Víctor Braberman
Laurent Fribourg (Eds.)

LNCS 8053

Formal Modeling and Analysis of Timed Systems

11th International Conference, FORMATS 2013
Buenos Aires, Argentina, August 2013
Proceedings

Springer

Víctor Braberman   Laurent Fribourg (Eds.)

Formal Modeling and Analysis of Timed Systems

11th International Conference, FORMATS 2013
Buenos Aires, Argentina, August 29-31, 2013
Proceedings

Springer

Volume Editors

Víctor Braberman
Universidad de Buenos Aires - CONICET
Facultad de Ciencias Exactas y Naturales
Departamento de Computación
Intendente Güiraldes 2160, Pabellón 1, Ciudad Universitaria
C1428EGA Buenos Aires, Argentina
E-mail: vbraber@dc.uba.ar

Laurent Fribourg
LSV, CNRS & ENS de Cachan
61, avenue du Président Wilson
94235 Cachan Cedex, France
E-mail: fribourg@lsv.ens-cachan.fr

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-40228-9 e-ISBN 978-3-642-40229-6
DOI 10.1007/978-3-642-40229-6
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2013944565

CR Subject Classification (1998): F.3, D.2, F.1, D.3, F.4, G.3

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

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

Printed on acid-free paper

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

Preface

This volume contains the papers presented at the 11th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS 2013), held during August 29–31, 2013, in Buenos Aires, Argentina.

Timing aspects of systems from a variety of computer science domains have been treated independently by diferent communities. Researchers interested in semantics, verification, and performance analysis study models such as timed automata and timed Petri nets, the digital design community focuses on propagation and switching delays, while designers of embedded controllers have to take account of the time taken by controllers to compute their responses after sampling the environment. Timing-related questions in these separate disciplines do have their particularities. However, there is a growing awareness that there are basic problems that are common to all of them. In particular, all these subdisciplines treat systems whose behavior depends on combinations of logical and temporal constraints; namely, constraints on the temporal distances between occurrences of events.

The aim of FORMATS is to promote the study of fundamental and practical aspects of timed systems, and to bring together researchers from diferent disciplines that share interests in 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 dierent 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 temporal constraints (scheduling, worst-case execution time analysis, optimization, model-checking, testing, constraint solving, etc.)

-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 telecommunications)

This year FORMATS received 41 submissions. Most submissions were reviewed by four Program Committee members. The committee decided to accept 18 papers for publication and presentation at the conference. The program also included an invited talk (together with CONCUR 2013):

Reinhard Wilhelm, Saarland University, Germany: "Performance Analysis: Multicores, multi problems!"

This time, FORMATS was co-located together with the International Conference on Concurrency Theory (CONCUR), the International Conference on Quantitative Evaluation of SysTems (QEST), and the International Symposium on Trustworthy Global Computing (TGC). We would like to thank the general organizers, in particular Pedro R. D'Argenio and Hernan Melgratti for their helpful cooperation.

We would like to thank all the authors for submitting their work to FORMATS. We wish to thank the invited speaker for accepting our invitation. We are particularly grateful to the Program Committee members and the other reviewers for their insightful and timely reviews of the submissions and the subsequent discussions, which were instrumental to getting such an attractive program.

Throughout the entire process of organizing the conference and preparing this volume, we used the EasyChair conference management system, which provided excellent support. Finally, we gratefully acknowledge the financial support provided by the Argentinian National Council of Research (CONICET), and the Mobility between Europe and Argentina applying Logics to Systems (MEALS), a mobility project financed by the 7th Framework programme under Marie Curie International Research Staff Exchange Scheme.

June 2013

Víctor Braberman
Laurent Fribourg

Organization

Program Committee

Erika Abraham	RWTH Aachen University, Germany
Víctor Braberman	FCEyN, UBA, CONICET, Argentina
Thomas Chatain	LSV - ENS de Cachan & CNRS, France
Alexandre David	CISS / Aalborg University, Denmark
Alexandre Donzé	UC Berkeley, USA
Georgios Fainekos	Arizona State University, USA
Jean-Marie Farines	UFSC, Brazil
Ansgar Fehnker	University of the South Pacific, Fiji
Goran Frehse	VERIMAG, France
Diego Garbervetsky	FCEyN, UBA, Argentina
Holger Giese	Hasso Plattner Institute at the University of Potsdam, Germany
Radu Grosu	Stony Brook University, USA
Martijn Hendriks	Radboud University Nijmegen, The Netherlands
Marcin Jurdzinski	University of Warwick, UK
Fribourg Laurent	LSV, France
Dejan Nickovic	Austrian Institute of Technology AIT, Austria
Wojciech Penczek	IPI PAN and University of Podlasie, Poland
Leila Ribeiro	Universidade Federal do Rio Grande do Sul, Brazil
Olivier H. Roux	IRCCyN/Ecole Centrale de Nantes, France
Jun Sun	Singapore University of Technology and Design
Paulo Tabuada	UCLA, USA
Stavros Tripakis	University of California, Berkeley, USA
Ashutosh Trivedi	University of Pennsylvania, USA
Enrico Vicario	Università di Firenze, Italy
Sergio Yovine	CONICET-UBA, Argentina

Additional Reviewers

André, Étienne
Baldissera, Fabio
Balkan, Ayca
Bartocci, Ezio
Beauquier, Joffroy
Bertrand, Nathalie
Bogomolov, Sergiy
Bollig, Benedikt
Bortolussi, Luca
Boucheneb, Hanifa
Carnevali, Laura
Cassez, Franck
Chen, Xin
Corzilius, Florian

D' Souza, Deepak
Derler, Patricia
Donaldson, Alastair
Dyck, Johannes
Faucou, Sebastien
Feo-Arenis, Sergio
Ferrere, Thomas
Forejt, Vojtech
Fruth, Matthias
Gaggi, Ombretta
Gui, Lin
Jansen, Nils
Jha, Sumit Kumar
Kandl, Susanne
Knapik, Michał
Krenn, Willibald
Lambers, Leen
Lime, Didier
Loup, Ulrich
Maasoumy, Mehdi
Melgratti, Hernan
Meski, Artur
Mikučionis, Marius
Moreaux, Patrice
Muniz, Marco
Neumann, Stefan
Niebert, Peter
Paolieri, Marco
Pavese, Esteban
Picaronny, Claudine
Polrola, Agata
Poulsen, Danny Bøgsted
Rungger, Matthias
S., Krishna
S., Akshay
Schapachnik, Fernando
Schupp, Stefan
Schwoon, Stefan
Seidner, Charlotte
Sproston, Jeremy
Stainer, Amélie
Stergiou, Christos
Szreter, Maciej
Tribastone, Mirco
Uchitel, Sebastian
Vogel, Thomas
Wojtczak, Dominik
Wätzoldt, Sebastian
Yan, Rongjie
Zhang, Shaojie

Table of Contents

Precise Robustness Analysis of Time Petri Nets with Inhibitor Arcs

Étienne André, Giuseppe Pellegrino*, and Laure Petrucci

Université Paris 13, Sorbonne Paris Cité, LIPN
93430, Villetaneuse, France

Abstract. Quantifying the robustness of a real-time system consists in measuring the maximum extension of the timing delays such that the system still satisfies its specification. In this work, we introduce a more precise notion of robustness, measuring the allowed variability of the timing delays in their neighbourhood. We consider here the formalism of time Petri nets extended with inhibitor arcs. We use the inverse method, initially defined for timed automata. Its output, in the form of a parametric linear constraint relating all timing delays, allows the designer to identify the delays allowing the least variability. We also exhibit a condition and a construction for rendering robust a non-robust system.

Keywords: Time Petri nets, Quantitative robustness, Parameter synthesis.

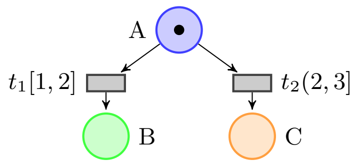
1 Introduction

Formalisms for modelling real-time systems, such as time Petri nets [10] or timed automata [3], have been extensively used in the past decades, and led to useful and efficient implementations. Time Petri nets (TPNs for short) are an extension of Petri nets where firing conditions are given in the form of intervals $[a, b]$. Each transition can only fire at least a time units and at most b time units after it is enabled. ITPNs extend TPNs with inhibitor arcs, i.e. arcs that disable their outgoing transition if their incoming place is not empty.

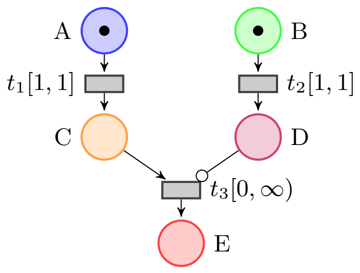
However, these formalisms allow for modelling in theory delays arbitrarily close (or even equal) to zero; this implies that the real system must be arbitrarily fast, which may be unrealistic in practice, where response times may not be neglected. These formalisms also allow for simultaneous occurrence of events, which may not be realistic in practice either, due to slightly different clock rates of several processors. And similarly, they allow for arbitrary precision, which is unrealistic: For example, a system where some component performs an action for e.g. 2 seconds can be implemented with a delay greater but very close to 2 (e.g. 2.0001 s), in which case the formal guarantee may not hold anymore.

The implementation in practice of a real-time system (modelled, e.g. by an ITPN) can lead in particular to two kinds of undesired consequences: the occurrence of behaviours that were proven impossible in theory, and the unlikely occurrence of behaviours that were proven possible in theory.

