems modelled as (networks of) timed automata (TA). We first formulate the principles of layering and transition independence for TA, and demonstrate the validity of the communication closed layer (CCL) laws in such a setting, by means of an operator for layered composition that is intermediate between parallel and sequential composition. Next, we introduce the principles of input/output (i/o) and partial-order (po) equivalences, and show that such equivalences are preserved when the layered composition operator is replaced by sequential composition within the expressions appearing in the CCL laws. Finally, we proceed to show that such layering (together with equivalences obtained through the CCL laws) can be useful in the design and verification of dense real-time systems that consist of a network of interacting components, by bringing about a reduction of the state-space through the exploitation of transition independence. This is illustrated by considering a collision avoidance protocol developed for an audio/video system of Bang and Olufsen.

1 Introduction and Related Work

Real-time systems have strict timing requirements and are used within many safety critical applications. Such systems are becoming increasingly complex, and often consist of multiple parallel interacting components, with the interaction taking place by means of shared variables or by message passing along common channels. Reasoning about such systems is much easier when the execution of the components is viewed *sequentially*, as opposed to corresponding *distributed or concurrent representations*. This is because the *physical structure* of the system is often in the form of multiple *parallel* interacting components, each executing a (sequential) program, while the system's *logical structure* is in the form of a complex protocol consisting of a sequence of *layers*, wherein each layer consists of many actions distributed across the whole system [1].

Such a distinction between the physical and logical structures of a distributed system has motivated research into techniques such as *communication closedness* [2] for transforming concurrent/distributed representations of the system into

* This work has been partially funded 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, www.avacs.org).

K. Chatterjee and T.A. Henzinger (Eds.): FORMATS 2010, LNCS 6246, pp. 228–242, 2010.

appropriate (and equivalent) *layered* ones, in order to enable easier reasoning about system properties.

Layering for real-time systems was first investigated in [3], where systems are expressed in a process language that allows for the specification of events of a (sub-)process within a given time interval (expressed by means of a timing operator), with interaction among the (sub-)processes being via shared variables. The timing operator is used here to enforce a temporal order between events, which is then exploited in the layer transformation rules. A process algebra with a new operator for *layered composition* (intermediate between sequential and parallel composition) based on hierarchical graphs is presented in [4]. This layered composition operator is then used to formalize *equivalences* between distributed and layered representations of a system, by exploiting *independence* between events across multiple system components, through the *communication closed layer (CCL) laws*. Section 9.1 of [4] discusses real-time extensions of the process algebra, with real-time behaviour being embedded into the algebra by means of explicit constructs (such as a delay construct) that model the passage of time. Such real-time extensions to the process algebra are elaborated further in [1], wherein an assertional proof system is presented that is then used for checking real-time side conditions for (extended) CCL laws.

However, the assertional proof system presented in the above works is not amenable to efficient and automatic tool-based verification. There has since been extensive study of the formalism of *timed automata (TA)* as an effective means for the modelling and (automatic) verification of real-time systems. TA [5] extend (finite) automata by augmenting them with non-negative real-valued clocks, in order to quantitatively capture the timing behaviour of the system. The TA model has been shown to be very amenable to automatic verification of large real-time systems, with TA model checkers such as UPPAAL [6] and KRONOS [7] having been successfully used in industrial case studies (such as [8]). We therefore propose here to investigate notions of layering and communication closedness in the context of TA. Our approach thus differs from that of previous related works on layering (particularly [1,4]) in the following aspects:

- Previous works on layering present assertional proof systems for validating (real-time) side conditions for the applicability of the CCL laws for process algebras based on hierarchical graphs, under notions of process independence, and the techniques therein are not amenable to efficient and automatic verification. Our approach however investigates the conditions for applicability of the CCL laws on timed automata based models that come equipped with extensive tool-support for automatic verification.
- The layered composition operator used in all cases relies crucially on notions of *independence* between actions, transitions, and processes across a system. Previous works (particularly [1,4]) rely on explicitly postulated notions of (in-)dependence, based on *dependency graphs*. Our work additionally uses a semantic notion of transition independence (involving the conditions of *enabledness* and *commutativity*), similar to the ones that have been extensively

studied in the context of *partial order reduction* approaches [9] (for tackling the state-space explosion problem in model checking).
- The approach in previous works on layering is mainly motivated by a design process that starts with a sequential algorithm (corresponding to the system's logical structure) and then gradually transforms it into a distributed version (corresponding to the system's actual implementation), by successively applying the layering operator. The approach in our work, on the other hand, is motivated by the necessity of transforming a distributed representation of a system into an equivalent layered one, in order to permit easier verification of system properties. Such easier verification is made possible by means of the notion of *input/output (i/o) and partial order (po) equivalences* between the sequential and layered compositions of processes [4]. We establish in this paper the validity of such i/o and po equivalences also between sequential and layered compositions of timed automata, and exploit their use in reducing the state space of large real time systems for easier verification.

Partial order reduction for timed automata has been investigated in many papers [10,11,12,13,14,15,16,17], with a view towards reducing the system's state space by exploiting transition independence, with efficient symbolic techniques being integrated within the UPPAAL model checker, by means of the UPPAAL PORT extension [16]. Our goal here is to exploit the notions of transition independence used within such partial order reduction techniques for timed automata, in order to bring about a further reduction of a large system's state space by transforming its distributed representation into an equivalent layered version, by investigating conditions for the applicability of the CCL laws. It should be noted here that our layering approach is *complementary* to partial order reduction based approaches, while exploiting similar (in-)dependence notions at different levels. More specifically, our contributions here are the following:

1. We first formulate the principles of layering and communication closedness for (networks of) timed automata, by exploiting notions of transition independence in order to formulate the CCL laws. We then proceed to show the validity of such CCL laws in the absence of "cross-dependencies" between system components, corresponding to [1,4].
2. Next, we formulate the notions of i/o and po equivalences, and show that these equivalences are preserved when one replaces layered composition by sequential composition within the expressions in the CCL laws.
3. We then proceed to show that such layering can be useful in the design and verification of dense real-time systems that consist of a network of interacting components, by exploiting (in-)dependencies in order to bring about a reduction of the state-space, through the CCL laws, together with i/o and po equivalences. Such a reduction is shown to complement partial order reduction approaches applied to (parallel compositions of) timed automata. This is illustrated by considering a collision avoidance protocol that was developed for an audio/video system of Bang and Olufsen [8].

The rest of the paper is structured as follows. Section 2 reviews timed automata and their semantics. Section 3 discusses composition technqiues for

timed automata, and introduces our operator $\bullet$ for layered composition, and the consequent CCL laws. Section 4 introduces the notions of i/o and po equivalences, and demonstrates their validity in the context of the CCL laws. Extension of these results to TA enriched with data (widely used within the model checker UPPAAL) and a methodology of using these results for state-space reduction is also discussed. Section 5 provides preliminary ideas on the application of our results by considering a collision avoidance protocol that was developed for an audio/video system of Bang and Olufsen [8]. Section 6 concludes the paper along with directions for future research.

2 Preliminaries

Given a finite set C of *clocks*, a *clock valuation* over C is a map $v : C \to \mathbb{R}_{\geq 0}$ that assigns a non-negative real value to each clock in C. If $|C| = n$, a clock valuation is identified with a point in $\mathbb{R}^n_{\geq 0}$, which we henceforth denote by $\vec{u}, \vec{v}, \vec{x}, \vec{y}$ etc. By $\vec{0}$ we denote the clock valuation where all clocks are set to 0.

A *zone* over a set of clocks C is a constraint defined by the following grammar $g ::= x \triangleright d \mid g \wedge g$, where $x \in C$, $d \in \mathbb{N}$, and $\triangleright \in \{<, \leq, >, \geq\}$. The set of zones over C is denoted $Z(C)$. The subset of zones having only upper bounds $<, \leq$ is denoted by $Z_U(C)$. In the sequel we shall identify zones with the set of clock valuations satisfying them, so that set-theoretic operations may be applied on zones.

Definition 1 (Timed automaton). *A* timed automaton *(TA) is a tuple* $A = (L, \Sigma, C, l_0, l_F, Inv, E)$*, where*

- L *is a finite set of* locations*,* Σ *a finite alphabet, and* C *a finite set of* clocks*,*
- $l_0 \in L$ *is the* initial *location, and* $l_F \in L$ *the* final *location, with* $l_0 \neq l_F$*,*
- $Inv : L \to Z_U(C)$ *assigns a* clock invariant *to each location,*
- $E \subseteq L \times \Sigma \times Z(C) \times 2^C \times L$ *is a finite set of directed* edges *between locations. An edge* $e = (l, a, g, r, l')$ *from* l *to* l' *involves an* action $a \in \Sigma$*, a* guard $g \in Z(C)$*, and a* reset set $r \subseteq C$*.*

For a clock valuation $\vec{x}$, its *time-passage* is $timepass(\vec{x}) = \{\vec{x} + d \mid d \geq 0\}$, where $\vec{x} + d$ denotes the addition of a scalar $d \in \mathbb{R}_{\geq 0}$ to each component of $\vec{x}$. The *k-region-equivalence relation* $\approx_k$ on clock valuations $\vec{x}$ and $\vec{y}$ is defined by

$$\vec{x} \approx_k \vec{y} \text{ iff } \forall i \leq n : \left(\begin{array}{l} (x_i > k) \wedge (y_i > k) \\ \vee\ (int(x_i) = int(y_i) \wedge (fr(x_i) = 0 \Leftrightarrow fr(y_i) = 0) \wedge \\ \quad \forall j \leq n : (fr(x_i) \leq fr(x_j) \Leftrightarrow fr(y_i) \leq fr(y_j))\ ) \end{array} \right),$$

where, for a clock valuation $\vec{x} \in \mathbb{R}^n_{\geq 0}$, x_i denotes its i-th component, i.e., the value of the i-th clock, and $int(x_i)$ and $fr(x_i)$ denote the integer and fractional parts of x_i, respectively. By $[\vec{x}]_k$ we denote the *k-region* containing $\vec{x}$, which is the equivalence class induced by $\approx_k$.

The semantics of a TA is given in terms of its underlying *timed transition system*, which consists of an infinite set of states of the form $(l, \vec{x})$, where $l \in L$ and

$\vec{x} \in \mathbb{R}^n_{\geq 0}$, with the transitions between states resulting in the formation of *canonical paths* through the transition system.

Definition 2 (Canonical path). *A* canonical path π *through such a timed transition system is a (possibly infinite) sequence* $\langle (l_0, \vec{x}_0), (l_1, \vec{x}_1), \ldots \rangle$ *of states, subject to the following* initiation *and* consecution *conditions:*

1. *Initiation:* l_0 *is the initial location and* $\vec{x}_0 = \vec{0}$.
2. *Consecution (time-passage): for even* i *we require* $l_{i+1} = l_i$ *and* $\vec{x}_{i+1} \in Inv(l_i) \cap timepass(\vec{x}_i)$.
3. *Consecution (edges): for odd* i *we require* $\exists\, e = (l_i, a, g, r, l_{i+1}) \in E :$ $\vec{x}_i \in Inv(l_i) \cap g$ *and* $\vec{x}_{i+1} \in Inv(l_{i+1}) \cap r(\vec{x}_i)$.[1]

Let Π denote the set of all such canonical paths. A *transition* between two consecutive states in such a path thus corresponds either to time-passage within a location, or to an edge-traversal between discrete locations. The reachable state space of the TA is then given by the set of states that are reachable from the initial state, through the transitions of all possible canonical paths, and is defined as follows.

Definition 3 (Reachable state space). $Reach(A) \subseteq L \times (C \to \mathbb{R}_{\geq 0})$ *is the* reachable state space *of a TA* A*, consisting of an infinite set of states of the form* $(l, \vec{x})$*, where* $l \in L$ *and* $\vec{x} \in \mathbb{R}^n_{\geq 0}$*. It is defined inductively as follows, with* $Reach_i(A)$ *denoting the reach-set under* $i \in \mathbb{N}$ *steps, starting from the initial state* $(l_0, \vec{0})$ *and alternating between time-passage and discrete transitions:*

- $Reach_0(A) = \left\{ (l_0, \vec{0}) \right\}$,
- $Reach_{i+1}(A) = Reach_i(A) \cup Succ(Reach_i(A))$, *where*

  if $i \geq 0$ *even,* $Succ(Reach_i(A)) = \left\{ (l, \vec{x}) \;\middle|\; \begin{array}{l} \exists\, \vec{u} \in Inv(l) : (l, \vec{u}) \in Reach_i(A) \\ \wedge\ \ \vec{x} \in timepass(\vec{u}) \cap Inv(l) \end{array} \right\}$

  if $i \geq 0$ *odd,* $Succ(Reach_i(A)) = \left\{ (l, \vec{x}) \;\middle|\; \begin{array}{l} \exists\, e = (l', a, g, r, l) \in E \\ \exists\, \vec{u} \in Inv(l') \cap g : \\ \quad (l', \vec{u}) \in Reach_i(A) \\ \wedge\ \ \vec{x} \in Inv(l) \cap r(\vec{u}) \end{array} \right\}$
- $Reach(A) = \bigcup_{i \in \mathbb{N}} Reach_i(A)$.