* This work is partially supported by an Erasmus grant.



(a) Example of undesired reachability



(b) Example of unlikely reachability

Fig. 1. Examples of non-robust ITPNs

Consider the simple TPN in Fig. 1a (from [2]). According to the semantics of TPNs (e.g. defined in [13]), place C is unreachable, that is, there exists no reachable marking such that the number of tokens in C is greater than 0. Indeed, starting from marking A (i.e. a marking with 1 token in place A), t_1 can fire anytime between 1 and 2 time units after the system start. At time 2, t_1 must fire if it has not yet fired, because its associated interval is about to expire and no other transition is firable (t_2 will be firable right after time 2). Hence, C is unreachable. Now suppose that the upper bound of the firing interval of t_1 is increased, even by an infinitesimal duration. Then, t_2 is firable immediately after time 2, and C can be reached in some executions.

Now consider the ITPN in Fig. 1b. According to the semantics of ITPNs, place E is reachable. Indeed, starting from a marking AB (i.e. a marking with 1 token in place A and 1 token in place B), t_1 can fire at time 1, giving marking CB. Then, after a null duration, t_3 can fire due to the absence of token in D. This sequence of transitions is unlikely to happen in practice due to delays exactly equal to zero; if the bounds of t_1 or the lower bound of t_3 become slightly larger, or the bounds of t_2 becomes slightly smaller, E becomes unreachable.

In this work, we use techniques based on parameter synthesis to compute a precise quantitative analysis of the admissible variability of the timing bounds of an ITPN with respect to linear-time properties. We use PITPNs, that is extensions of ITPNs where timing bounds are *parameters*, i.e. unknown constants. Our contributions are as follows:

1. We define the notion of *covering constraint* for parametric time Petri nets with inhibitor arcs (PITPNs), and characterise it;
2. We extend the *inverse method* to PITPNs (initially defined in the setting of parametric timed automata [4]), and prove that it preserves linear-time properties, based on the notion of covering constraint; and
3. We exploit the constraint output to obtain a precise quantitative measure of the system robustness for linear-time properties.

Given in the form of a constraint on the timing bounds seen as parameters, our robustness condition allows a designer (i) to relate the variability of the timing bounds with each other, (ii) to exhibit the critical timing bounds that do not allow any variability, and (iii) to render a system robust under certain conditions.

Related Work. Robustness in the setting of timed automata has received much attention in the past decade (see [9] for a survey). Most previous works (see e.g. [5,9,8,2,12,6]) consider that all timing constraints can be enlarged by a single very small (but positive) variation Δ. This robustness condition considers a unique positive parameter Δ; hence, roughly speaking, the robustness is guaranteed as long as the different clocks remain in intervals $[a - \Delta, b + \Delta]$ instead of $[a, b]$. In a geometrical context, the admissible variability can be seen as a simple hypercube (called "Δ-cube" from now on) in $2*n$ dimensions, with n the number of timing constraints. In contrast, we give a precise measure of the robustness, by considering possible local variations of each lower and upper bound of the firing intervals of a time Petri net. This is given in the form of a *polyhedron* in $2 * n$ dimensions, where n is the number of transitions. Hence, each bound can vary independently of the others. Our approach has the following advantages: (1) it identifies the most critical interval bounds, and helps the designer in tuning them (when possible) so that the system becomes robust; (2) it relates bounds in a parametric way, identifying bounds that should, for example, remain smaller than others; (3) it also outputs a constraint even when some bounds cannot tolerate any variation, whereas Δ-based approaches would just classify the system as non-robust (i.e. synthesise a $\Delta = 0$). Since parameter synthesis is undecidable for PITPNs [13], our algorithm may not terminate in the general case; however, we give sufficient termination conditions for subclasses of PITPNs.

In [6], it is shown that parameterised robust reachability in timed automata is decidable, again for a single Δ. In [8], computing the greatest acceptable variation Δ is proven decidable for flat timed automata with progressive clocks. In [12], a counter-example refinement approach is used with parametric techniques to evaluate the greatest acceptable variation Δ for parametric timed automata (although not decidable in the general case). These works share similarities with ours in the problem addressed and in the use of parametric techniques. However, beyond the fact that these works consider (a restriction of) timed automata whereas we consider (an extension of) time Petri nets, the main difference lies in the number of dimensions, since they all consider a simple Δ.

Recent work also considered robustness issues in time Petri nets. In [2], the quantification of robustness is performed by considering that the firing intervals can be enlarged by a (positive) parameter. Two problems are considered: the robust boundedness of the net (a bounded net remains bounded even in presence of small time variations) and the robust untimed language preservation (the untimed language remains preserved in presence of small time variations). Our work is close to [2], with notable differences. First, we use here a technique based on parameter synthesis. Second, we give a condition for trace preservation, where traces are defined as alternating markings and actions. Hence, the robustness condition in our work is different from the boundedness and language preservation of [2]. Last but not least, the robustness condition in [2] again considers a unique positive parameter Δ, whereas we compute a polyhedron in $2 * n$ dimensions. In [1], a more general notion of robustness is used for time Petri nets, that includes not only a robustness with respect to time, but also with constraints on the resources (e.g. memory), scheduling schemes (in a multi-processor environment) and possible system failures.

Outline. Section 2 recalls PITPNs and related results. In Section 3, we introduce and characterise covering constraints. In Section 4, we introduce the inverse method for PITPNs and prove its correctness. In Section 5, we exhibit a precise quantitative measure of the system robustness, and use it to turn some non-robust systems robust. We give directions of future research in Section 6.

2 Preliminaries

We denote by $\mathbb{N}$, $\mathbb{Q}_+$ and $\mathbb{R}_+$ the sets of non-negative integers, non-negative rational and non-negative real numbers, respectively.

2.1 Firing Times, Parameters and Constraints

Throughout this paper, we assume a set $\{\theta_1, \theta_2, \dots\}$ of *firing times.* A *firing time* is a variable with value in $\mathbb{R}_+$, encoding the time remaining before a given transition fires. In the following, Θ will denote a finite set $\{\theta_1, \dots, \theta_H\}$ of firing times, for some $H \in \mathbb{N}$. A *firing time valuation* is a function $\nu : \Theta \to \mathbb{R}_+^H$ assigning a non-negative real value with each firing time.

We also assume a set $\{\lambda_1, \lambda_2, \dots\}$ of *parameters*, i.e. unknown constants. In the following, $\Lambda = \{\lambda_1, \dots, \lambda_l\}$ denotes a finite set of parameters for some $l \in \mathbb{N}$. A *parameter valuation* π is a function $\pi : \Lambda \to \mathbb{R}_+$ assigning with each parameter a value in $\mathbb{R}_+$. A valuation π can be seen as a point $(\pi(\lambda_1), \dots, \pi(\lambda_l))$.

Constraints are defined as a set of inequalities. A (linear) inequality over Θ and Λ is $lt \prec lt'$, where $\prec \in \{<, \leq\}$, and lt, lt' are two linear terms of the form $\sum_{1 \leq i \leq N} \alpha_i z_i + d$ where $z_i \in \Theta \cup \Lambda$, $\alpha_i \in \mathbb{Q}_+$, for $1 \leq i \leq N$, and $d \in \mathbb{Q}_+$. We define similarly inequalities over Θ (resp. Λ). A constraint is a conjunction of inequalities. In particular, a constraint over the parameters can be seen as a polyhedron in l dimensions. We denote by $\mathcal{L}(\Lambda)$ the set of all constraints over the parameters. In the sequel, J denotes an inequality over the parameters, E a constraint over the firing times, K a constraint over the parameters, and D a constraint over firing times and parameters. Often, given a PITPN transition t_i, we will denote its parametric lower and upper bounds by λ_i^- and λ_i^+, respectively.

Given an inequality J of the form $lt < lt'$ (respectively $lt \leq lt'$), the *negation* of J, denoted by $\neg J$, is the inequality $lt' \leq lt$ (respectively $lt' < lt$).

Given a constraint E and a firing time valuation ν, $[\![E]\!]_\nu$ denotes the expression obtained by replacing each firing time θ in E with $\nu(\theta)$. A firing time valuation ν *satisfies* constraint E (denoted by $\nu \models E$) if $[\![E]\!]_\nu$ evaluates to true.

Given a parameter valuation π and a constraint D, $[\![D]\!]_\pi$ denotes the constraint over Θ obtained by replacing each parameter λ in D with $\pi(\lambda)$. Likewise, given a firing time valuation ν, $[\![[\![D]\!]_\pi]\!]_\nu$ denotes the expression obtained by replacing each firing time θ in $[\![D]\!]_\pi$ with $\nu(\theta)$. We say that a parameter valuation π *satisfies* a constraint D, denoted by $\pi \models D$, if the set of firing time valuations that satisfy $[\![D]\!]_\pi$ is non-empty.

A parameter valuation π *satisfies* a constraint K over the parameters, denoted by $\pi \models K$, if the expression obtained by replacing each parameter λ in K with $\pi(\lambda)$ evaluates to true. Given two constraints K_1 and K_2, K_1 is *included in* K_2, denoted by $K_1 \subseteq K_2$, if $\forall \pi : \pi \models K_1 \Rightarrow \pi \models K_2$. We consider `true` as a constraint over Λ, corresponding to the set of all possible values for Λ.