This leads to the following notion of *reachability equivalence* denoted by $\equiv$. Given two TA A_1 and A_2, we define

$$A_1 \equiv A_2 \quad \text{iff} \quad \forall i \in \mathbb{N} : Reach_i(A_1) = Reach_i(A_2).$$

Thus the equivalence requires equal sets of reachable states after every iteration of the transition relation.

[1] To simplify subsequent proofs, we use even- and odd-numbered steps to distinguish between time-passage and taking edges between discrete locations. Here $r(\vec{x})$ denotes the valuation obtained from $\vec{x}$ after resetting all the clocks in r.

3 Layered Composition and CCL Laws

We have thus far considered the semantics of timed automata that operate in isolation. However, real-time systems in practice *communicate* with each other and with the environment, and this results in a *composite* system consisting of communicating components. The communication between components is often through synchronization actions drawn from a shared alphabet. We now define three operators for constructing such composite systems: *sequential*, *parallel*, and *layered* composition, where the latter is new for timed automata.

In the sequel we consider timed automata $A_i = (L_i, C_i, l_{0i}, l_{Fi}, Inv_i, E_i, \Sigma_i)$, $i = 1, 2$, with disjoint locations and clocks: $L_1 \cap L_2 = C_1 \cap C_2 = \emptyset$.

Definition 4 (Sequential composition). *Let $l_{01} \neq l_{F1}$ and $l_{02} \neq l_{F2}$. Then the* sequential composition *of A_1 and A_2 is defined as the timed automaton*

$$A_1; A_2 = (L_1 \cup L_2 \cup \{\widetilde{l_{F1}}\}, \Sigma_1 \cup \Sigma_2, C_1 \cup C_2, l_{01}, l_{F2}, Inv_1 \cup Inv_2, E),$$

where $\widetilde{l_{F1}}$ is a copy of l_{F1} disjoint from $L_1 \cup L_2$, with $Inv(\widetilde{l_{F1}}) = Inv(l_{F1}) \wedge Inv(l_{02})$, and E is given by:

$$\begin{aligned} E = {} & (E_1 \setminus \{(l_1, a_1, g_1, r_1, l_{F1}) \mid (l_1, a_1, g_1, r_1, l_{F1}) \in E_1\}) \\ & \cup \{(l_1, a_1, g_1, r_1, \widetilde{l_{F1}}) \mid (l_1, a_1, g_1, r_1, l_{F1}) \in E_1\} \\ & \cup (E_2 \setminus \{(l_{02}, a_2, g_2, r_2, l_2) \mid l_{02}, a_2, g_2, r_2, l_2) \in E_2\}) \\ & \cup \{(\widetilde{l_{F1}}, a_2, g_2, r_2, l_2) \mid (l_{02}, a_2, g_2, r_2, l_2) \in E_2\}. \end{aligned}$$

Thus $A_1; A_2$ is obtained by first performing the actions in A_1 and then performing the actions in A_2. The final location l_{F1} of A_1 and the initial location l_{02} of A_2 are amalgamated to a new location $\widetilde{l_{F1}}$. It is assumed here that invariants of l_{F1} and l_{02} are mutually consistent[2]. This definition is adapted from [18,19].

Definition 5 (Parallel composition). *The* parallel composition *is defined by*

$$A_1 \| A_2 = (L_1 \times L_2, \Sigma_1 \cup \Sigma_2, C_1 \cup C_2, (l_{01}, l_{02}), (l_{F1}, l_{F2}), Inv, E),$$

where $\forall\, (l_1, l_2) \in L_1 \times L_2 : Inv(l_1, l_2) = Inv_1(l_1) \wedge Inv_2(l_2)$ and E given by:

- *Synchronization: If $e_i = (l_i, a_i, g_i, r_i, l_i') \in E_i$ for $i = 1, 2$ with $a_1 = a_2$ then $((l_1, l_2), a_1, g_1 \wedge g_2, r_1 \cup r_2, (l_1', l_2')) \in E$.*
- *Interleaving: (1) If $e_1 = (l_1, a_1, g_1, r_1, l_1') \in E_1$ with $a_1 \notin \Sigma_2$ then $\forall\, l_2 \in L_2 : ((l_1, l_2), a_1, g_1, r_1, (l_1', l_2)) \in E$.*
  (2) Conversely, if $e_2 = (l_2, a_2, g_2, r_2, l_2') \in E_2$ with $a_2 \notin \Sigma_1$ then $\forall l_1 \in L_1 : ((l_1, l_2), a_2, g_2, r_2, (l_1, l_2')) \in E$.

[2] Our example in Section 5 satisfies this condition.

Note that parallel composition is *symmetric*, it involves a CSP-style *synchronization* on common actions and *interleaving* on disjoint actions. This composition does not respect the *dependencies* between the individual components[3]. As mentioned in the introduction, a real-time distributed system often consists of (sequential) phases that execute in parallel on multiple platforms, wherein an action (resp. transition) within a given phase can execute only after all *dependent* actions (resp. transitions) in each preceding phase have been executed.

Motivated by this, we now proceed to define *layered composition* between two TA. As in [1,4], we assume in this definition an explicitly postulated notion of *dependencies* between the *actions* of the timed automata that constitute the layered composition, as part of the overall system specification. The edges of the timed automata are then dependent iff they execute dependent actions.

Actions/edges that are not dependent are termed *independent*. Two independent actions need to satisfy the conditions of *enabledness* and *commutativity*. Enabledness here implies that they do not disable each other, while commutativity implies that they can be executed in either order starting from a given input state, and yet result in the same (or an "equivalent") output state. We refer to [10,16] for formal definitions of independent actions in timed automata. The dependency between two actions a and b is denoted $a \leftrightsquigarrow b$, their independence by $a \not\leftrightsquigarrow b$. We stipulate that the dependency relation $\leftrightsquigarrow$ is reflexive. Two TA A_1 and A_2 are said to be *independent*, denoted $A_1 \not\leftrightsquigarrow A_2$, iff every action of A_1 is independent of every action of A_2.

Definition 6 (Layered composition). *The* layered composition *is defined by*

$$A_1 \bullet A_2 = (L_1 \times L_2, \Sigma_1 \cup \Sigma_2, C_1 \cup C_2, (l_{01}, l_{02}), (l_{F1}, l_{F2}), Inv, E),$$

where Inv and E are as in the parallel composition $A_1 \| A_2$, except that part (2) of the interleaving case is now different:

- *Synchronization: If $e_i = (l_i, a_i, g_i, r_i, l_i') \in E_i$ for $i = 1, 2$ with $a_1 = a_2$ then $((l_1, l_2), a_1, g_1 \wedge g_2, r_1 \cup r_2, (l_1', l_2')) \in E$.*
- *Interleaving: (1) If $e_1 = (l_1, a_1, g_1, r_1, l_1') \in E_1$ with $a_1 \notin \Sigma_2$ then $\forall\, l_2 \in L_2 : ((l_1, l_2), a_1, g_1, r_1, (l_1', l_2)) \in E$.*
  (2) If $e_2 = (l_2, a_2, g_2, r_2, l_2') \in E_2$ with $a_2 \notin \Sigma_1$ and $\forall l_1, l_1^* \in L_1 : l_1 \xrightarrow{*} l_1^* \;\; \forall e_1 = ({l_1}^*, a_1, g_1, r_1, l_1') \in E_1 : a_1 \not\leftrightsquigarrow a_2$, then $((l_1, l_2), a_2, g_2, r_2, (l_1, l_2')) \in E$,

where $l_1 \xrightarrow{*} l_1^*$ expresses that l_1^* is reachable from l_1 in the syntactic structure of A_1 through an arbitrary sequence of edges.

Thus only part (2) of the interleaving case differs from parallel composition: an interleaving edge of the second automaton A_2 is allowed to execute only after all

[3] We assume *local time semantics* as in [10,16] to obtain fewer dependencies induced by timing.

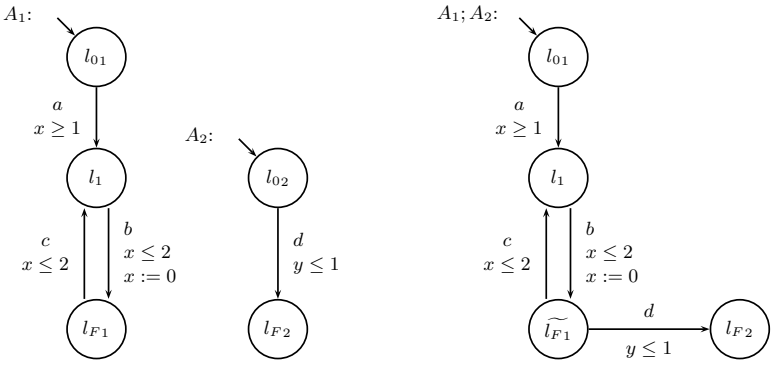


Fig. 1. *Left*: timed automata A_1 and A_2 with the stipulated dependency $a \rightsquigarrow d$; *right*: sequential composition $A_1; A_2$

dependent edges of the first automaton A_1 have been executed. Figures 1 and 2 illustrate the three composition oparators for two simple timed automata.

We now proceed to formulate and validate the CCL laws (equivalences) of [1,4] for layered composition of timed automata.

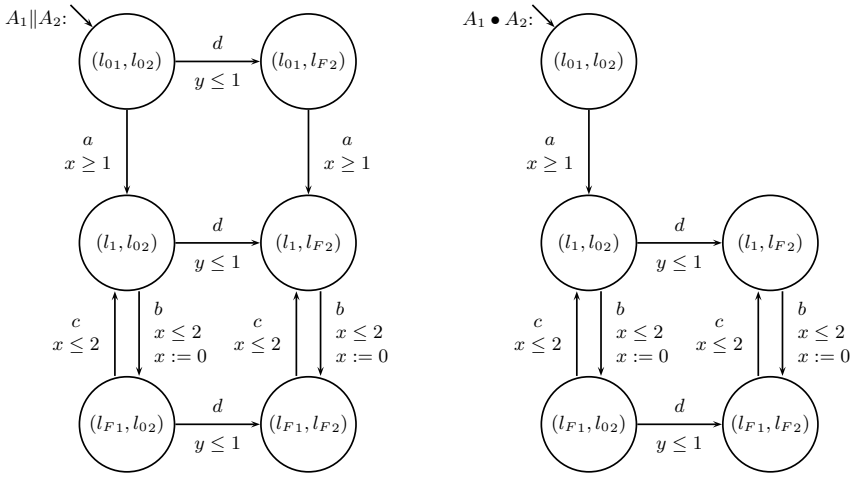


Fig. 2. *Left*: parallel composition $A_1 \| A_2$; *right*: layered composition $A_1 \bullet A_2$

Theorem 1 (CCL laws for timed automata). *For all timed automata A_1, A_2, B_1, and B_2, with $A_1 \leftrightarrow\!\!\!\!\!\!\rightarrow B_2$ and $A_2 \leftrightarrow\!\!\!\!\!\!\rightarrow B_1$, the following* communication closed layer *equivalences (CCL laws) hold for the reachability equivalence $\equiv$:*

1. $A_1 \bullet B_2 \;\equiv\; A_1 \| B_2$ (Indep)
2. $(A_1 \bullet A_2) \| B_2 \;\equiv\; A_1 \bullet (A_2 \| B_2)$ (CCL-L)

3. $(A_1 \bullet A_2) \| B_1 \equiv (A_1 \| B_1) \bullet A_2$ (CCL-R)
4. $(A_1 \bullet A_2) \| (B_1 \bullet B_2) \equiv (A_1 \| B_1) \bullet (A_2 \| B_2)$ (CCL)

Proof. Owing to space limitations, we provide below only a brief sketch of the proof of the law CCL-L. The proofs of the other CCL laws are similar.
Given TA A_1, A_2, B_2, with $A_1 \leftrightarrow B_2$, we show $(A_1 \bullet A_2) \| B_2 \equiv A_1 \bullet (A_2 \| B_2)$, i.e., the sets of reachable states are equal at every iteration i of their transition relations.[4] The proof is by induction over i.

The containment $Reach_i(A_1 \bullet (A_2 \| B_2)) \subseteq Reach_i((A_1 \bullet A_2) \| B_2)$ is not hard to see intuitively, as the parallel composition operator dominates on the right-hand side and the layered composition operator on the left-hand side. The dominance of layered composition induces fewer interleavings on the basis of the respective dependencies, as seen from the definition earlier.

We show $Reach_i((A_1 \bullet A_2) \| B_2) \subseteq Reach_i(A_1 \bullet (A_2 \| B_2))$ by induction over i.
Induction Basis. This case $i = 0$ is obvious.
Assume that the containment holds for some i.

Induction Step. Consider $((l_{A1}, l_{A2}, l_{B2}), (\vec{u}, \vec{v}, \vec{w})) \in Reach_{i+1}((A_1 \bullet A_2) \| B_2)$. For the case where $((l_{A1}, l_{A2}, l_{B2}), (\vec{u}, \vec{v}, \vec{w})) \in Reach_i((A_1 \bullet A_2) \| B_2)$ the proof is again immediate from the induction hypothesis. We now examine the cases where $((l_{A1}, l_{A2}, l_{B2}), (\vec{u}, \vec{v}, \vec{w})) \in Succ(Reach_i((A_1 \bullet A_2) \| B_2))$.

If i is even, the preceding transition corresponds to time-passage, which is possible also in $A_1 \bullet (A_2 \| B_2)$, and the proof then follows immediately.

If i is odd, using the fact that A_1 and B_2 are independent (thereby ruling out synchronization between A_1 and B_2), we see that the preceding transition corresponds to an action which could have been performed (a) either by A_1, A_2, B_2 individually, (b) or as a synchronization action involving A_1 and A_2, (c) or as a synchronization action involving A_2 and B_2.

The cases corresponding to the preceding transition having been executed individually either by A_1 or by A_2 are relatively straightforward, as are the cases (b) and (c). We now consider the case where the preceding transition corresponds to an action performed by B_2 alone. This means that there exist $((l_{A1}, l_{A2}, l'_{B2}), (\vec{u'}, \vec{v'}, \vec{w'})) \in Reach_i((A_1 \bullet A_2) \| B_2)$ and $e = (l'_{B2}, a, g, r, l_{B2}) \in E_{B_2}$ such that $\vec{u'} \in Inv(l_{A1}) \cap g$, $\vec{v'} \in Inv(l_{A2}) \cap g$, $\vec{w'} \in Inv(l_{B2}) \cap g$, with $\vec{u} = r(\vec{u'})$, $\vec{v} = r(\vec{v'})$, $\vec{w} = r(\vec{w'})$. The proof is then immediate from the induction hypothesis, and using the fact that A_1 and B_2 are independent. □

4 Equivalences

We now formalize the notions of *input/output (i/o)* and *partial order (po) equivalences* as a means of relating sequential and layered composition. In particular, we show that, for two timed automata A_1 and A_2, it is *always* the case that $A_1; A_2$ and $A_1 \bullet A_2$ are i/o and po equivalent, irrespective of any (in-) dependence relation between the actions of A_1 and A_2.

[4] We identify nested pairs $((x, y), z)$ and $(x, (y, z))$ of locations with tuples (x, y, z).

We elaborate on these equivalences, through the following definitions and lemmas.

Definition 7 (i/o equivalence of paths). *Given two TA A_1 and A_2 with Π_1 and Π_2 denoting the corresponding sets of finite, canonical paths ending in l_{F1} and l_{F2}, respectively, and given a relation $\approx$ relating the locations of A_1 and A_2, a path $\pi_1 \in \Pi_1$ is* i/o equivalent *to a path $\pi_2 \in \Pi_2$ (relative to $\approx$), denoted $\pi_1 \equiv_{i/o} \pi_2$, where $\pi_i = \langle (l_{0i}, \vec{0}), \ldots (l_{Fi}, \vec{x}_i) \rangle, i = 1, 2$, iff $l_{01} \approx l_{02}$, $l_{F1} \approx l_{F2}$, and $\vec{x}_1 \approx_k \vec{x}_2$, where k is the maximum of all constants in A_1 and A_2*

Definition 8 (Layered normal form). *A (finite) canonical path π of $A_1 \bullet A_2$ is in* layered normal form (LNF) *if it consists of consecutive transitions from E_1 passing through l_{F1}, followed by consecutive transitions from E_2 ending in l_{F2} (i.e., no transition from E_2 precedes a transition from E_1).*

Definition 9 (po equivalence of paths). *Let A_1 and A_2 be two TA sharing a common alphabet Σ, with Π_1 and Π_2 denoting the corresponding sets of finite, canonical paths. Let $\approx$ be a relation between the locations of A_1 and A_2. A path $\pi_1 \in \Pi_1$ is* po equivalent *to $\pi_2 \in \Pi_2$, denoted $\pi_1 \equiv_{po} \pi_2$, relative to $\approx$ on the corresponding locations, and region-equivalence on the corresponding clock-valuations (w.r.t the maximum constant of A_1 and A_2) if π_i can be obtained from π_{3-i} by repeated permutation of adjacent independent transitions separated by only one time-passage.*

Thus, two po equivalent paths π_1 and π_2 (relative to region-equivalence on their clock valuations and $\approx$ on their locations) differ only in the (permutative) ordering of *independent* transitions. This definition has been adapted for TA from [20].

Lemma 1. *Let A_1 and A_2 be two TA, let Π denote the set of all finite, canonical paths of $A_1 \bullet A_2$, and $\Pi_L \subseteq \Pi$ the subset of these paths that are in LNF. It then holds that $\forall \pi \in \Pi \;\; \exists \pi' \in \Pi_L : \pi \equiv_{i/o} \pi' \;\wedge\; \pi \equiv_{po} \pi'$.*

The proof follows from the definitions of layered composition of TA, and of i/o and partial order equivalences between paths of a TA. Thus, every path of a layered composition can be rewritten into an i/o and po equivalent path that is in layered normal form.

The notions of i/o and po equivalence are then lifted to TA as follows:

Definition 10 (i/o and po equivalence of TA). *Let A_1 and A_2 be two TA sharing a common alphabet Σ, with Π_1 and Π_2 denoting the corresponding sets of finite, canonical paths that end in their respective final states. Then A_1 and A_2 are* i/o (resp. po) equivalent, *denoted $A_1 \equiv_{i/o} A_2$ (resp. $A_1 \equiv_{po} A_2$), iff $\forall \pi_1 \in \Pi_1 \;\; \exists \pi_2 \in \Pi_2 : \pi_1 \equiv_{i/o} \pi_2$ (resp. $\pi_1 \equiv_{po} \pi_2$), and conversely, $\forall \pi_2 \in \Pi_2 \;\; \exists \pi_1 \in \Pi_1 : \pi_1 \equiv_{i/o} \pi_2$ (resp. $\pi_1 \equiv_{po} \pi_2$).*

We then have the following theorem that establishes the i/o and po equivalence between sequential and layered compositions of timed automata

Theorem 2. *For any two TA A_1 and A_2 we have $A_1 \bullet A_2 \equiv_{i/o} A_1; A_2$ and $A_1 \bullet A_2 \equiv_{po} A_1; A_2$.*

Proof. We provide a brief sketch of the proof here owing to space limitations.

We first introduce a relation $\succcurlyeq$ that relates locations of $A_1 \bullet A_2$ with those of $A_1; A_2$. $\forall l_1 \in L_1$ with $l_1 \neq l_{F1} : (l_1, l_{02}) \succcurlyeq l_1$, $\forall l_2 \in L_2$ with $l_2 \neq l_{02} : (l_{F1}, l_2) \succcurlyeq l_2$, and $(l_{F1}, l_{02}) \succcurlyeq \widetilde{l_{F1}}$ (where $\widetilde{l_{F1}}$ is as defined for sequential composition).

Let $\Pi^{;}$ (resp. $\Pi^{\bullet}$) be the set of all finite, canonical paths of $A_1; A_2$ (resp. $A_1 \bullet A_2$) ending in l_{F2}. Then

- $\forall \pi \in \Pi^{;} \quad \exists \pi' \in \Pi^{\bullet} : \pi' \equiv_{i/o} \pi \ \wedge \ \pi' \equiv_{po} \pi$, such that π' is in LNF.
- $\forall \pi \in \Pi^{\bullet} \quad \exists \pi' \in \Pi^{;} : \pi \equiv_{i/o} \pi' \ \wedge \ \pi \equiv_{po} \pi'$, where π is either in LNF, or is i/o and po equivalent to another path in $\Pi^{\bullet}$ that is in LNF (cf. Lemma 1).

The i/o and po equivalence between π and π' above is relative to $\succcurlyeq$ between their respective locations, and to region equivalence (w.r.t the maximum of all constants of A_1 and A_2) between the clock valuations. The i/o and po equivalence between $A_1; A_2$ and $A_1 \bullet A_2$ then follows as an immediate consequence. □

Corollary 1. *Replacing $\bullet$ by ; within the expressions appearing in Theorem 1 yields i/o and partial order equivalences.*

Proof. The proof follows from Theorems 1 and 2. □

A consequence of Corollary 1 is the preservation of (timed) LTL and CTL properties without the *next* operator (see [12] and Chapter 8 of [21]).

Extensions to TA with Data. We have thus far considered (simple) TA that communicate by means of sychronization actions drawn from a shared alphabet, and an explicitly postulated notion of a dependency relation between actions. TA models used within model checkers such as UPPAAL are often extended with *data variables* that range over finite subsets of integers. We do not provide a formal definition of such extended TA, but instead refer the reader to Section 4.4 of [22] for the details concerning their syntax and semantics.

For such extended TA, one may now *infer* the dependencies from the syntactic structure of the given TA, based on the *Read* and $Write$ sets associated with the actions. Two actions are dependent in such a setting if one of the two writes a variable that is read or written by the other action (see Section 4 of [1] for formal details).

The complementary *independence* relation then respects the commutativity condition necessary for partial order reduction. I/O and po equivalence for paths of such extended TA are defined as earlier (i.e., relative to a $\approx$-relation on their locations and appropriate region-equivalence on their clock valuations), together with *identity on their data valuations*.

Theorems 1 and 2 and Corollary 1 then carry over to extended TA under such modified notions of i/o and po equivalences.

Methodology. We now outline the intended methodology on a schematic example. Suppose we want to verify for TA A_1, A_2, B_1 that the system $(A_1; A_2) \| B_1$ satisfies a temporal formula φ without the *next* operator. If $A_2 \leftrightarrow B_1$ we can reduce the state space of the TA system by applying to it the equivalences stated above, as shown on the right-hand side. Thus it suffices to check that $(A_1 \| B_1); A_1$ satisfies φ. If each of A_1, A_2, B_1 has 10 locations then original TA system has 200 locations whereas the transformed system has 110 locations.

$$
\begin{array}{ll}
 & (A_1; A_2) \| B_1 \\
\equiv_{po} & \{ \text{ Corollary 1 } \} \\
 & (A_1 \bullet A_2) \| B_1 \\
\equiv & \{ \text{ CCL-R } \} \\
 & (A_1 \| B_1) \bullet A_2 \\
\equiv_{po} & \{ \text{ Corollary 1 } \} \\
 & (A_1 \| B_1); A_2.
\end{array}
$$

We proceed to apply such a methodology for easier automatic verification of a large system consisting of a network of data-enriched TA.

5 Example: Audio/Video Collision Avoidance Protocol

We present in this section preliminary ideas on the application of the techniques discussed so far (in particular, the CCL laws and the corresponding i/o and po equivalences) towards easier automatic verification of large real-time systems modelled as networks of timed automata. For this purpose, we consider a collision avoidance protocol that was developed for an audio/video system of Bang and Olufsen, whose formal modelling and analysis is considered in detail in [8], using the UPPAAL tool for modelling the protocol as a network of timed automata. The treatment of the protocol in this section is brief, and is only intended to illustrate a possible application of the layering and partial order equivalences discussed hitherto for easier verification of networks of TA.

The UPPAAL model of the collision avoidance protocol presented in [8] consists of a network of nine timed automata communicating in parallel. The protocol schematic is described in Figure 3. This is a simplified version of the schematic found in [8], omitting the names of shared variables and channels. The schematic essentially describes the communication between two (sender) systems A and B that send data frames via a shared *Bus*.[5]. Each (sender) system consists of a corresponding Sender (S_A, S_B), Detector (Det_A, Det_B), Frame Generator (FG_A, FG_B), and Observer (Obs_A, Obs_B).