We denote by $D{\downarrow_\Lambda}$ the constraint over Λ obtained by projecting D onto Λ, i.e. after elimination of the firing times. Formally, $D{\downarrow_\Lambda} = \{\pi \mid \pi \models D\}$.

We finally define intervals as in [13]. An interval I of $\mathbb{R}_+$ is a $\mathbb{Q}_+$-interval if its left endpoint ${}^{\uparrow}I$ belongs to $\mathbb{Q}_+$ and its right endpoint $I^{\uparrow}$ belongs to $\mathbb{Q}_+ \cup \{\infty\}$. We denote by $\mathcal{I}(\mathbb{Q}_+)$ the set of $\mathbb{Q}_+$-intervals of $\mathbb{R}_+$. A parametric time interval is a function $J : {\mathbb{Q}_+}^{\Lambda} \to \mathcal{I}(\mathbb{Q}_+)$ that associates with each parameter valuation a $\mathbb{Q}_+$-interval. The set of parametric time intervals over Λ is denoted by $\mathcal{J}(\Lambda)$. As for I, we define ${}^{\uparrow}J$ and $J^{\uparrow}$ as the minimum and maximum bounds of J, respectively. They can both be represented using a constraint over Λ.

2.2 Parametric Time Petri Nets with Inhibitor Arcs

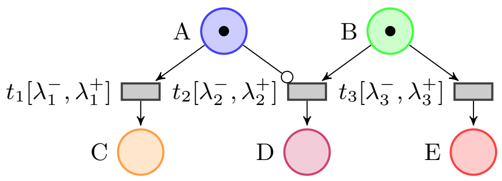
Parametric time Petri nets with inhibitor arcs (PITPNs) are a parametric extension of ITPNs, where the temporal bounds of the transitions can be parameters. We slightly adapt the notations defined in [13] to fit our setting.

Definition 1. *A parametric time Petri nets with inhibitor arcs (PITPN) is a tuple* $\mathcal{N} = \langle P, T, \Lambda, {}^{\bullet}(.), (.)^{\bullet}, (.)^{\circ}, M_0, J_s, K_0\rangle$ *where*

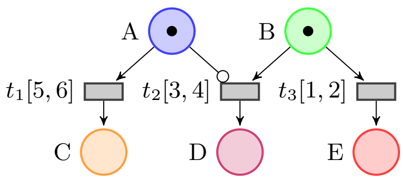
- $P = \{p_1, \dots, p_m\}$ *is a non-empty finite set of* places*,*
- $T = \{t_1, \dots, t_n\}$ *is a non-empty finite set of* transitions*,*
- $\Lambda = \{\lambda_1, \dots, \lambda_l\}$ *is a finite set of* parameters*,*
- ${}^{\bullet}(.)$ *(resp.* $(.)^{\bullet}$*)* $\in (\mathbb{N}^P)^T$ *is the* backward *(resp.* forward*)* incidence function*,*
- $(.)^{\circ} \in (\mathbb{N}^P)^T$ *is the* inhibition function*,*
- $M_0 \in \mathbb{N}^P$ *is the* initial marking*,*
- $J_s \in \mathcal{J}(\Lambda))^T$ *is the function that associates a* parametric firing interval *with each transition, and*
- $K_0 \in \mathcal{L}(\Lambda)$ *is the* initial constraint *over* Λ*.*

K_0 is a constraint over Λ giving the initial domain of the parameters, and must at least specify that the minimum bounds of the firing intervals are lower than or equal to the maximum bounds. Additional linear constraints may of course be given. Sometimes, given a constraint K_0, we will denote a PITPN by $\mathcal{N}(K_0)$ when clear from the context, and to emphasise the value of K_0 in $\mathcal{N}$.

Given a PITPN $\mathcal{N}$ and a valuation π, we denote by $[\![\mathcal{N}]\!]_\pi$ the (non-parametric) ITPN where each occurrence of a parameter has been replaced by its constant value as in π. Formally, given $\mathcal{N} = \langle P, T, \Lambda, {}^{\bullet}(.), (.)^{\bullet}, (.)^{\circ}, M_0, J_s, K_0\rangle$, then $[\![\mathcal{N}]\!]_\pi = \langle P, T, \Lambda, {}^{\bullet}(.), (.)^{\bullet}, (.)^{\circ}, M_0, J_s, K_0 \wedge K_\pi\rangle$, where $K_\pi = \bigwedge_{\lambda \in \Lambda} \big(\lambda = \pi(\lambda)\big)$. For example, the ITPN in Fig. 2b corresponds to the PITPN in Fig. 2a valuated with $\pi = \{\lambda_1^- \to 5, \lambda_1^+ \to 6, \lambda_2^- \to 3, \lambda_2^+ \to 4, \lambda_3^- \to 1, \lambda_3^+ \to 2\}$.


A
B
$t_1[\lambda_1^-, \lambda_1^+]$
$t_2[\lambda_2^-, \lambda_2^+]$
$t_3[\lambda_3^-, \lambda_3^+]$
C
D
E

(a) A PITPN $\mathcal{N}$



(b) A valuated (P)ITPN $[\![\mathcal{N}]\!]_\pi$

Fig. 2. A PITPN and its valuation

Semantics. We mostly reuse here the definitions and semantics from [13]. The reachable states of a PITPN are *parametric state-classes* (or simply *classes*), i.e. pairs $c = (M, D)$ where M is a marking of the net and D is a parametric firing domain, that is, a constraint over Θ and Λ. Given a class $c = (M, D)$, a transition t is *enabled* in c if $M \geq {}^\bullet t$ (i.e. if the number of tokens in M in each input place of t is greater than or equal to the value on the arc between this place and the transition). Transition t is *inhibited* if the place connected to one of its inhibitor arc is marked with at least as many tokens than the weight of the considered inhibitor arc between this place and t. Transition t is *active* if it is enabled and not inhibited. Transition t is *firable* if it has been active for at least ${}^\uparrow J_s(t)$ time units.

For a given class, the firing times in Θ correspond to variables encoding the time remaining before an active transition can fire. Hence, these variables *decrease* with time. The initial class of $\mathcal{N}(K)$ is $c_0 = (M_0, D_0)$, with $D_0 = K \wedge \{\theta_k \in J_s(t_k) | (t_k \in \textit{enabled}(M_0)\}$, where $\textit{enabled}(M_0)$ denotes the enabled transitions in M_0. For example, suppose that $K = \lambda_1^- \leq \lambda_1^+ \wedge \lambda_2^- \leq \lambda_2^+ \wedge \lambda_3^- \leq \lambda_3^+$; then the initial class of $\mathcal{N}$ in Fig. 2a is:

$$c_0 = (AB, \lambda_1^- \leq \theta_1 \leq \lambda_1^+ \wedge \lambda_2^- \leq \theta_2 \leq \lambda_2^+ \wedge \lambda_3^- \leq \theta_3 \leq \lambda_3^+).$$

We consider a (classical) semantics where a transition must fire before its upper interval bound, unless another transition fires first and disables it; for example, in Fig. 2a, t_1 must fire before t_3 if $\lambda_1^+ < \lambda_3^-$, t_3 must fire before t_1 if $\lambda_3^+ < \lambda_1^-$, and both orders are possible otherwise. Given a class $c = (M, D)$ and a firable transition t_f, $c' = (M', D')$ can be reached from c in one step via transition t_f (denoted by $c \overset{t_f}{\Rightarrow} c'$) if the following holds:

- $M' = M - {}^\bullet t_f + t_f^\bullet$
- D' is computed along the following steps:
  1. intersection with the firability constraints: $\forall j$ s.t. t_j is active, $\theta_f \leq \theta_j$,
  2. variable substitutions for all enabled transitions t_j that are active, i.e. $\theta_j = \theta_f + \theta'_j$,
  3. elimination (using for instance the Fourier-Motzkin method) of all variables relative to transitions disabled by the firing of t_f,

4. addition of inequalities relative to newly enabled transitions[1]: $\forall t_k \in$ *NewlyEnabled*$(M, t_f), {}^{\uparrow}J_s(t_k) \leq \theta'_k \leq J_s(t_k)^{\uparrow}$, with *NewlyEnabled*$(M, t_f)$ denoting the set of transitions newly enabled by firing the transition t_f from marking M.

The full semantics can be found in [13].

A *run* of $\mathcal{N}$ is a sequence $c_0 \overset{t_0}{\Rightarrow} \cdots \overset{t_{n-1}}{\Rightarrow} c_n$. Given a run r of $\mathcal{N}$ of the form $(M_0, D_0) \overset{t_0}{\Rightarrow} \cdots \overset{t_{n-1}}{\Rightarrow} (M_n, D_n)$, the *trace associated with* r is the alternating sequence of markings and actions $M_0 \overset{t_0}{\Rightarrow} \cdots \overset{t_{n-1}}{\Rightarrow} M_n$. The *trace set* of $\mathcal{N}$ is the set of all traces associated with the runs of $\mathcal{N}$. This corresponds to the discrete (or time-abstract) behaviour of $\mathcal{N}$. $\mathit{Post}_{\mathcal{N}(K)}(C)$ (resp. $\mathit{Post}^i_{\mathcal{N}(K)}(C)$) is the set of classes reachable from a set C of classes in exactly one step (resp. i steps) in $\mathcal{N}(K)$. Furthermore, we define $\mathit{Post}^*_{\mathcal{N}(K)}(C)$ as $\bigcup_{i\geq 0} \mathit{Post}^i_{\mathcal{N}(K)}(C)$. We define $\mathit{Reach}(\mathcal{N}(K))$ as the set of reachable classes of $\mathcal{N}(K)$, that is $\mathit{Post}^*_{\mathcal{N}(K)}(\{c_0\})$. Finally, we define $\mathcal{G}(\mathcal{N}(K))$ as the parametric reachability graph of $\mathcal{N}(K)$, that is the set of reachable parametric state-classes with the transition relation $\Rightarrow$.