The protocol is given by the Sender and Detector, where the Sender transmits frames over the shared bus, while the Detector performs collision detection. The Frame Generator and Observer in a sense constitute the *environment* in which the protocol operates. The protocol together with its environment is then represented by a parallel composition of nine (data-enriched) timed automata:

$$System = S_A \| Obs_A \| Det_A \| FG_A \| Bus \| S_B \| Obs_B \| Det_B \| FG_B.$$

This system is required to satisfy the following informal correctness properties

1. Any frame transmitted by a Sender X (where X is A or B) that is destroyed due to collision is (eventually) detected by X.
2. Collision detection must be simultaneous across all senders.

[5] Receiver systems are not relevant for the analysis.

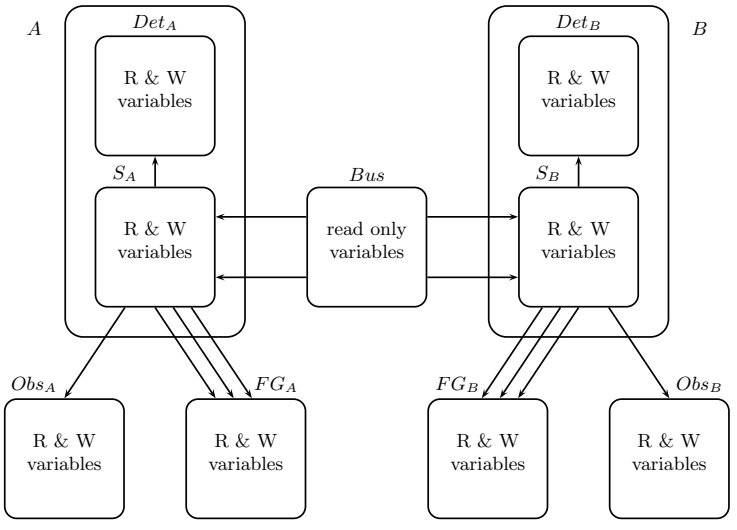


Fig. 3. Audio/video protocol as a network of timed automata, adapted from [8]

These properties may be formally expressed as a CTL formula (without the *next* operator) involving conditions on the shared data variables (see [8]).

Based on the detailed timed automata models given in [8], we find that each Sender System comprises 19 discrete locations, each Detector 8 locations, each Frame Generator 7 locations, each Observer 1 location, while the Bus contains 2 discrete locations. Thus the total number of discrete locations in the parallel composition is $19^2 \times 8^2 \times 7^2 \times 1^2 \times 2$ (i.e., over 2 million discrete locations). The timing behaviour of the system is governed by a single clock A_c per sender that runs locally, consistent with the *local time semantics* introduced in [10].

We however note that the execution of each Sender System described in [8] consists of *three sequential phases* corresponding to *initialization (Init)* (comprising 8 discrete locations), *transmission (Tx)* (comprising 5 discrete locations), and *collision response (Coll)* (comprising 6 discrete locations). For simplicity, we consider only a single run of the protocol and thus ignore cross-over edges between phases. Each Sender (as given in Figure 10 of [8]) may then be described by the following sequential composition of timed automata

$$S_X = Init_X; Tx_X; Coll_X,$$

where X denotes either A or B.

Exploiting the fact that certain "cross-dependencies" do not exist in the system (for instance, between Tx_A and $Init_B$), and the consequent application of the CCL laws and the corresponding partial order equivalences, we may rewrite the system's composition as follows:

$$
\begin{aligned}
System' = Bus \,\|\, (\ & (Init_A \| Init_B) \\
& ; \\
& (FG_A \,\|\, FG_B \,\|\, TX_A \,\|\, TX_B) \\
& ; \\
& (Det_A \,\|\, Det_B \,\|\, Obs_A \,\|\, Obs_B \,\|\, Coll_A \,\|\, Coll_B)\ )
\end{aligned}
$$

Based on the methodology described at the end of Section 3, it follows that $System$ and $System'$ are po equivalent, and thus the satisfaction of the desired correctness property (expressed in CTL without *next*) is preserved when transforming $System$ into $System'$. An advantage of such a transformation is that $System'$ is much easier to reason about than $System$ (given that ; dominates in the former, as opposed to $\|$ in the latter). In fact, it may be seen that the number of discrete locations in $System'$ is a little over 7000, yielding a state space reduction by a factor of over 300.

Such a layered transformation may be seen as being complementary to the well-studied partial order reduction approach to the model checking of networks of timed automata, given that exactly the same class of system properties is preserved. In fact, such a layered transformation performs, in a certain sense, partial order reduction on the system *apriori*.

6 Conclusion

We have presented a framework for layered reasoning of complex real-time systems modelled as networks of timed automata. This was achieved by means of a layered composition operator that enables a combination of both parallel execution and sequential verification, by appropriately exploiting (in-)dependence conditions across components. The approach complements the partial order reduction approach in the verification of real-time systems, in the sense that layered transformation using the CCL laws and the resulting i/o and po equivalences bring about an *apriori* (partial order) reduction of the state space to be explored. Preliminary ideas on the application of the approach have been illustrated on a realistic example. Future work includes the extension of these techniques to more complex models of real-time systems such as Phase Event Automata [23], and their application to detailed analysis on realistic examples.

Acknowledgements. We wish to thank the reviewers for useful feedback.

References

1. Janssen, W., Poel, M., Xu, Q., Zwiers, J.: Layering of Real-Time Distributed Processes. In: Langmaack, H., de Roever, W.-P., Vytopil, J. (eds.) FTRTFT 1994 and ProCoS 1994. LNCS, vol. 863, pp. 393–417. Springer, Heidelberg (1994)
2. Elrad, T., Francez, N.: Decomposition of Distributed Programs into Communication Closed Layers. Science of Computer Programming 2(3), 155–173 (1982)
3. Zwiers, J.: Layering and Action Refinement for Timed Systems. In: Huizing, C., de Bakker, J.W., Rozenberg, G., de Roever, W.-P. (eds.) REX 1991. LNCS, vol. 600, pp. 687–723. Springer, Heidelberg (1992)
4. Janssen, W.: Layered Design of Parallel Systems, PhD Dissertation, Universiteit Twente (1994)

5. Alur, R., Dill, D.: A Theory of Timed Automata. Theoretical Computer Science 126(2), 183–235 (1994)
6. Behrmann, G., David, A., Larsen, K.: A Tutorial on Uppaal. In: Bernardo, M., Corradini, F. (eds.) SFM-RT 2004. LNCS, vol. 3185, pp. 200–236. Springer, Heidelberg (2004)
7. Bozga, M., Daws, C., Maler, O., Olivero, A., Tripakis, S., Yovine, S.: KRONOS: A Model-Checking Tool for Real-Time Systems. In: Ravn, A.P., Rischel, H. (eds.) FTRTFT 1998. LNCS, vol. 1486, pp. 298–302. Springer, Heidelberg (1998)
8. Havelund, K., Skou, A., Larsen, K.G., Lund, K.: Formal modeling and analysis of an audio/video protocol: an industrial case study using UPPAAL. In: RTSS'97, pp. 2–13. IEEE CS Press, Los Alamitos (1997)
9. Godefroid, P.: Using Partial Orders to Improve Automatic Verification Methods. In: Clarke, E., Kurshan, R.P. (eds.) CAV 1990. LNCS, vol. 531, pp. 176–185. Springer, Heidelberg (1991)
10. Bengtsson, J., Jonsson, B., Lilius, J., Yi, W.: Partial Order Reductions for Timed Systems. In: Sangiorgi, D., de Simone, R. (eds.) CONCUR 1998. LNCS, vol. 1466, pp. 485–500. Springer, Heidelberg (1998)
11. Dams, D., Gerth, R., Knaack, B., Kuiper, R.: Partial-order Reduction Techniques for Real-time Model Checking. Formal Aspects of Computing 10(5-6), 469–482 (1998)
12. Minea, M.: Partial Order Reduction for Model Checking of Timed Automata. In: Baeten, J.C.M., Mauw, S. (eds.) CONCUR 1999. LNCS, vol. 1664, pp. 431–436. Springer, Heidelberg (1999)
13. Zhao, J., Xu, H., Li, X., Zheng, T., Zheng, G.: Partial Order Path Technique for Checking Parallel Timed Automata. In: Damm, W., Olderog, E.-R. (eds.) FTRTFT 2002. LNCS, vol. 2469, pp. 417–432. Springer, Heidelberg (2002)
14. Lugiez, D., Niebert, P., Zennou, S.: A Partial Order Semantics Approach to the Clock Explosion Problem of Timed Automata. Theoretical Computer Science 345(1), 27–59 (2005)
15. Ben Salah, R., Bozga, M., Maler, O.: On Interleaving in Timed Automata. In: Baier, C., Hermanns, H. (eds.) CONCUR 2006. LNCS, vol. 4137, pp. 465–476. Springer, Heidelberg (2006)
16. Haakansson, J., Pettersson, P.: Partial Order Reduction for Verification of Real-Time Components. In: Raskin, J.-F., Thiagarajan, P.S. (eds.) FORMATS 2007. LNCS, vol. 4763, pp. 211–226. Springer, Heidelberg (2007)
17. Malinowski, J., Niebert, P.: SAT Based Bounded Model Checking with Partial Order Semantics for Timed Automata. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 405–419. Springer, Heidelberg (2010)
18. Bouyer, P., Petit, A.: Composition and Decomposition of Timed Automata. In: Wiedermann, J., Van Emde Boas, P., Nielsen, M. (eds.) ICALP 1999. LNCS, vol. 1644, pp. 210–219. Springer, Heidelberg (1999)
19. Dong, J.S., Hao, P., Qin, S., Sun, J., Yi, W.: Timed Automata Patterns. IEEE Transactions on Software Engineering 34(6), 844–859 (2008)
20. Alur, R., Brayton, R.K., Henzinger, T.A., Qadeer, S., Rajamani, S.K.: Partial-Order Reduction in Symbolic State Space Exploration. In: Grumberg, O. (ed.) CAV 1997. LNCS, vol. 1254, pp. 340–351. Springer, Heidelberg (1997)
21. Baier, C., Katoen, J.-P.: Principles of Model Checking. MIT Press, Cambridge (2008)
22. Olderog, E.-R., Dierks, H.: Real-Time Systems: Formal Specification and Automatic Verification. Cambridge University Press, Cambridge (2008)
23. Hoenicke, J., Olderog, E.-R.: CSP-OZ-DC: A Combination of Specification Techniques for Processes, Data and Time. Nordic Journal of Computing 9(4), 301–334 (2002)

A Conformance Testing Relation for Symbolic Timed Automata*

Sabrina von Styp[1], Henrik Bohnenkamp[1], and Julien Schmaltz[2,3]

[1] Software Modeling and Verification (i2)
Department of Computer Science
RWTH Aachen University Aachen, Germany
[2] School of Computer Science, Open University of the Netherlands
Heerlen, The Netherlands
[3] Institute for Computing and Information Sciences
Radboud University Nijmegen, The Netherlands

Abstract. We introduce Symbolic Timed Automata, an amalgamation of symbolic transition systems and timed automata, which allows to express nondeterministic data-dependent control flow with inputs and outputs and real-time behaviour. In particular, input data can influence the timing behaviour. We define two semantics for STA, a concrete one as timed labelled transition systems and another one on a symbolic level. We show that the symbolic semantics is complete and correct w.r.t. the concrete one. Finally, we introduce symbolic conformance relation *stioco*, which is an extension of the well-known *ioco* conformance relation. Relation *stioco* is defined using FO-logic on a purely symbolic level. We show that *stioco* corresponds on the concrete semantic level to Krichen and Tripakis' implementation relation *tioco* for timed labelled transition systems.

Keywords: Real-time conformance testing, symbolic execution, implementation relation, semantics, FO logics.

1 Introduction

Specification-based testing is a branch of model-based testing, where test-cases are derived automatically from a formal specification (given as a labelled transition system or a related formalism) and executed against real-life implementations. The distinction to many other instances of model-based testing is that the whole test-case derivation and test-execution process is described formally. This rigorous definition enables the proof of soundness of the approach. In particular, it is possible to show that the execution of a derived test-case does

* This work has been performed as part of the "Quantitative System Properties in Model-Driven-Design of Embedded Systems" (Quasimodo) project, supported by the Seventh Research Framework Programme of the European Commission. Grant agreement number: INFSO-ICT-214755.