Results. The following lemma, recalled from [13], states that the projection onto the parameters of the constraint associated with a class always gets stronger (i.e. more restricted) along a run of the system.

Lemma 1 (Lemma 14 in [13]). *Given a PITPN $\mathcal{N}$, let $c = (M, D)$ and $c' = (M', D')$ be two classes in $\mathcal{G}(\mathcal{N})$. If $c \overset{t}{\Rightarrow} c'$, then $D'{\downarrow_\Lambda} \subseteq D{\downarrow_\Lambda}$.*

The following result states that the valuation with π of a class c of $\mathcal{N}$ belongs to the graph of $\mathcal{N}$ valuated with π if and only if π belongs to the constraint associated with c.

Theorem 1 (Theorems 12 and 13 in [13]). *Given a PITPN $\mathcal{N}(K)$ and a valuation $\pi \models K$, let $c = (M, D)$ be a class in $\mathcal{G}(\mathcal{N}(K))$. Then: $[\![c]\!]_\pi \in \mathcal{G}([\![\mathcal{N}]\!]_\pi)$ iff $\pi \models D{\downarrow_\Lambda}$.*

3 Covering Constraint

We introduce the notion of covering constraint as the constraint resulting from the intersection of the projection onto the parameters of the constraints associated with all the reachable classes of a PITPN.

Definition 2. *Let $\mathcal{N}$ be a PITPN. The* covering constraint *of $\mathcal{N}$ is:*
$\bigcap_{(M,D)\in \mathit{Reach}(\mathcal{N})} D{\downarrow_\Lambda}$.

In the general case, it is possible that the covering constraint of a PITPN will be empty, due to the intersection of disjoint constraints over the parameters. But in the setting of the inverse method (see Section 4), it will not be.

The following lemma relates parametric and non-parametric runs, and derives from Theorem 1.

[1] For sake of simplicity, we only consider here closed intervals of the form $[a, b]$. For open intervals (e.g. $(2, 3]$ in Fig. 1a), one should use strict instead of large inequalities.

Lemma 2. *Let $\mathcal{N}$ be a PITPN, let π be a parameter valuation. Let r be a run of $\mathcal{N}$ reaching a class (M, D) in $\mathcal{G}(\mathcal{N})$. Then there exists an equivalent run in $[\![\mathcal{N}]\!]_\pi$ reaching class $(M, [\![D]\!]_\pi)$ in $\mathcal{G}([\![\mathcal{N}]\!]_\pi)$ iff $\pi \models D{\downarrow_\Lambda}$.*

Proof. Let $(M_0, D_0) \overset{t_0}{\Rightarrow} \dots \overset{t_{k-1}}{\Rightarrow} (M_k, D_k)$ be a run of $\mathcal{N}$. From Theorem 1, we have that $[\![(M_k, D_k)]\!]_\pi \in [\![\mathcal{G}(\mathcal{N}(K))]\!]_\pi$ iff $\pi \models D_k{\downarrow_\Lambda}$. Now consider transition $(M_{k-1}, D_{k-1}) \overset{t_{k-1}}{\Rightarrow} (M_k, D_k)$ in $\mathcal{G}(\mathcal{N})$. Then, from the semantics of PITPNs, for all $\pi \models [\![D_k]\!]_\pi$, then $(M_{k-1}, [\![D_{k-1}]\!]_\pi) \overset{t_{k-1}}{\Rightarrow} (M_k, [\![D_k]\!]_\pi) \in \mathcal{G}([\![\mathcal{N}]\!]_\pi)$. The result then derives from a reasoning by induction on k, with $(M, D) = (M_k, D_k)$. $\square$

Conversely, the following lemma states that, given a PITPN $\mathcal{N}$, a run in a valuation of $\mathcal{N}$ always has an equivalent run in $\mathcal{N}$.

Lemma 3. *Let $\mathcal{N}(K)$ be a PITPN, let π be a parameter valuation such that $\pi \models K$. Let r be a run of $[\![\mathcal{N}]\!]_\pi$. Then there exists an equivalent run in $\mathcal{N}(K)$.*

Proof. $[\![\mathcal{N}]\!]_\pi$ can be seen as a PITPN (hence parametric) with an initial constraint K_π. Since $K_\pi \subseteq K$, from the semantics of PITPNs, the set of behaviours of $\mathcal{N}(K)$ includes the behaviours of $\mathcal{N}(K_\pi)$. Hence any run in $\mathcal{N}(K_\pi)$ has an equivalent in $\mathcal{N}(K)$. $\square$

We now state below a general result that will be used to prove Lemma 5.

Lemma 4. *Let $\mathcal{N}(K)$ be a PITPN. Then for all $(M, D) \in \mathcal{G}(\mathcal{N}(K))$, $D{\downarrow_\Lambda} \subseteq K$.*

Proof. By induction on Lemma 1, with $K_0 \subseteq K$ as the base case. $\square$

The following result states that, for a PITPN with its own covering constraint K_{cov} as initial constraint, the projection onto the parameters of the constraint associated with a reachable class is always the same, and equal to K_{cov}.

Lemma 5. *Let $\mathcal{N}(K)$ be a PITPN, let K_{cov} be the covering constraint of $\mathcal{N}(K)$. Then for all $(M, D) \in \mathcal{G}(\mathcal{N}(K_{cov}))$: $D{\downarrow_\Lambda} = K_{cov}$.*

Proof. If K_{cov} is empty, $\mathcal{G}$ is empty too and the result trivially holds. Suppose K_{cov} is non-empty. Let $c = (M, D) \in \mathcal{G}(\mathcal{N}(K_{cov}))$. Let $\pi \models D{\downarrow_\Lambda}$. By Lemma 4, $D{\downarrow_\Lambda} \subseteq K_{cov}$. By construction of K_{cov}, we have that $K_{cov} \subseteq K$. Hence $\pi \models D{\downarrow_\Lambda} \Rightarrow \pi \models K$. Since $\pi \models D{\downarrow_\Lambda}$, from Lemma 2, there exists an equivalent run in $[\![\mathcal{N}]\!]_\pi$ reaching class $(M, [\![D]\!]_\pi)$ in $\mathcal{G}([\![\mathcal{N}]\!]_\pi)$. Since $\pi \models K$, from Lemma 3, there exists an equivalent run in $\mathcal{N}(K)$ reaching class (M, D') for some D'.

Let $\pi' \models K_{cov}$. By construction, $K_{cov} \subseteq D'{\downarrow_\Lambda}$, hence $\pi' \models D'{\downarrow_\Lambda}$. By Lemma 4, $D{\downarrow_\Lambda} \subseteq K$, hence $\pi' \models K$. Since $\pi' \models K$ and $\pi' \models D'{\downarrow_\Lambda}$, applying Theorem 1 to $\mathcal{N}(K)$ gives that $[\![c]\!]_{\pi'} \in \mathcal{G}([\![\mathcal{N}]\!]_{\pi'})$. Since $\pi' \models K_{cov}$ by hypothesis, and $[\![c]\!]_{\pi'} \in \mathcal{G}([\![\mathcal{N}]\!]_{\pi'})$, then applying Theorem 1 to $\mathcal{N}(K_{cov})$ gives that $\pi' \models D{\downarrow_\Lambda}$. Hence $K_{cov} \subseteq D{\downarrow_\Lambda}$. (Lemma 4 gives the other direction.) $\square$

Finally, Theorem 2 states that the trace set of a PIPTN valuated with any parameter valuation satisfying its covering constraint K_{cov} is the same as the trace set of this PITPN with K_{cov} as initial constraint.

Theorem 2. *Let $\mathcal{N}$ be a PITPN, let K_{cov} be the covering constraint of $\mathcal{N}$. Let $\pi \models K_{cov}$. Then the trace sets of $\mathcal{N}(K_{cov})$ and $[\![\mathcal{N}]\!]_\pi$ are equal.*

Proof. Let $\pi \models K_{cov}$. Consider a run of $\mathcal{N}(K_{cov})$ reaching a class (M, D) in $\mathcal{G}(\mathcal{N}(K_{cov}))$. By Lemma 5, it holds that $D{\downarrow}_\Lambda = K_{cov}$. Since $\pi \models K_{cov}$, then $\pi \models D{\downarrow}_\Lambda$. Hence, by Lemma 2, there exists an equivalent run in $\mathcal{G}([\![\mathcal{N}]\!]_\pi)$. Conversely, since $\pi \models K_{cov}$, by lemma 3, any run in $[\![\mathcal{N}]\!]_\pi$ has an equivalent run in $\mathcal{N}(K_{cov})$. □

We can derive from Theorem 2 that the trace set of a PIPTN with any parameter valuation satisfying its covering constraint is always the same. This result will be used to prove the correctness of the inverse method (see Section 4).

Corollary 1. *Let $\mathcal{N}$ be a PITPN, let K_{cov} be the covering constraint of $\mathcal{N}$. Then for all $\pi, \pi' \models K_{cov}$, the trace sets of $[\![\mathcal{N}]\!]_\pi$ and $[\![\mathcal{N}]\!]_{\pi'}$ are equal.*

4 The Inverse Method for Time Petri Nets

We extend to PITPNs the inverse method initially proposed for timed automata [4]. The algorithm relies on the following definition of π-compatibility.

Definition 3. *Given a parameter valuation π, a class (M, D) is said to be π-*compatible *if $\pi \models D{\downarrow}_\Lambda$, and π-incompatible otherwise.*

4.1 Principle

We introduce in Algorithm 1 *IMPN* (i.e. the Inverse Method for time Petri Nets with inhibitor arcs). It uses 3 variables: an integer i measuring the depth of the state space exploration, the current constraint K_c, and the set C of explored classes. Starting from the initial class c_0, *IMPN* iteratively computes classes. When a π-incompatible class is found, an incompatible inequality is nondeterministically selected within the projection of the constraint onto Λ (line 5);

Algorithm 1: *IMPN*$(\mathcal{N}, \pi)$

```
input  : PITPN $\mathcal{N}$ of initial class $c_0$ and initial constraint $K_0$, valuation $\pi$
output: Constraint $K_r$
1 $i \leftarrow 0$ ;  $K_c \leftarrow K_0$ ;  $C \leftarrow \{c_0\}$
2 while true do
3     while $\exists$ $\pi$-incompatible classes in $C$ do
4         Select a $\pi$-incompatible class $(M, D)$ of $C$
5         Select a $\pi$-incompatible $J$ in $D{\downarrow}_\Lambda$
6         $K_c \leftarrow K_c \wedge \neg J$ ;  $C \leftarrow \bigcup_{j=0}^{i} Post^j_{\mathcal{N}(K_c)}(\{c_0\})$
7     if $Post_{\mathcal{N}(K_c)}(C) \subseteq C$ then return $K_r \leftarrow \bigcap_{(M,D) \in C} D{\downarrow}_\Lambda$
8     $i \leftarrow i + 1$ ;  $C \leftarrow C \cup Post_{\mathcal{N}(K_c)}(C)$
```

its negation is then added to K_c (line 6). The set of reachable classes is then updated. When all successor classes have already been reached (line 7), *IMPN* returns the intersection K_r of the projection onto Λ of the constraints associated with all the reachable classes.

4.2 Results

Lemma 6. *Let $\mathcal{N}$ be a PITPN, and π be a parameter valuation. Suppose that algorithm IMPN$(\mathcal{N}, \pi)$ terminates with output K_r. It holds that $\pi \models K_r$.*

Proof. By construction, at the end of the inner **while** loop, all classes of C are π-compatible, that is for all $(M, D) \in C, \pi \models D$. As a consequence, $\pi \models D{\downarrow}_\Lambda$. Recall that $K_r = \bigcap_{(M,D)\in C} D{\downarrow}_\Lambda$. Hence $\pi \models K_r$. □

The correctness of *IMPN* mainly relies on the fact that K_r is the covering constraint of $\mathcal{N}$. Hence, the results of Section 3 can be applied.

Theorem 3 (Correctness). *Let $\mathcal{N}$ be a PITPN, and π be a parameter valuation. Suppose IMPN$(\mathcal{N}, \pi)$ terminates with output K_r. Then:*

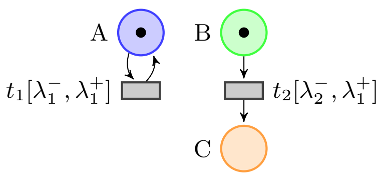
1. *$\pi \models K_r$, and*
2. *$\forall \pi' \models K_r$, $[\![\mathcal{N}]\!]_{\pi'}$ and $[\![\mathcal{N}]\!]_\pi$ have the same trace set.*

Proof. Item 1 comes from Lemma 6. For item 2, since K_r is the covering constraint of $\mathcal{N}$, then we can apply Corollary 1, which gives the result. Also note that the covering constraint cannot be empty since $\pi \models K_r$. □

Non-termination. Parameter synthesis is undecidable for PITPNs [13] and *IMPN* may not always terminate. Consider the PITPN $\mathcal{N}$ in Fig. 3a; then, *IMPN* applied to $\mathcal{N}$ and a reference valuation with all parameters equal to 0 will generate an infinite set of classes with constraints of the form $i * \lambda_1^- \leq \lambda_2^+$, with i infinitely growing. Intuitively, t_1 can fire an arbitrary number of times before t_2 fires. Of course, this is a typical Zeno-behaviour (an infinite number of transitions within a null duration) and, in the case of non-null reference parameter valuations, an inequality $i * \lambda_1^- \leq \lambda_2^+$ will eventually be π-incompatible, thus ensuring termination. Also note $\mathcal{N}$ is a bounded L/U (lower/upper bounds) PTPN [13], showing that termination of *IMPN* is not guaranteed for general bounded L/U PTPNs (although emptiness and reachability problems are decidable in theory). Studying the decidability of this problem, and adapting *IMPN* to ensure termination in this case is the subject of ongoing work.

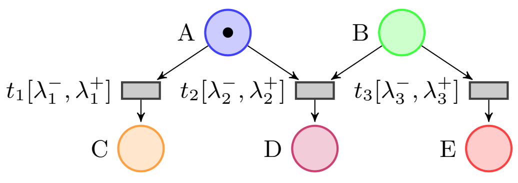
We can exhibit subclasses for which *IMPN* terminates. This is obviously the case of loopless PITPNs (in which no syntactical loop exists in the model). This is also the case of parametric sequential TPNs [2]; this subclass of TPNs is such that each time a discrete transition is fired, each transition that is enabled in the new/resulting marking is newly enabled. Hence, the problem of infinitely concurrent loops such as in Fig. 3a cannot happen.

Non-confluence and Non-completeness. Due to the non-deterministic selection of an inequality, *IMPN* is non-confluent (i.e. different applications of the



A
B
C
$t_1[\lambda_1^-, \lambda_1^+]$
$t_2[\lambda_2^-, \lambda_1^+]$

(a) Non-termination



A
B
C
D
E
$t_1[\lambda_1^-, \lambda_1^+]$
$t_2[\lambda_2^-, \lambda_2^+]$
$t_3[\lambda_3^-, \lambda_3^+]$

(b) Non-completeness

Fig. 3. Counter-examples PITPNs

algorithm can yield different outputs). As a consequence, it is also non-complete (i.e. the resulting constraint may not be the maximal one). Formally:

Proposition 1 (Non-completeness). *There may exist* $\pi' \not\models K_r$ *such that* $[\![\mathcal{N}]\!]_{\pi'}$ *and* $[\![\mathcal{N}]\!]_{\pi}$ *have the same trace set.*

An example for non-completeness is the PITPN $\mathcal{N}$ in Fig. 3b, with the reference parameter valuation $\pi = \{\lambda_1^- \to 5, \lambda_1^+ \to 6, \lambda_2^- \to 1, \lambda_2^+ \to 3, \lambda_3^- \to 2, \lambda_3^+ \to 4\}$. In $[\![\mathcal{N}]\!]_{\pi}$, either t_2 or t_3 can fire first, but not t_1, due to the fact that we have both ${}^{\uparrow}I(t_1) > I^{\uparrow}(t_2)$ and ${}^{\uparrow}I(t_1) > I^{\uparrow}(t_3)$.

When applying the inverse method to $\mathcal{N}$ and π, a class will be generated with BC as a marking, and an associated constraint containing in particular inequalities $\lambda_1^- \leq \lambda_2^+ \wedge \lambda_1^- \leq \lambda_3^+$. Since both are π-incompatible, the algorithm can add to the current constraint either $\lambda_1^- > \lambda_2^+$ or $\lambda_1^- > \lambda_3^+$. Either of them is sufficient to prevent BC to be reachable. Then the result of the application of *IMPN* to $\mathcal{N}$ and π is both non-confluent and non-complete. Also note that, due to the absence of inhibitor arc in $\mathcal{N}$, the non-completeness of *IMPN* also holds for PTPNs.

Nevertheless, it can be shown (as it was the case for timed automata [4]) that a sufficient (but non-necessary) condition for completeness is that *IMPN* does not perform non-deterministic selections of inequalities, i.e. at most one π-incompatible class is met at each iteration.

5 Precise Robustness Analysis

5.1 Local Robustness

Throughout this section, we assume an ITPN N, as well as a parameterised version $\mathcal{N}$ of N where each lower (resp. upper) bound of a transition t_i is replaced with a fresh parameter λ_i^- (resp. λ_i^+). Let π be the reference valuation such that $[\![\mathcal{N}]\!]_{\pi} = N$. We assume that $IMPN(\mathcal{N}, \pi)$ terminates with output K_r.

We will exploit K_r to characterise the precise robustness of the system, i.e. the admissible variability of each timing bound. The original trace set is preserved by any valuation satisfying K_r. Hence, any linear-time (LTL) property that is true in $[\![\mathcal{N}]\!]_{\pi}$ is also true in $[\![\mathcal{N}]\!]_{\pi'}$, for $\pi' \models K_r$. Thus, if the correctness is given in the form of an LTL property, the timing delays can safely vary as long as they satisfy K_r.