K. Chatterjee and T.A. Henzinger (Eds.): FORMATS 2010, LNCS 6246, pp. 243–255, 2010.

not yield false positives, *i.e.,* test-failures, when the implementation is actually correct. To achieve this form of rigorousity, a formal criterion is needed which relates specifications to its correct implementations: the *conformance relation*. Specification-based testing has developed over the last two decades. A well-known representative is the *ioco framework* [19], where the specification is given as a a labelled transition system (*LTS*) with input and output actions. The conformance relation is called *ioco*. The *ioco* relation expresses that an implementation may only produce outputs if those outputs are also produced by the specification. Additionally, *ioco* possesses a notation for silence, denoted *quiescence*. An implementation that is *ioco*-correct may only be quiescent if this is allowed by the specification. There exists several tools for the derivation of *ioco* test-cases, e.g. TorX [3], TGV [14] and AGEDIS TOOL SET [11]. Based on *ioco*, recently more expressive formalisms have been considered to serve as specifications. *Timed Automata* have been proposed as specification formalisms in several approaches for testing real-time behaviours [16,4,5]. Different notions of conformance have been defined on the basis of timed *LTS*(*TLTS*), *i.e.,* only on the semantic level. *Symbolic Transition Systems* (*STS*) [8,9] have been introduced to specify systems with input- and output-data. *STS* are *LTS* extended with a notion of data and data-dependent control flow based on first order logic. The symbolic representation of data in *STS* allows for infinite data domains without facing the problems of infinite branching and infinite state space. For *STS*, the implementation relation *sioco* has been developed, which is defined solely within the FO-Logic framework on *STS* level [9].

What does not exist yet is a combination of real-time and data. In this paper we take first steps in the direction of specification-based testing for systems combining input/output data with real-time aspects in a non-orthogonal way, *i.e.,* the input data can influence the real-time behaviour. In particular, we introduce a conformance relation which takes data and real-time into account.

Our contributions are (1) a new formalism – called Symbolic Timed Automata (*STA*) – for modelling reactive real-time systems with data input and output; (2) a concrete operational semantics (in terms of timed labelled transition systems) and a symbolic trace semantics for this formalism; (3) a family of conformance relations $stioco_{\mathcal{F}_s}$, which expresses a correctness criterion of input-enabled implementations, formulated as *STA*, with specifications, also given as *STA*; (4) a theorem stating that $stioco_{\mathcal{F}_s}$ coincides on the concrete semantical level with *tioco* of Krichen and Tripakis [16]. Our formalism allows the real-time behaviour to be influenced by inputs. While allowing nondeterministic *STA* in general, we restrict ourselves in this paper to *branching nondeterministic STA*, *i.e.,* without τ-steps.

Related Work. A detailed comparison of the different notions of conformance for real-time testing is given by Schmaltz et al. [17]. In the testing tool UPPAAL Tron [13], a pragmatic approach to combine data and time is implemented. It is possible to let clock constraints depend on integer variables. Moreover, global integer variables can be designated as input or output parameters to input or output actions. With this a notion of value passing is implemented.

However, values are limited to finite sub-sets of integers, which allows an explicit representation of data as actions on Timed Automata level. This approach is hardly formally described, least of all on a symbolic level. A similar approach is taken in JTorX [2].

STS and the implementation relation *sioco* have been introduced in [9]. This implementation relation is used in the java testing tool JAMBITION introduced by Frantzen *et al.* [7]. JAMBITION uses a random and on-the-fly approach to automatically test web services based on *STS* specifications. In order to simulate the *STS* in JAMBITION the Java library STSIMULATOR has been developed [18].

Another tool for symbolic testing is STG [6], an extension of TGV. It automatically derives symbolic test-cases from a given formal model and a test purpose. The issue of symbolic test-case generation and selection has been addressed in [21] and [15].

2 Timed Transition Systems and *Tioco*

A *timed labelled transition system* (*TLTS*) is a tuple $\langle S, Act, s_0, \rightarrow \rangle$ with S a set of states, $Act = Act_I \cup Act_U$ a disjoint union of two sets of input- and output-actions, s_0 the starting state, and $\rightarrow \subseteq S \times (Act \cup \mathbb{R}_{\geq 0}) \times S$ a transition relation, where the following conditions must hold: $\forall s, s', s'' \in S$ (i) $s \xrightarrow{0} s$; (ii) $s \xrightarrow{d} s' \xrightarrow{d'} s''$ if and only if $s \xrightarrow{d+d'} s''$; (iii) $s \xrightarrow{d} s'$ and $s \xrightarrow{d} s''$ implies $s' = s''$. In the following we consider a fixed *TLTS* $\langle S, Act, s_0, \rightarrow \rangle$, and identify it with its starting state s_0. The generalised transition relation $\Rightarrow \subseteq S \times (Act \cup \mathbb{R}_{\geq 0})^* \times S$ is defined as the least relation satisfying the following rules: (i) $s \stackrel{\epsilon}{\Rightarrow} s \ \forall s \in S$; (ii) $s \stackrel{\sigma \cdot d}{\Longrightarrow} s'$, if $s \stackrel{\sigma}{\Rightarrow} s'' \xrightarrow{d} s'$ for $d \in \mathbb{R}_{\geq 0}$; (iii) $s \stackrel{\sigma \cdot a}{\Longrightarrow} s'$, if $s \stackrel{\sigma}{\Rightarrow} s'' \xrightarrow{a} s'$, for $a \in Act$. We consider normalised traces where actions and delays strictly alternate, starting with a delay. It has been shown that this set characterises the set of all traces [5]. A timed trace is thus a sequence $\sigma \in (\mathbb{R}_{\geq 0} \cdot Act)^* \cdot (\mathbb{R}_{\geq 0} + \epsilon)$ such that $s_0 \stackrel{\sigma}{\Rightarrow} s'$ for some $s' \in S$. The set of traces of *TLTS* S is noted $\mathbf{traces}(S)$. The set of states that can be reached from state s via a trace σ is denoted as $s \ \mathbf{after}_t \ \sigma$.

Definition 1 ($\mathbf{after}_t$). *Let $\langle S, Act, s_0, \rightarrow \rangle$ be a TLTS and $\sigma \in (\mathbb{R}_{\geq 0} \cdot Act)^* \cdot (\mathbb{R}_{\geq 0} + \epsilon)$. Then $s \ \mathbf{after}_t \ \sigma =_{def} \{s' \mid s \stackrel{\sigma}{\Rightarrow} s'\}$.*

Crucial for the definition of *tioco* is the set of delay and output labels of the outgoing transitions of a state.

Definition 2 ($elapse(s)$ and $\mathbf{out}_t(s)$). *We define $elapse(s) =_{def} \{d \mid s \xrightarrow{d}\}$, and $\mathbf{out}_t(s) =_{def} \{o \in Act_U \mid s \xrightarrow{o}\} \cup elapse(s)$. For $S' \subseteq S, \mathbf{out}_t(S') =_{def} \bigcup_{s \in S'} \mathbf{out}_t(s)$.*

As in the *ioco* theory, we assume that implementations under test are input-enabled.

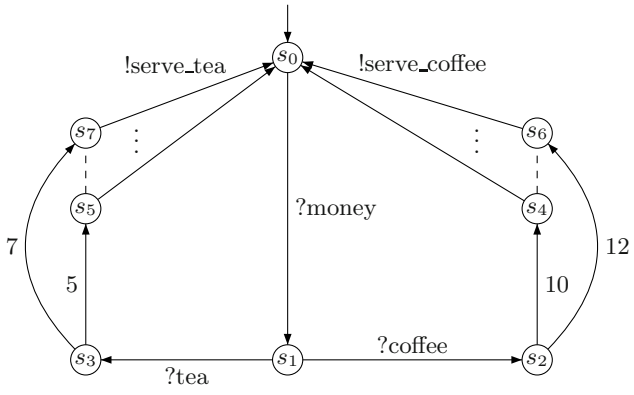


Fig. 1. *TLTS* Specification of a Beverage Vending Machine

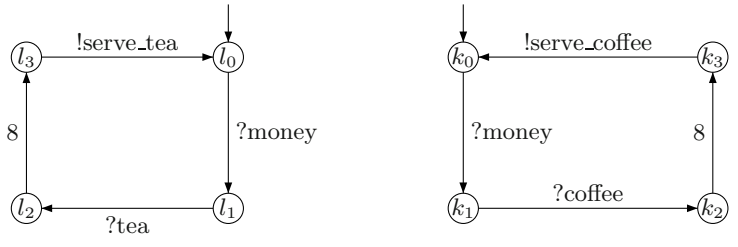


Fig. 2. Implementations of a Beverage Vending Machine

Definition 3 (Input-enabled *TLTS*). *A TLTS* $\langle S, Act, s_0, \rightarrow \rangle$ *is called input-enabled if and only if for all* $s \in S$ *and all* $i \in Act_I$*:* $s \xrightarrow{i}$.

With the introduced concepts we can define the family of implementation relations $tioco_{\mathcal{F}}$.

Definition 4 ($tioco_{\mathcal{F}}$). *Let* $\mathcal{P}$ *be an input-enabled TLTS,* $\mathcal{S}$ *a TLTS, and* $\mathcal{F} \subseteq$ **traces**$(\mathcal{S})$*. Then* $\mathcal{P}$ *conforms to* $\mathcal{S}$ *w.r.t.* $tioco_{\mathcal{F}}$ *(written* $\mathcal{P}$ $tioco_{\mathcal{F}}$ $\mathcal{S}$*) if and only if the following holds:* $\forall \sigma \in \mathcal{F} : \mathbf{out}_t(\mathcal{P}\ \mathbf{after}_t\ \sigma) \subseteq \mathbf{out}_t(\mathcal{S}\ \mathbf{after}_t\ \sigma)$.

Example 1. In Figure 1, a *TLTS* specifying a beverage vending machine is sketched. After inserting money, users can choose either tea or coffee. The former is produced between 5 and 7 time units after pushing the ?tea button. The latter is produced between 10 and 12 time units after pushing the ?coffee button (the time intervals are indicated by the dashed lines, which stand for a *continuum* of states). Figure 2 shows two implementations of beverage machines. None of these implementations conform to the specification according to the *tioco* relation. The implementation on the left (starting state l_0) is too slow to produce tea. The implementation on the right (starting state k_0) is too fast to produce coffee. Formally, the reasons for non-conformance are (1) that $8 \in elapse(l_0\ \mathbf{after}_t\ ?money \cdot ?tea)$ but $8 \notin elapse(s_0\ \mathbf{after}_t\ ?money \cdot ?tea)$ and

(2) that $!serve\_coffee \in \mathbf{out}_t(k_0 \ \mathbf{after}_t \ ?money \cdot ?coffee \cdot 8)$ but $!serve\_coffee \notin \mathbf{out}_t(s_0 \ \mathbf{after}_t \ ?money \cdot ?coffee \cdot 8)$.

3 Symbolic Timed Automata

A symbolic timed automaton (*STA*) combines a timed automaton [1,12] with a *symbolic transition system* (*STS*), as introduced in [8,9]. *STS* extend labelled transition systems with *variables*, *input- and output-parameters* associated with input- and output-actions, *guards*, which are first-order logic formulas controlling the enabledness of transitions, and *variable updates*, which manipulate the values of variables while performing transitions. In *STA*, the concepts of timed automata, namely clocks, guards, invariants and clock resets, are integrated. In particular, clock constraints become just another sort of FO formulas.

3.1 An Example

Figure 3 shows an *STA* modelling a beverage vending machine. The machine accepts money in bills (parameter x of input ?money) and returns change in coins (parameter x of output !serve). If there is not enough change in the machine on a bill, the bill is returned (output !return). Otherwise, the user can choose a beverage, the change is returned, and the beverage served (parameter y of output !serve). Variables x, y, t are *interaction variables*. Interaction variables represent the possible values that can be passed through input and output actions. Variables $\mathtt{i}, \mathtt{q}, \mathtt{change}, \mathtt{beverage}, \mathtt{money}, \mathtt{time}$ are *location variables — i.e.,* the store of the *STA* — and c is a clock, conceptually identical with a clock of a timed automaton. The switch from l_0 to l_1 is labelled with input action ?money, with parameter x. Only if the value of x is larger or equal than 1 the switch can be executed. In that case, `change` is mapped to $x - 1$ and `money` to x. From location l_1, the money is returned immediately, if $\mathtt{change} > \mathtt{q}$, *i.e.,* the required change is not available. In that case, `money` and `change` are both mapped to 0. If $\mathtt{change} < \mathtt{q}$, it is possible to choose a beverage via input ?choice, where the type of beverage is communicated via y, and t stands for the time after which the machine shall serve the beverage. t could for example express the brewing time of a tea or whether an espresso should be short or long. Input t is assigned to location variable `time`, which occurs in the invariant of location l_2 and the clock guard of the switch $l_2 \rightarrow l_0$. Together they ensure that location l_2 is left after exactly `time` time units. In that case, output !serve with output parameters y and x is sent, where $x = \mathtt{change}$ and $y = \mathtt{beverage}$ is required by the guard. Location variables `money` and `change` are then mapped to 0, and `q` is mapped on $\mathtt{q} - \mathtt{change}$. Note that location variable `i`, used as bound in the invariant of l_1 and the guard leading back to l_0, is never explicitly set in the *STA*. It is assumed that location variables (thus also `i`) are initialised when starting the *STA*. Different initialisations might define different behaviours of the same *STA*.