We use here several examples in order to better illustrate the notions. For the PITPN in Fig. 2a, with $\pi = \{\lambda_1^- \rightarrow 5, \lambda_1^+ \rightarrow 6, \lambda_2^- \rightarrow 3, \lambda_2^+ \rightarrow 4, \lambda_3^- \rightarrow 1, \lambda_3^+ \rightarrow 2\}$ as a reference valuation, *IMPN* outputs the constraint $K_r = \lambda_1^- \leq \lambda_1^+ \wedge \lambda_2^- \leq \lambda_2^+ \wedge \lambda_3^- \leq \lambda_3^+ \wedge \lambda_3^+ < \lambda_1^-$. For a parameterised version of the ITPN in Fig. 1a, *IMPN* outputs the constraint $K_r = \lambda_1^- \leq \lambda_1^+ \wedge \lambda_2^- \leq \lambda_2^+ \wedge \lambda_2^- \geq \lambda_1^+$. For a parameterised version of the ITPN in Fig. 1b, *IMPN* outputs the constraint $K_r = \lambda_1^- \leq \lambda_1^+ \wedge \lambda_2^- \leq \lambda_2^+ \wedge \lambda_3^- = 0 \wedge 0 \leq \lambda_3^+ \wedge \lambda_1^+ = \lambda_2^-$.

Definition 4. *An ITPN N is* robust with respect to linear-time properties *(or* LT-robust*) if there exists $\gamma > 0$ such that for any linear time property φ, $N' \models \varphi$ if and only if $N \models \varphi$, where N' is an ITPN similar to N where each timing bound c can be replaced with any value within $[c - \gamma, c + \gamma]$.*

For example, the ITPN in Fig. 2a is LT-robust (with e.g. $\gamma = 1$), whereas the ITPNs in Fig. 1 are not.

Local Robustness. The resulting constraint K_r is given in the form of a convex (possibly unbounded) polyhedron. For each interval bound λ_i in $\mathcal{N}$, its *local robustness* $LR(\lambda_i)$ is defined as the distance between $\pi(\lambda_i)$ and the closest border of the polyhedral representation of K_r. For example, in Fig. 2a, $LR(\lambda_1^-) = 1$. In Fig. 1a, $LR(\lambda_1^-) = 1$ whereas $LR(\lambda_1^+) = 0$, showing that this latter bound renders the system non-robust. The following lemma follows from Definition 4, from the definition of LR and the correctness of *IMPN*.

Lemma 7. *If for each parameter λ in $\mathcal{N}$, $LR(\lambda) > 0$, then N is LT-robust.*

Ranging Interval. For each interval bound λ_i in $\mathcal{N}$, its *ranging interval* $RI(\lambda_i)$ is defined as its minimum and maximum admissible values within K_r. It is computed by valuating all parameters but λ_i in K_r, and converting the resulting inequality in the form of an interval. For example, in Fig. 2a, $RI(\lambda_1^-) = (2, 6]$. In Fig. 1a, $RI(\lambda_1^+) = [1, 2]$.

The local lower (resp. upper) variability is defined as the distance between the parameter valuation and the lower (resp. upper) bound of RI; formally, given $RI(\lambda_i) = (a, b)$, $LLV(\lambda_i) = \pi(\lambda_i) - a$ and $LUV(\lambda_i) = b - \pi(\lambda_i)$. Note that the local robustness can be obtained from the local variability: $LR(\lambda_i) = \min(LLV(\lambda_i), LUV(\lambda_i))$.

Computation of Δ. Our approach also allows to retrieve the value of the "Δ" of Δ-based approaches. It is defined as the minimum over the set of parameters of the distance between a parameter and the closest border of the polyhedron. Formally, $\Delta = \min\big(\min_{i \in \Delta^-} LLV(\lambda_i), \min_{i \in \Delta^+} LUV(\lambda_i)\big)$, where Δ^- (resp. Δ^+) denotes the set of parameters appearing in an interval lower (resp. upper) bound. This distinction is necessary, since Δ-based approaches only consider the positive enlarging of intervals. For the ITPN in Fig. 2a, the maximum possible Δ is 1.5 (see Section 5.3). And, obviously, $\Delta = 0$ for the ITPNs in Fig. 1.

5.2 Improving the System Robustness

Identifying Critical Timing Bounds. Our approach allows to exhibit *critical* timing bounds: critical timing bounds are those rendering the system non-robust, i.e. with a null local robustness. For example, in Fig. 1a, λ_1^+ and λ_2^- are the critical timing bounds. In Fig. 1b, λ_1^+, λ_2^- and λ_3^- are the critical timing bounds.

Relaxing Bounds. For some systems, it is possible to refine the values of the critical timing bounds so that the system becomes robust, with the same discrete behaviour. In practice, this may in particular be the case of hardware systems, where the timing bounds come from the traversal time of micro components: One can change the timing bounds by replacing a component with another one. In software, one can also refine the values of some timers if needed.

In that case, one can exploit the precise robustness analysis to synthesise values for the timing bounds so that the system is robust. A system is said to be potentially robust if all timing bounds λ_i have a ranging interval non-reduced to a point (even if their local robustness may possibly be null, i.e. $LR(\lambda_i) = 0$).

Definition 5. *An ITPN N is* potentially robust *if, for all timing bounds λ_i, $LLV(\lambda_i) \neq LUV(\lambda_i)$.*

This notion of potential robustness is a sufficient condition so that an ITPN becomes robust with the same discrete behaviour.

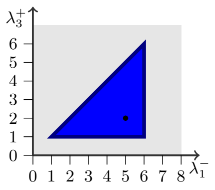
Theorem 4. *If N is potentially robust, then there exists π_R such that $[\![\mathcal{N}]\!]_{\pi_R}$ is LT-robust, and has the same trace set as N.*

Proof. By Lemma 7, only the timing bounds λ_i such that $LR(\lambda_i) = 0$ render $\mathcal{N}$ non-LT-robust. For all λ_i such that $LR(\lambda_i) > 0$, we set $\pi_R(\lambda_i) = \pi(\lambda_i)$. Now consider a λ_i such that $LR(\lambda_i) = 0$. By definition of LR, either $LLV(\lambda_i) = 0$ or $LUV(\lambda_i) = 0$. Consider the former case (the latter case is dual). Let $\pi_R(\lambda_i) = \pi(\lambda_i) + \big(LLV(\lambda_i) + LUV(\lambda_i)\big)/2$. Since $\mathcal{N}$ is potentially robust, $LLV(\lambda_i) \neq LUV(\lambda_i)$; hence $LLV(\lambda_i) < \pi_R(\lambda_i) < LUV(\lambda_i)$. As a consequence, in π_R, we have $LR(\lambda_i) > 0$. By construction, and from the convexity of K_r, $\pi_R(\lambda_i)$ is in K_r; hence, from Theorem 3, $[\![\mathcal{N}]\!]_{\pi_R}$ and $[\![\mathcal{N}]\!]_{\pi}$ have the same trace set. $\square$

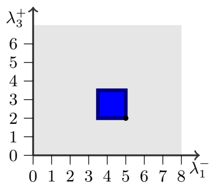
Note that this is a sufficient but non-necessary condition, since the notion of potential robustness is based on LLV and LUV, that come from K_r, which is non-complete. Furthermore, one can find further conditions (and constructions) to render a system robust. For example, the ITPN in Fig. 1b is not potentially robust; but it can be made robust with the same discrete behaviour, e.g. by replacing the intervals associated with both t_1 and t_2 with $[0, 1]$.

5.3 Comparison with Δ-Based Approaches

The main drawback of our approach is that it does not terminate in the general case, although we exhibited cases for which termination is guaranteed (see



(a) Representation of K_r



(b) Representation of the Δ-cube

Fig. 4. Graphical comparison for the example in Fig. 2a

Section 4.2). In contrast, related work show that deciding only whether a system is robust is decidable in most cases. However, beside the fact that we give a quantitative measure of the robustness in the form of a constraint in $2*n$ dimensions (with n the number of transitions), our approach is particularly interesting in the case of a non-robust system. First, we exhibit which timing bounds are responsible for the non-robustness. Second, we give a condition to render the system robust without changing its discrete behaviour.

Furthermore, our approach may output a significantly larger constraint than the Δ-cube output by Δ-based approaches. Actually, when the result of *IMPN* is complete, the resulting polyhedron is necessarily at least as large as the Δ-cube. Consider again the example in Fig. 2a. In order to enable a graphical comparison in 2 dimensions, we assign all parameters but λ_1^- and λ_3^+ to their value as in π. Hence the constraint becomes $\lambda_1^- \leq 6 \wedge 1 \leq \lambda_3^+ \wedge \lambda_3^+ < \lambda_1^-$. This constraint is depicted in Fig. 4a. As of Δ-based approaches, they cannot compute a value for Δ greater than 1.5 in this situation. Indeed, with $\Delta = 1.5$, λ_3^+ becomes $\lambda_3^+ + \Delta = 3.5$, λ_1^- becomes $\lambda_1^- - \Delta = 3.5$, in which case the discrete behaviour becomes different (t_1 can fire before t_3). This Δ is given in Fig. 4b.