In this example, the time at which the transition from location l_2 to l_0 can be taken is controlled by input data t given in the previous transition (from l_1

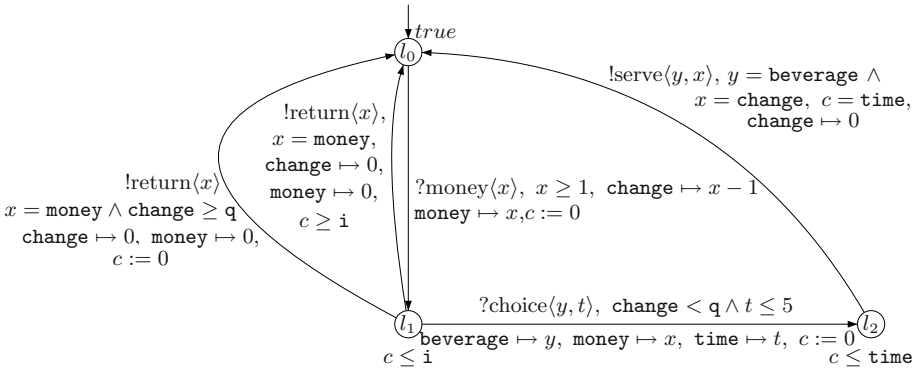


Fig. 3. *STA* Beverage Vending Machine

to l_2). In an *STA*, location variables are allowed to serve as bounds in clock guards and invariants. This is the only instance in which we allow an interaction between time (clocks) and data (variables). Interaction variables cannot be used to control time. This prevents the case where the time at which a transition can be taken depends on the value of the parameter of the action of that same transition. In our example, time t must first be stored in local variable `time` before it can be used to control time.

3.2 Definition and Concrete Semantics of *STA*

An *STA* is defined over a two sorted FO structure $FO_\Sigma = (\{\mathfrak{U}_d, \mathfrak{U}_t\}, \{r_\Sigma \mid r \in \mathcal{R}\}, \{f_\Sigma \mid f \in \mathcal{F}\})$, where $\mathfrak{U}_d$ denotes the universe for data and $\mathfrak{U}_t$ the time domain (*e.g.*, the reals). The signature contains constant 0_t, binary addition $+_t$, and relations $\leq, <$, declared for all combinations $\{t, d\} \times \{t, d\}$, *i.e.,* comparisons between time and data.

Clock-constraints are FO formulas using the above relations. Let $\mathcal{C}$ be the set of clocks and *Var* be a set of variables. An *atomic clock-constraint* is a formula of the form $b_l \prec x \prec b_u$ for $x \in \mathcal{C}$, $\prec \in \{<, \leq\}$, and $b_l, b_u \in \mathbb{N}_{FO} \cup Var$, where $\mathbb{N}_{FO}$ stands here for the representation of natural numbers on the logic level[1]. Clock constraints are conjunctions of atomic clock constraints. The set of all clock constraints over clock set $\mathcal{C}$ and variables *Var* is denoted by $\mathcal{B}(\mathcal{C}, Var)$, and $\mathcal{B}(\mathcal{C})$, if $Var = \emptyset$. Bounds of atomic clock constraints can be variables. We denote by $\mathfrak{T}(V)$ the set of all terms over a variable set V and by $\mathfrak{F}(V)$ the set of all FO formulas. Function $\rho : V \to \mathfrak{T}(V)$ is called a term mapping.

Definition 5 (Symbolic Timed Automaton). *A symbolic timed automaton is a tuple* $\mathcal{A} = \langle L, l_0, \mathcal{V}, \mathcal{I}, \mathcal{G}, \texttt{type}, \mathcal{C}, Inv, \rightarrow\rangle$ *with* L *being a finite set of locations,* $l_0 \in L$ *is the initial location,* $\mathcal{V}$ *and* $\mathcal{I}$ *disjoint sets of location and interaction variables,* $\mathcal{G}$ *is a set of* gates, $\texttt{type} : \mathcal{G} \to 2^{\mathcal{I}}$ *assigns sets of interaction*

[1] Other literals are imaginable as bounds, depending on the choice of $\mathfrak{U}_d$.

variables to gates, $\mathcal{C}$ is a set of clocks, $Inv : L \to \mathcal{B}(\mathcal{C}, \mathcal{V})$ assigns a clock invariant to a location, and $\to \subseteq L \times \mathcal{G} \times \mathfrak{F}(Var) \times \mathcal{B}(\mathcal{C}, \mathcal{V}) \times \bigcup_{V \subseteq \mathcal{V}} \mathfrak{T}(Var)^V \times 2^{\mathcal{C}} \times L$ is the transition relation, where $Var = \mathcal{I} \cup \mathcal{V}$.

As usual, we write $l \xrightarrow{\gamma,\varphi,g,\rho,r} l'$ to denote $(l, \gamma, \varphi, g, \rho, r, l') \in \to$, where γ is the gate, φ the data guard, g the clock guard, ρ the update function, and r the clock reset.

A variable valuation is a function $\vartheta : Var \to \mathfrak{U}_d$ from variables to concrete values in the universe $\mathfrak{U}_d$. A *clock valuation* is a function $u : \mathcal{C} \to \mathfrak{U}_t$. We denote with $[\mathcal{C} \mapsto 0]$ the constant 0 clock valuation. For $d \in \mathfrak{U}_t$, we define $(u + d)(c) = u(c) + d$. If ϑ is a variable valuation and $C \in \mathcal{B}(\mathcal{C}, Var)$, then we denote with $C[\vartheta]$ the clock constraint, where every occurrence of a variable $x \in Var$ is replaced by $\vartheta(x)$; thus, $C[\vartheta] \in \mathcal{B}(\mathcal{C})$. We write $u \models C$, if clock valuation u satisfies clock constraint $C \in \mathcal{B}(\mathcal{C})$, *i.e.,* if the relational expression obtained by replacing all occurrences of clock names c by $u(c)$ evaluates to true. If $r \subseteq \mathcal{C}$, then $u[r \mapsto 0](c) = 0$, if $c \in r$, and $u[r \mapsto 0](c) = u(c)$, otherwise. The semantics of an *STA* is given as a *TLTS*, defined as follows.

Definition 6. *Let $\mathcal{A} = \langle L, l_0, \mathcal{V}, \mathcal{I}, \mathcal{G}, \mathtt{type}, \mathcal{C}, Inv, \to \rangle$ be an STA. Its TLTS semantics in the context of an initial valuation $\iota \in \mathfrak{U}^{\mathcal{V}}$ of location variables and $\zeta_0 = [\mathcal{C} \mapsto 0]$ for clocks, is a TLTS $[\![\mathcal{A}]\!]_\iota = \langle S, Act, s_0, \to \rangle$, where $S = L \times \mathfrak{U}_d^{\mathcal{V}} \times \mathfrak{U}_t^{\mathcal{C}}$, $s_0 = (l_0, \iota, \zeta_0)$, $Act = \bigcup_{\gamma \in \mathcal{G}} (\{\gamma\} \times \mathfrak{U}^{\mathtt{type}(\gamma)})$, and $\to$ is defined as the least set of transitions derivable by the following rules:*

$$\langle Delay \rangle \frac{\zeta \models Inv(l)[\vartheta] \quad \forall d' \leq d : \zeta + d' \models Inv(l)[\vartheta]}{(l, \vartheta, \zeta) \xrightarrow{d} (l, \vartheta, \zeta + d)} (d \in \mathbb{R}_{\geq 0})$$

$$\langle Action \rangle \frac{l \xrightarrow{\gamma,\varphi,\rho} l' \quad \vartheta \cup \varsigma \models \varphi \quad \zeta' \models Inv(l')[\vartheta']}{(l, \vartheta, \zeta) \xrightarrow{(\gamma, \varsigma(\mathtt{type}(\gamma)))} (l', \vartheta', \zeta')} (\varsigma \in \mathfrak{U}^{\mathtt{type}(\gamma)})$$

with $\vartheta' = ((\vartheta \cup \varsigma)_{ev} \circ \rho)_{\mathcal{V}}$ and $\zeta' = ((\zeta)_{ev} \circ \rho)_{\mathcal{C}}$.

With $(\cdot)_{\text{ev}}$ we denote the lifting of a variable/clock valuation to terms. With $(\cdot)_{\mathcal{V}}$ and $(\cdot)_{\mathcal{C}}$ we denote the restriction of a valuation or term mapping to sets $\mathcal{V}$ and $\mathcal{C}$. In Definition 6 we see that we can only delay if all clock valuations ζ before the delay and all clock valuations ζ' after the delay satisfy the invariant of location l. Delaying has no influence on the location itself or on the valuation of location variables ϑ. To take a switch from l to l' with gate γ, the constraint φ over variables and clocks and the invariant of the next location l' have to be satisfied. Important is that the invariant of l' has to be satisfied after the clocks in ρ have been set to zero.

4 Symbolic Trace Semantics for *STA*

Inspired by Frantzen *et al.* [9], we define a symbolic trace semantics for *STA*, which is sound and complete *w.r.t.* the *TLTS* semantics described before. We use *delay-variable* d to represent time symbolically. Variable d is of sort t and

represents delaying in a location. We define the following two notations. First, for $r \subseteq \mathcal{C}$, we denote with $FO(r)$ a term-mapping that maps all clocks $c \in r$ to zero, *i.e.,* $FO(r) : c \mapsto 0_t$ for $c \in r$ and $c \mapsto c$ otherwise. Second, $\varrho : \mathcal{C} \to \mathfrak{T}(\mathcal{C} \cup \mathcal{T})$ is a partial term-mapping that expresses the passing of time: $\varrho : c \mapsto c + d$, *i.e.,* delaying is described by adding delay variable d to all clocks $c \in \mathcal{C}$.

The symbolic trace semantics is expressed by transition relation $l \stackrel{\sigma,\varphi,\rho}{\Longrightarrow} l'$, where φ denotes the logical condition which needs to be fulfilled to reach location l' from l with symbolic trace σ, and ρ is a term mapping that denotes symbolically the possible variable and clock valuations after l' has been reached with σ. Relation $\Rightarrow$ thus describes the symbolic execution of the *STA*. To define φ and ρ, we introduce history variables, *i.e.,* variables that allow to distinguish between different input- and output values communicated over the gates, and time delays spent in locations. The history variables are an infinite number of "copies" $i_1, i_2, i_3, \ldots$ of each interaction variable $i \in \mathcal{I}$, and delay history variables $d_1, d_2, d_3, \ldots$ for variable d. We define $\mathcal{I}_n = \{i_n | i \in \mathcal{I}\}$ for $n \geq 1$ and $\widehat{\mathcal{I}} =_{def} \bigcup_{n=1}^{\infty} \mathcal{I}_n$. Similarly, we consider sets $\mathcal{T}_n = \{d_n\}$ for $(n \geq 1)$ and $\widehat{\mathcal{T}}$ defined analogously to $\widehat{\mathcal{I}}$. Let $\mathcal{H}_n =_{def} \mathcal{I}_n \cup \mathcal{T}_n$ and $\widehat{\mathcal{H}} =_{def} \widehat{\mathcal{I}} \cup \widehat{\mathcal{T}}$, $\mathcal{H} =_{def} \mathcal{I} \cup \mathcal{T}$. We define renaming bijections $r_n : \mathcal{H} \to \mathcal{H}_n$ with $r_n(v) = v_n$ for all $v \in \mathcal{H}$, and $n \in \mathbb{N}$. Function $s^{\gg i} : \widehat{\mathcal{H}} \to \widehat{\mathcal{H}}$ is defined as $s^{\gg i} : x \mapsto x_{n+i}$ for all $x = x_n \in \widehat{\mathcal{H}}$. In the following, the entirety of all relevant variables is the set $\widehat{Var} =_{def} \mathcal{V} \cup \mathcal{C} \cup \mathcal{H} \cup \widehat{\mathcal{H}}$, and $Var =_{def} \widehat{Var} \setminus \widehat{\mathcal{H}}$.

If $\rho \in (\mathfrak{T}(\widehat{Var}))^{\widehat{Var}}$ and $t \in \mathfrak{T}(\widehat{Var})$, we denote by $t[\rho]$ the term obtained by substituting all variables x occurring in t by $\rho(x)$. Analogously for all formulas $\varphi \in \mathfrak{F}(\widehat{Var})$, where all *free* variables are substituted.

4.1 Symbolic Trace Semantics

The symbolic trace semantics of an *STA* $\langle L, l_0, \mathcal{V}, \mathcal{I}, \mathcal{G}, \mathtt{type}, \mathcal{C}, Inv, \rightarrow \rangle$ is then given by the transition relation $\Rightarrow \subseteq L \times ((d \cdot \mathcal{G})^* \cdot \{d, \varepsilon\}) \times \mathfrak{F}(\widehat{Var}) \times (\bigcup_{V \subseteq \mathcal{V}} \mathfrak{T}(\widehat{Var})^V \cup \bigcup_{C \subseteq \mathcal{C}} \mathfrak{T}(\mathcal{C} \cup \widehat{\mathcal{T}})^C) \times L$, which is defined as follows:

Definition 7 (Generalised Switch Relation for *STA*).

$$(d)\ \frac{}{l \stackrel{d,\kappa[\varrho][r_1],\varrho[r_1]}{\Longrightarrow} l} \qquad (Sd)\ \frac{l \stackrel{\sigma\cdot\gamma,\varphi,\rho}{\Longrightarrow} l'}{l \stackrel{\sigma\cdot\gamma\cdot d,\ \varphi \wedge \kappa'[\varrho][r_n][\rho],\ \Theta(\rho,r_n,\varrho)}{\Longrightarrow} l'} \qquad (S\gamma)\ \frac{l \stackrel{\sigma\cdot d,\varphi,\rho}{\Longrightarrow} l'' \quad l'' \stackrel{\gamma,\psi,\pi}{\longrightarrow} l'}{l \stackrel{\sigma\cdot d\cdot\gamma,\ \varphi \wedge \psi[r_n][\rho] \wedge \kappa'[\pi][\rho],\ \Theta(\rho,r_n,\pi)}{\Longrightarrow} l'}$$

where $n = |\sigma| + 2$, $\Theta(a, b, c) = c[b][a]$, $\kappa \equiv Inv(l)$, *and* $\kappa' \equiv Inv(l')$.

Rule (d) states that delaying in a location l is possible as long as formula $\kappa[\varrho][r_n]$, *i.e.,* the historised and updated invariant of l, is satisfied. The clocks change according to term mapping $\varrho[r_1]$.

Rule (Sd) states that if location l' is reached from l with trace $\sigma \cdot \gamma$ under condition φ, then the condition to delay further in l' is the conjunction of φ and $\kappa[\varrho][r_n][\rho]$. As an example, we assume that $n = 7$, $\kappa \equiv c \leq v$, where c is

a clock, and v a location variable, and $\rho(c) = d_1 + d_3 + d_5$, $\rho(v) = 4$. Then $\kappa[\varrho] \equiv c + d \leq v$, $\kappa[\varrho][r_7] \equiv c + d_7 \leq v$, and $\kappa[\varrho][r_7][\rho] \equiv d_1 + d_3 + d_5 + d_7 \leq 4$.

Similarly, Rule $(S\gamma)$ states that if l'' is reached from l with $\sigma{\cdot}d$ under condition φ, then the condition to reach l' from l with trace $\sigma{\cdot}d{\cdot}\gamma$ is the conjunction of φ and $\psi[r_n][\rho] \wedge \kappa[\pi][\rho]$. Formula $\psi[r_n][\rho]$ is the "historised" enabling condition of switch $l'' \xrightarrow{\gamma,\psi,\pi} l'$, with all variables and clocks substituted according to ρ. Formula $\kappa[\pi][\rho]$ is the historised invariant of l' and expresses an extra condition on the symbolic clock valuations under which l' may be entered.

In both rules (Sd) and $(S\gamma)$, the new update mapping Θ for variables and clocks is obtained by the concatenation of the current update mapping ρ with the renaming function r_n and the variable update π for $(S\gamma)$ or the clock update ϱ for (Sd). If, for example, we assume $\rho(c) = d_1 + d_3$, $r_n = 5$, then $\varrho[r_n][\rho](c) = ([\rho] \circ [r_n] \circ \varrho)(c) = d_1 + d_3 + d_5$.

4.2 Symbolic States and Relation to *TLTS* Semantics

To record under which conditions a location can be reached and with what potential variable valuations, we introduce *symbolic states*. Whenever $l \xRightarrow{\sigma,\varphi,\rho} l'$, tuple (l', φ, ρ) is a symbolic state. If in φ and ρ only history variables with an index up to at most i occur, we note this fact by indexing the symbolic state with i, *i.e.,* in this example $(l', \varphi, \rho)_i$.

A symbolic state (l, φ, ρ) defines implicitly a set of concrete states $[\![(l, \varphi, \rho)]\!]_\iota$ for $\iota \in \mathfrak{U}_d^{\mathcal{V}}$ (as defined in the *TLTS* semantics above). Let $\upsilon \in \mathfrak{U}_d^{\widehat{\mathcal{I}}}$ and $\varpi \in \mathfrak{U}_t^{\widehat{\mathcal{I}}}$. Then $[\![(l, \varphi, \rho)]\!]_{\iota,\upsilon,\varpi\cup[\mathcal{C}\mapsto 0]} =_{def} \{(l, ((\iota \cup \upsilon)_{\mathrm{ev}} \circ \rho)_\mathcal{V}, ((\varpi \cup [\mathcal{C} \mapsto 0])_{\mathrm{ev}} \circ \rho)_\mathcal{C}) \mid \iota \cup \upsilon \cup \varpi \cup [\mathcal{C} \mapsto 0] \models \varphi\}$. Note that $[\![(l, \varphi, \rho)]\!]_{\iota,\upsilon,\varpi}$ is either a singleton or empty.

Also traces $\sigma \in (d{\cdot}\mathcal{G})^*$ have an interpretation on the semantic level. Let $\chi \in \mathfrak{F}(\widehat{\mathcal{H}} \cup \mathcal{V} \cup \mathcal{C})$. Then we call (σ, χ) an extended trace. χ can be chosen freely. The set of all symbolic extended traces of an *STA* $\mathcal{S}$ is defined by the following set: $\mathcal{E}Traces(\mathcal{S}) = \{(\sigma, \chi) \mid l_0 \xRightarrow{\sigma,\varphi,\rho} l',\ \chi \in \mathfrak{F}(\widehat{\mathcal{H}} \cup \mathcal{V} \cup \mathcal{C})\}$. Defining ι, υ, ϖ as above, $[\![(\sigma, \chi)]\!]_{\iota,\upsilon,\varpi\cup[\mathcal{C}\mapsto 0]} = \{\mathbf{ettraces}_{\upsilon,\varpi}(\sigma) \mid \iota \cup \upsilon \cup \varpi \models \chi\}$, where $\mathbf{ettraces}_{\upsilon,\varpi}$ is defined inductively as follows:

$$\mathbf{ettraces}_{\upsilon,\varpi}(\epsilon) = \epsilon$$
$$\mathbf{ettraces}_{\upsilon,\varpi}(\sigma \cdot g) = \mathbf{ettraces}_{\upsilon,\varpi}(\sigma) \cdot (g, \upsilon(r_{length(\sigma)+1}(\mathtt{type}(g))))$$
$$\mathbf{ettraces}_{\upsilon,\varpi}(\sigma \cdot d) = \mathbf{ettraces}_{\upsilon,\varpi}(\sigma) \cdot \varpi\left(d_{length(\sigma)+1}\right).$$

The following two theorems state that the interpretations of symbolic states and extended traces are correct *w.r.t.* the *TLTS* semantics. Variable mapping $\mathtt{id}$ is defined as $\mathtt{id}(x) = x$.

Theorem 1 (Soundness). *For all* $\varpi \in \mathfrak{U}_t^{\widehat{\mathcal{I}}}$, $\iota \in \mathfrak{U}_d^{\mathcal{V}}$ *and* $\upsilon \in \mathfrak{U}_d^{\widehat{\mathcal{I}}}$ *it holds that* $l \xRightarrow{\sigma,\varphi,\rho} l'$ *and* $\iota \cup \upsilon \cup \varpi \cup [\mathcal{C} \mapsto 0] \models \varphi$ *implies* $[\![(l, \top, \mathtt{id})]\!]_{\iota,\upsilon,\varpi\cup[\mathcal{C}\mapsto 0]} \xRightarrow{[\![(\sigma,\varphi)]\!]_{\iota,\upsilon,\varpi\cup[\mathcal{C}\mapsto 0]}} [\![(l', \varphi, \rho)]\!]_{\iota,\upsilon,\varpi\cup[\mathcal{C}\mapsto 0]}$

Theorem 2 (Completeness). *For all semantical states* $(l, \upsilon, \zeta) \in L \times \mathfrak{U}_d^{\mathcal{V}} \times \mathfrak{U}_t^{\mathcal{C}}$ *such that* $(l_0, \iota, [\mathcal{C} \mapsto 0]) \overset{\overline{\sigma}}{\Longrightarrow} (l, \upsilon, \zeta)$ *for* $\iota \in \mathfrak{U}_d^{\mathcal{V}}$ *and some timed trace* $\overline{\sigma}$*, there is a*

valuation $\upsilon' \in \mathfrak{U}_d^{\widehat{\mathcal{I}}}$, $\varpi \in \mathfrak{U}_t^{\widehat{\mathcal{I}}}$ *and a transition* $l_0 \xrightarrow{\sigma,\varphi,\rho} l$ *such that* $\iota \cup \upsilon' \cup \varpi \cup [\mathcal{C} \mapsto 0] \models \varphi$, $\overline{\sigma} = [\![(\sigma,\varphi)]\!]_{\iota,\upsilon',\varpi\cup[\mathcal{C}\mapsto 0]}$ *and* $(l,\upsilon,\zeta) = [\![(l,\varphi,\rho)]\!]_{\iota,\upsilon',\varpi\cup[\mathcal{C}\mapsto 0]}$.

The proofs of Theorems 1 and 2 can be found in [20].

5 *Stioco* - A Symbolic Timed Implementation Relation

Taking the symbolic trace semantics in Definition 7 as a foundation, we define symbolic implementation relation *stioco*. This definition is based on two central notions: **after**, a function which returns the symbolic states reachable with an extended trace, and **out**, the outputs that can potentially be observed from a set of symbolic states. The big difference is that outputs are accompanied by FO formulas which state the conditions under which outputs can be observed.

Definition 8 (after). *Let* $(l,\varphi,\rho)_i$ *be a symbolic state with index* i *and* (σ,χ) *an extended trace with* n *the length of* σ. *Then* **after** *is defined as*

$$(l,\varphi,\rho)_i\mathbf{after}(\sigma,\chi) =_{def} \{(l',\varphi'(\psi),\rho'(\pi)) \mid l \xrightarrow{\sigma,\psi,\pi} l'\},$$

where $\varphi'(\psi) = \varphi \wedge ((\psi \wedge \chi)[s^{\gg i}])[\rho]$, *and* $\rho'(\pi) = ([\rho] \circ [s^{\gg i}] \circ \pi)$.

$\varphi'(\psi)$ is the conjunction of φ and the condition $\psi \wedge \chi$, where every index of a history variable is increased by i and every clock and location variable is substituted according to ρ. $\rho'(\pi)$ is the symbolic variable valuation of the location variables and clocks after (σ,χ) has been executed. Note that the symbolic states in $(l,\varphi,\rho)_i\mathbf{after}(\sigma,\chi)$ have index $i+n$.

Symbolic observations are tuples (γ,φ,ψ), where $\gamma \in \mathcal{G}_{U_d}$, φ is a general enabling condition for γ and ψ is a special enabling condition. φ and ψ are thus both formulas, which are however defined over different variable sets: $\varphi \in \mathfrak{F}(\widehat{Var})$ and $\psi \in \mathfrak{F}(Var)$. The set of symbolic observations, denoted as $\mathcal{O}$, is thus defined as $\mathcal{O} =_{def} \mathcal{G}_{U_d} \times \mathfrak{F}(\widehat{Var}) \times \mathfrak{F}(Var)$.

The **out**-set of a symbolic state is defined, similar to $\mathbf{out}_t$ for *TLTS*, as the union of delays and output actions that can be made in the symbolic state. We use symbolic observations in order to symbolically represent the conditions under which an output or a delay may be observed.