The interpretation of the much larger parametric domain covered by K_r compared to the Δ-cube can be explained as follows: (1) The parametric domain below $\lambda_3^+ = 2$ and above $\lambda_1^- = 5$ is not covered by the Δ-cube, because Δ-based approaches consider a positive parameter $\Delta \geq 0$. Hence, it is not possible to study, e.g. by how much an upper bound can be decreased. (2) The constraint K_r allows to relate parameters. Whereas the value of Δ prevents λ_1^- and λ_3^+ to vary by more than 1.5, the inequality $\lambda_3^+ < \lambda_1^-$ states that λ_1^- may vary by more than 1.5, as long as λ_3^+ varies less (i.e. $\lambda_3^+ < \lambda_1^-$). This is of particular interest in systems where some bounds are more likely to vary than others. (3) This small example is a “good” example for Δ-based approaches. In the case where at least one parameter cannot vary, Δ would be inevitably equal to 0, whereas K_r would still give an output for other dimensions. This is the case of the ITPNs in Fig. 1.

6 Final Remarks

In this paper, we extended the inverse method to PITPNs and showed how to exploit its output to obtain a precise quantitative measure of the system robustness for linear-time properties. This paper considers the quantification of the system robustness with respect to linear-time (hence time-abstract) properties only. Nevertheless, timed properties can also be considered, by adding an observer net. This observer synchronises with the system ITPN, and can reduce timed properties to time-abstract properties.

Our algorithms should be implemented and compared with similar tools, such as Shrinktech [11]. Finally, we only addressed here the variability of the timing delays ($\varDelta$), but not the admissible variations of the clock speed (usually called "ϵ"). Our approach could be extended to this setting using extensions of the inverse method for parameterised hybrid systems [7], by adding for each clock two additional parameters ϵ_i^- and ϵ_i^+ measuring the admissible decrease and increase speed rate.

Acknowledgment. We are grateful to an anonymous reviewer for his/her very detailed comments.

References

1. Akshay, S., Hélouët, L., Jard, C., Lime, D., Roux, O.H.: Robustness of time petri nets under architectural constraints. In: Jurdziński, M., Ničković, D. (eds.) FORMATS 2012. LNCS, vol. 7595, pp. 11–26. Springer, Heidelberg (2012)
2. Akshay, S., Hélouët, L., Jard, C., Reynier, P.-A.: Robustness of time Petri nets under guard enlargement. In: Finkel, A., Leroux, J., Potapov, I. (eds.) RP 2012. LNCS, vol. 7550, pp. 92–106. Springer, Heidelberg (2012)
3. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science 126(2), 183–235 (1994)
4. André, É., Soulat, R.: The Inverse Method. FOCUS Series in Computer Engineering and Information Technology. ISTE Ltd and John Wiley & Sons Inc. (2013)
5. Bouyer, P., Larsen, K.G., Markey, N., Sankur, O., Thrane, C.: Timed automata can always be made implementable. In: Katoen, J.-P., König, B. (eds.) CONCUR 2011. LNCS, vol. 6901, pp. 76–91. Springer, Heidelberg (2011)
6. Bouyer, P., Markey, N., Sankur, O.: Robust reachability in timed automata: A game-based approach. In: Czumaj, A., Mehlhorn, K., Pitts, A., Wattenhofer, R. (eds.) ICALP 2012, Part II. LNCS, vol. 7392, pp. 128–140. Springer, Heidelberg (2012)
7. Fribourg, L., Kühne, U.: Parametric verification and test coverage for hybrid automata using the inverse method. IJFCS 24(2), 233–249 (2013)
8. Jaubert, R., Reynier, P.-A.: Quantitative robustness analysis of flat timed automata. In: Hofmann, M. (ed.) FOSSACS 2011. LNCS, vol. 6604, pp. 229–244. Springer, Heidelberg (2011)
9. Markey, N.: Robustness in real-time systems. In: SIES, pp. 28–34. IEEE Computer Society Press (2011)
10. Merlin, P.M.: A study of the recoverability of computing systems. PhD thesis, University of California, Irvine, CA, USA (1974)
11. Sankur, O.: Shrinktech: A tool for the robustness analysis of timed automata. In: CAV. LNCS. Springer (to appear, 2013)
12. Traonouez, L.-M.: A parametric counterexample refinement approach for robust timed specifications. In: FIT. EPTCS, vol. 87, pp. 17–33 (2012)
13. Traonouez, L.-M., Lime, D., Roux, O.H.: Parametric model-checking of stopwatch Petri nets. Journal of Universal Computer Science 15(17), 3273–3304 (2009)

Spectral Gap in Timed Automata[⋆]

Eugene Asarin[1], Nicolas Basset[1,2], and Aldric Degorre[1]

[1] LIAFA, University Paris Diderot and CNRS, France
[2] LIGM, University Paris-Est Marne-la-Vallée and CNRS, France

Abstract. Various problems about probabilistic and non-probabilistic timed automata (computing probability density, language volume or entropy) can be naturally phrased as iteration of linear operators in Banach spaces. Convergence of such iterations is guaranteed whenever the operator's spectrum has a gap. In this article, for operators used in entropy computation, we use the theory of positive operators to establish the existence of such a gap. This allows to devise simple numeric algorithms for computing the entropy and prove their exponential convergence.

1 Introduction

This work emerged from our study of entropy of timed regular languages. Entropy can be seen as a measure of the size of a language or of the information content of its words. We have related it to Kolmogorov complexity [5] and to capacity of timed communication channels [2].

In our previous works [4,5], we devised several techniques for computing that entropy but they required some technical hypotheses and numerical convergence was not ensured. In this paper, we present new techniques of analysis of timed automata based on theory of positive linear operators. These techniques allow, in particular, simple and converging algorithms for computing entropy under a very general setting.

Positive linear operators (represented by matrices with non-negative elements) are extensively used in the analysis of finite-state systems (automata and Markov chains). The main technical tool is Perron-Frobenius theorem, which guarantees, under certain hypotheses, that positive operators have a particular structure of the spectrum: they have one simple real eigenvalue ρ with maximal modulus and eigenvector $\mathbf{v}$, and all other (complex) eigenvalues have smaller moduli, thus they are separated from the maximal one by a gap. When such an operator is applied many times to any non-negative vector $\mathbf{x}$, the result $A^n\mathbf{x}$ converges in direction to $\mathbf{v}$, and its length behaves roughly as ρ^n. The most-known consequences of this result are existence of (and convergence to) steady state probability in Markov chains and a characterization of the entropy of a regular language.

In [4], following the same approach and generalizing the notion of adjacency matrix, we have introduced an operator Ψ associated to a timed automaton and stated that the logarithm of its spectral radius equals the entropy of the language.

[⋆] The support of Agence Nationale de la Recherche under the project EQINOCS (ANR-11-BS02-004) is gratefully acknowledged.

This operator was used in [3]: the generating function of a timed language can be obtained using the resolvent of Ψ. This operator acts in a Banach space of infinite dimension, and thus its study is much more complicated than for finite automata. In particular, analogs of Perron-Frobenius theorem are more involved and in previous works we had no spectral gap properties and no algorithms with guaranteed convergence for computing entropy.

In this paper, we show that any timed deterministic automaton with finite entropy (after some preprocessing and decomposition) has an operator Ψ with a spectral gap. As a consequence, simple iterative numeric procedures can be used to compute entropy and they converge with exponentially small error bound. The proof is rather technical, and is only sketched.

Related Works. As mentioned, this work generalizes classical studies on entropy of regular languages [10] and on finite Markov chains, all based on Perron-Frobenius theorem (see [17,14] and reference therein). We strongly use a chapter of functional analysis – theory of positive linear operators, generalizing Perron-Frobenius theory, presented in detail in the monograph [13].

As far as we know, linear operators have not been explicitly considered by other authors in the context of timed automata, however [9,16,15] are very close in spirit and in fact proceed by iteration of some integral operators.

Paper Structure. In Sect. 2 we recall classical recipes for finite automata and finite-dimensional operators. In Sect. 3 we recall the notion of entropy of timed languages and proceed with some preprocessing of timed automata. In Sect. 4 we recall how to associate an operator to a timed automaton and characterize the entropy. In Sect. 5 we sketch the proof of the main result: the operator has a spectral gap. In Sect. 6 we apply the main result to compute the entropy of timed languages with a guaranteed convergence. In Sect. 7 we conclude with some perspectives.

2 Linear Operators and Finite Automata

In this first preliminary section, we recall a typical application of linear operators to finite automata and regular languages (in the rest of the paper these results are extended to timed regular languages). More details can be found in [14,17].

Consider a deterministic finite automaton $\mathcal{A} = (Q, \Sigma, \delta, q_1, F)$, with states $q_1, q_2, \dots, q_s$ such that every state is reachable from q_1 and F is reachable from any state. Let L be the language accepted, and L_n its sublanguage containing all its words of length n. In most cases, its cardinality $\#L_n$ depends on n exponentially, and the *entropy* of L (or of $\mathcal{A}$) is defined as the growth rate of this cardinality: $\mathcal{H}(L) = \limsup_{n\to\infty} \frac{\log \#L_n}{n}$. We recall how this entropy (which is an important size and information measure) is related to linear operators.