Definition 9 (out). *Let* (l,φ,ρ) *be a symbolic state. Then* $\mathbf{out}((l,\varphi,\rho))$ *is a set of symbolic observations, defined as follows.*

$$\mathbf{out}((l,\varphi,\rho)) =_{def} \{(\gamma,\varphi,\psi[\rho] \wedge Inv(l')[\pi][\rho]) \in \mathcal{O} \mid \exists \gamma \in \mathcal{G}_U, \psi, \pi, l' : l \xrightarrow{\gamma,\psi,\pi} l'\} \cup \{(d,\varphi,Inv(l)[\varrho][\rho]) \in \mathcal{O}\}.$$

We define $\mathbf{out}(\mathcal{Q}) =_{def} \bigcup_{(l,\varphi,\rho)\in\mathcal{Q}} \mathbf{out}((l,\varphi,\rho))$, *for* $\mathcal{Q}$ *a set of symbolic states.*

The following definition of *stioco* is in essence very similar to the definition of *tioco*: there, an implementation conforms to the specification *w.r.t. tioco*, if, whenever the implementation, after the execution of a certain timed trace σ, produces an output of a certain kind or a delay of a certain length, then also the

specification, after the execution of σ, must be able to produce the same output or delay. In the case of *stioco*, however, this condition is expressed logically in terms of an FO formula, which says the following: the implementation conforms to the specification *w.r.t. stioco*, if, whenever the logical conditions are satisfied for the implementation to produce an output or delay, then also the specification satisfies the conditions to produce the same output or delay. It is therefore necessary to collect all conditions that lead to an observation. This is done by formula ϕ defined as follows.

Definition 10. *Let $\gamma \in \mathcal{G}_{U_d}$, $l \in L$, and σ a trace. Then $\phi(\gamma, l, \sigma)$ is an FO formula defined as*

$$\phi(\gamma, l, \sigma) =_{def} \bigvee \{\varphi \wedge \psi \mid (\gamma, \varphi, \psi) \in \mathbf{out}\,((l, \top, \mathtt{id})\ \mathbf{after}\ (\sigma, \top))\}.$$

Formula ϕ is a disjunction of all conditions that lead to an observation after a trace σ. There is a valuation such that (1) at least one sub-term $\varphi \wedge \psi$ of the disjunction is satisfied, (2) formula ϕ holds, and (3) we know that there is a least one symbolic observation that can be observed after executing σ.

The *tioco*-relation is defined for input-enabled implementations. We define an *STA* $\mathcal{S}$ to be input-enabled if and only if its semantics $[\![\mathcal{S}]\!]_\iota$ is an input-enabled *TLTS*.

Definition 11 (*stioco*). *Let $\mathcal{S}(\iota_s) = (L_s, l_s, \mathcal{V}_s, \mathcal{I}, \mathcal{G}, \mathcal{C}_\mathcal{S}, Inv, \rightarrow)$ be an initialised STA (the specification) $\mathcal{A}_\mathcal{S}$, $\mathcal{F}_s \subseteq \mathcal{ET}races(\mathcal{S})$ and let $\mathcal{P}(\iota_p) = (L_p, l_p, \mathcal{V}_p, \mathcal{I}, \mathcal{G}, \mathcal{C}_\mathcal{P}, Inv, \rightarrow)$ be an input-enabled implementation given as an STA, with $\mathcal{V}_p \cap \mathcal{V}_s = \emptyset$ and $\mathcal{C} = \mathcal{C}_\mathcal{S} \cup \mathcal{C}_p$ the set of all clocks. $\mathcal{P}$ conforms to $\mathcal{S}$ with respect to $stioco_{\mathcal{F}_s}$ (written as $\mathcal{P}$ $stioco_{\mathcal{F}_s}$ $\mathcal{S}$,) if and only if the following holds. $\forall(\sigma, \chi) \in \mathcal{F}_s, \gamma \in \mathcal{G}_{U_d}$:*

$$(\iota_p)_{\mathcal{V}_p} \cup (\iota_s)_{\mathcal{V}_s} \cup [\mathcal{C} \mapsto 0] \models \underbrace{\bar{\forall}_{\widehat{\mathcal{H}} \cup \mathcal{H}}(\phi(l_p, \gamma, \sigma) \wedge \chi \rightarrow \phi(l_s, \gamma, \sigma))}_{(*)}.$$

The heart of this definition lies in the universally quantified formula $(*)$. At the symbolic level, we do not look at concrete outputs but at symbolic constraints defining a set of possible concrete actions. On the left-hand side of the implication we have for the implementation a conjunction of a disjunction of the symbolic constraints accumulated after trace σ and restriction χ which can be used to prune the symbolic execution. On the right-hand side, we have the disjunction of all accumulated symbolic constraints for the specification. The implication states that the constraints accumulated for the implementation imply the constraints accumulated by the specification.

Example 2. We consider two instances, $\mathcal{S}_1$ and $\mathcal{S}_2$, of the *STA* shown in Figure 3. Assuming the following instantiations for variable i: $i := 8$ for $\mathcal{S}_1$ and $i := 10$ for $\mathcal{S}_2$ we get that $\mathcal{S}_1$ *stioco* $\mathcal{S}_2$. Since whenever the conditions for $\mathcal{S}_1$ to produce an output or for delaying are satisfied the conditions for $\mathcal{S}_2$ are also satisfied. However this does not hold for $\mathcal{S}_2$ *stioco* $\mathcal{S}_1$. The condition for an output of location l_1 for *STA* $\mathcal{S}_2$ is $\phi_{\mathcal{S}_2} := (d_3 \leq 10 \wedge (x = \mathtt{money} \wedge (\mathtt{change} \geq \mathtt{q} \vee d_3 \geq 10)$ and $\phi_{\mathcal{S}_1} := (d_3 \leq 8) \wedge (x = \mathtt{money} \wedge (\mathtt{change} \geq \mathtt{q} \vee d_3 \geq 8))$ for $\mathcal{S}_1$. Therefore $\phi_{\mathcal{S}_2} \not\rightarrow \phi_{\mathcal{S}_1}$. Thus, formula $*$ in Definition 11 for *stioco* is violated.

The following theorem states that *stioco* corresponds to *tioco* on the semantical level. The proof of Theorem 3 is given in [20].

Theorem 3.
Let $\mathcal{P}(\iota_{\mathcal{P}})$ be an input-enabled initialised STA $\mathcal{S}(\iota_{\mathcal{S}})$ be an initialised STA. Let $\mathcal{F}_s$ be a set of delayed symbolic extended traces for $\mathcal{S}$. Then:

$$\mathcal{P}(\iota_{\mathcal{P}}) \ stioco_{\mathcal{F}_s} \ \mathcal{S}(\iota_{\mathcal{S}}) \ \Leftrightarrow \ [\![\mathcal{P}]\!]_{\iota_{\mathcal{P}}} tioco_{[\![\mathcal{F}_s]\!]_{\iota_{\mathcal{S}}}} [\![\mathcal{S}]\!]_{\iota_{\mathcal{S}}}.$$

6 Conclusions

We presented a symbolic framework for timed automata combined with symbolic transitions systems. We defined an implementation relation *stioco* on *STA* which coincides with *tioco* [16] on the semantical level. To define such an implementation relation we provided symbolic trace executions and symbolic observations. It must be noted that, since timed automata are a subclass of *STA*, *stioco* is also a symbolic implementation relation expressing *tioco* on a symbolic level for timed automata.

The interaction between time and data in this paper is restricted to the influence that data inputs can have on the timing behaviour of the considered *STA*. This was expressed by allowing location variables to serve as bounds in clock constraints and invariants. More and different interactions between time and data are imaginable, for example, by assigning clock valuations to location variables, *i.e.,* by keeping historic information about the occurrence time of events in the *STA*. In principle, this extension could also be encoded in the first-order logical framework. However, even for the more restricted case considered in this paper, it is necessary to investigate first whether the obtained formalism is not already too expressive to be useful for practical testing, in terms of decidability of the forward reachability problem. A suitable subclass of FO logic might have to be identified to ensure this and to be able to apply the provided theory on practical applications. This would encompass the development of an algorithm for automatic test-case generation and test-execution.

Our theory is restricted to systems without τ-transitions. Adding those would be straightforward, in a similar way as it has been done for *STS* in [9], although special care has to be taken to combine successive delays. This will be part of future investigations.

Acknowledgments. We thank Lars Frantzen for helpful discussions and a preliminary chapter of his PhD thesis on *STS*.

References

1. Alur, R., Dill, D.L.: A theory of timed automata. TCS 126(2), 183–235 (1994)
2. Belinfante, A.: JTorX: A tool for on-line model-driven test derivation and execution. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 266–270. Springer, Heidelberg (2010)

3. Belinfante, A., Feenstra, J., de Vries, R.G., Tretmans, J., Goga, N., Feijs, L.M.G., Mauw, S., Heerink, L.: Formal test automation: A simple experiment. In: IWTCS, pp. 179–196 (1999)
4. Bohnenkamp, H., Belinfante, A.: Timed testing with TorX. In: Fitzgerald, J., Hayes, I.J., Tarlecki, A. (eds.) FM 2005. LNCS, vol. 3582, pp. 173–188. Springer, Heidelberg (2005)
5. Brandán Briones, L., Brinksma, H.: A test generation framework for *quiescent* real-time systems. In: Grabowski, Nielsen (eds.) [10], pp. 64–78
6. Clarke, D., Jéron, T., Rusu, V., Zinovieva, E.: STG: a symbolic test generation tool. In: Katoen, J.-P., Stevens, P. (eds.) TACAS 2002. LNCS, vol. 2280, p. 470. Springer, Heidelberg (2002)
7. Frantzen, L., Huerta, M.N., Kiss, Z.G., Wallet, T.: On-The-Fly Model-Based Testing of Web Services with Jambition. In: Bruni, R., Wolf, K. (eds.) WS-FM 2008. LNCS, vol. 5387, pp. 143–157. Springer, Heidelberg (2009)
8. Frantzen, L., Tretmans, J., Willemse, T.A.C.: Test generation based on symbolic specifications. In: Grabowski, Nielsen (eds.) [10], pp. 1–15
9. Frantzen, L., Tretmans, J., Willemse, T.A.C.: A Symbolic Framework for Model-Based Testing. In: Havelund, K., Núñez, M., Roşu, G., Wolff, B. (eds.) FATES 2006 and RV 2006. LNCS, vol. 4262, pp. 40–54. Springer, Heidelberg (2006)
10. Grabowski, J., Nielsen, B. (eds.): FATES 2004. LNCS, vol. 3395. Springer, Heidelberg (2005)
11. Hartman, A., Nagin, K.: The AGEDIS tools for model based testing. SIGSOFT Softw. Eng. Notes 29(4), 129–132 (2004)
12. Henzinger, T.A., Nicollin, X., Sifakis, J., Yovine, S.: Symbolic model checking for real-time systems. Inf. and Comp. 111(2), 193–244 (1994)
13. Hessel, A., Larsen, K.G., Mikucionis, M., Nielsen, B., Pettersson, P., Skou, A.: Testing real-time systems using UPPAAL. In: Hierons, R.M., Bowen, J.P., Harman, M. (eds.) FORTEST 2000. LNCS, vol. 4949, pp. 77–117. Springer, Heidelberg (2008)
14. Jard, C., Jéron, T.: TGV: theory, principles and algorithms: A tool for the automatic synthesis of conformance test cases for non-deterministic reactive systems. J. STTS 7(4), 297–315 (2005)
15. Jéron, T.: Symbolic model-based test selection. In: Machado, P., Andrade, A., Duran, A. (eds.) SBMF 2008, pp. 17–32 (2008)
16. Krichen, M., Tripakis, S.: Black-box conformance testing for real-time systems. In: Graf, S., Mounier, L. (eds.) SPIN 2004. LNCS, vol. 2989, pp. 109–126. Springer, Heidelberg (2004)
17. Schmaltz, J., Tretmans, J.: On conformance testing for timed systems. In: Cassez, F., Jard, C. (eds.) FORMATS 2008. LNCS, vol. 5215, pp. 249–263. Springer, Heidelberg (2008)
18. STSimulator homepage, `http://www.cs.ru.nl/~lf/tools/stsimulator/`
19. Tretmans, J.: Test generation with inputs, outputs and repetitive quiescence. Software - Concepts and Tools 17(3), 103–120 (1996)
20. von Styp-Rekowski, S.: Towards a testing theory for timed symbolic systems. Diplomarbeit, RWTH Aachen University (November 2009), `http://moves.rwth-aachen.de/dl/vstyp/styp_da.pdf`
21. Rusu, V., du Bousquet, L., Jéron, T.: An approach to symbolic test generation. In: Grieskamp, W., Santen, T., Stoddart, B. (eds.) IFM 2000. LNCS, vol. 1945, pp. 338–357. Springer, Heidelberg (2000)

Author Index