Let $L_{i,n}$ be the set of all n-letter words accepted by $\mathcal{A}$ when starting at the state q_i, and $x_{i,n}$ its cardinality. From usual language equations

$$L_{i,0} = \begin{cases} \{\varepsilon\}, & \text{if } q_i \in F \\ \emptyset, & \text{otherwise;} \end{cases} \qquad L_{i,n+1} = \bigcup_{(q_i,a,q_j)\in\delta} aL_{j,n}$$

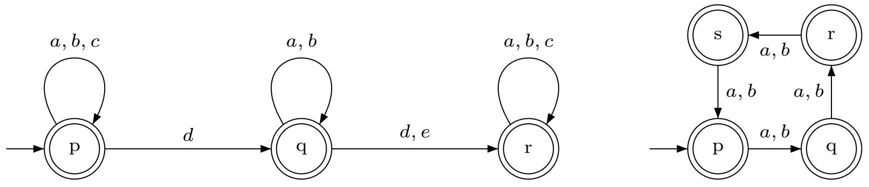


Fig. 1. Left: non-strongly-connected automaton. Right: periodic automaton.

one passes to equations on cardinalities

$$x_{i,0} = \begin{cases} 1, & \text{if } q_i \in F \\ 0, & \text{otherwise;} \end{cases} \qquad x_{i,n+1} = \sum_{(q_i,a,q_j)\in\delta} x_{j,n},$$

or, in vector notation, $\mathbf{x}_0 = \mathbf{x}_F$ and $\mathbf{x}_{n+1} = A\mathbf{x}_n$, with the vector $\mathbf{x}_n = (x_{1,n}, \dots, x_{s,n}) \in \mathbb{R}^s$ and the $s \times s$ matrix $A = (a_{i,j})$ such that $a_{i,j}$ is the number of transitions in $\mathcal{A}$ leading from q_i to q_j. In other words, A is the adjacency matrix of the automaton $\mathcal{A}$. We conclude with the explicit formula for cardinalities: $\mathbf{x}_n = A^n\mathbf{x}_F$. Thus size analysis of the automaton $\mathcal{A}$ is phrased as iteration of the linear operator A on $\mathbb{R}^s$. In particular, the entropy is the growth rate of the first coordinate $x_{1,n}$.

Exploration of the matrix A is simplified by non-negativity of its elements. Using Perron-Frobenius theory, the entropy can be characterized as follows:

Proposition 1. *It holds that $\mathcal{H}(\mathcal{A}) = \log\rho(A)$, where $\rho(A)$ stands for the spectral radius, i.e. the maximal modulus of eigenvalues of A.*

2.1 Two Decompositions

To rule out pathological behaviors where the iterations of A would not converge, the automaton must be decomposed, first, in strongly-connected components, then in aperiodic components.

Examples: why decompose. Consider first an example of a non-strongly-connected automaton, in Fig. 1, left. It has three strongly connected components. The matrix is $A = \begin{pmatrix} 3 & 1 & 0 \\ 0 & 2 & 2 \\ 0 & 0 & 3 \end{pmatrix}$ with two positive eigenvalues (3, which is double, and 2), and three positive eigenvectors (those of the standard basis). When we iterate the operator (i.e. compute $A^n\mathbf{x}$ for some initial non-negative vector $\mathbf{x}$), the growth rate can be 3^n (e.g. if we start with $\mathbf{x} = (1,1,1)$) which corresponds to the spectral radius, but it can also be 2^n (if we start with $\mathbf{x} = (0,1,0)$).

The second automaton, on the right of Fig. 1, is strongly connected but periodic. It has four eigenvalues with maximal modulus: $2; 2i; -2; -2i$. Iterating the operator leads to a fast rotating sequence of vectors: $(1,0,0,0)^\top$, $(0,2,0,0)^\top$, $(0,0,4,0)^\top$, $(0,0,0,8)^\top$, …

SCC decomposition. The automaton $\mathcal{A}$, considered as a graph, can be decomposed into strongly connected components. We will distinguish non-trivial components $\mathcal{A}_c$ (containing a cycle) from transient states (i.e single-state components without self-loops). For every $\mathcal{A}_c$ we consider the corresponding matrix A_c (which is a submatrix of A, i.e the matrix is obtained by selecting rows, then columns corresponding to states in $\mathcal{A}_c$). Computation of the entropy of $\mathcal{A}$ reduces to those of $\mathcal{A}_c$ thanks to the following result.

Proposition 2. $\rho(A) = \max_c \rho(A_c)$ *and thus* $\mathcal{H}(\mathcal{A}) = \max_c \mathcal{H}(\mathcal{A}_c)$.

Thus, we can restrict ourselves to the study of operators of strongly connected automata, which constitutes our first decomposition.

Periodic decomposition. Given a strongly-connected automaton $\mathcal{A}$, we define its period $\mathfrak{p}$ as the greatest common divisor of the lengths of its cycles. Then the following decomposition is possible (see e.g. [11]).

Proposition 3. *The set Q can be split into $\mathfrak{p}$ periodic components $Q_0, \dots, Q_{\mathfrak{p}-1}$ satisfying the following properties:*

- *any path visits cyclically in turn all the components $Q_0, \dots, Q_{\mathfrak{p}-1}$;*
- *hence, any path of length $\mathfrak{p}$ starts and ends in the same component;*
- *there exists a natural b such that any two states within the same Q_i are connected by some path of length $b\mathfrak{p}$.*

The space $\mathbb{R}^s$ is naturally split into a direct sum of subspaces E_i for $i \in 0..\mathfrak{p}-1$ corresponding to periodic components. Each E_i consists of vectors in $\mathbb{R}^s$ with coordinates vanishing outside of Q_i.

Operator A maps each E_i to $E_{i-1 \bmod \mathfrak{p}}$; hence each E_i is invariant under $A^{\mathfrak{p}}$. We denote the restriction of $A^{\mathfrak{p}}$ to E_i by $A_i^{\mathfrak{p}}$ (which is a submatrix of $A^{\mathfrak{p}}$).

Proposition 4. *For all $i \in 0..\mathfrak{p}-1$, $\rho(A_i^{\mathfrak{p}}) = \rho(A^{\mathfrak{p}})$.*

We conclude that for a strongly connected automaton $\rho(A) = \rho(A^{\mathfrak{p}})^{1/\mathfrak{p}} = \rho(A_i^{\mathfrak{p}})^{1/\mathfrak{p}}$ for any $i \in 0..\mathfrak{p}-1$. Thus we can concentrate our effort on the operator restricted to one periodic component: $A_i^{\mathfrak{p}}$.

2.2 Spectral Gap and Its Consequences

Consider now the operator for one periodic component $B = A_i^{\mathfrak{p}}$. It has particular properties. In terms of Perron-Frobenius theory it is irreducible. All its powers B^n with $n \geq b$ (with b as in Prop. 3) are matrices with all positive elements. It follows from Perron-Frobenius theory that the operator B has a *spectral gap* $\beta \in (0,1)$, in the following sense:

1. $\rho(B)$ is a positive simple[1] eigenvalue of B;
2. the rest of the spectrum of B belongs to the disk $\{z \mid |z| \leq (1-\beta)\rho(B)\}$;

[1] An eigenvalue λ is simple if its generalized eigenspace has dimension 1.

Fig. 2, left, illustrates the spectrum of such an operator.

Due to this gap, iterations of B on any positive vector behave in a very regular way, and numerical computation of $\rho(B)$ and of the eigenvector $\mathbf{v}$ become particularly easy.

Proposition 5. *For any positive vector* $\mathbf{x}$*:*

– *the vector* $B^n\mathbf{x}$ *converges in direction to* $\mathbf{v}$*;*
– *the ratio* $|B^{n+1}\mathbf{x}|/|B^n\mathbf{x}|$ *converges to* $\rho(B)$*;*
– *the error in both cases converges in* $O((1-\beta)^n)$*.*

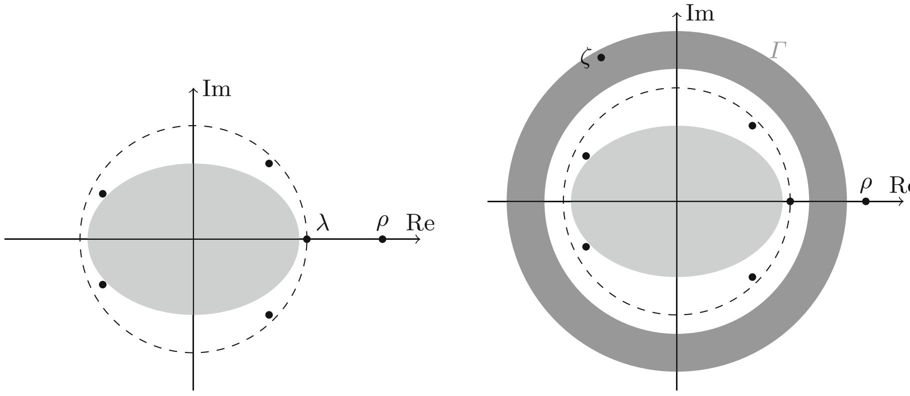


Fig. 2. Left: spectrum of an operator with β-gap. Right: finding δ for Lem. 6; the spectrum of the perturbed operator cannot cross the ring Γ.

3 Timed Automata, Volumes and Entropy

In this second preliminary section, we recall the notions of volume and entropy of timed languages from [5,4], describe a form of region graph from [4] and a characterization of languages with non-vanishing entropy from [8].

3.1 Geometry, Volume and Entropy of Timed Languages

A *timed word* of length n over an alphabet Σ is a sequence $t_1a_1 \dots t_na_n$, where $a_i \in \Sigma, t_i \in \mathbb{R}_{\geq 0}$. Here t_i represents the delay between the events a_{i-1} and a_i. For every timed language $L \subseteq \Sigma^*$ and word $w = w_1 \dots w_n \in \Sigma^n$ we define $P_w^L = \{(t_1, \dots, t_n) \mid t_1w_1 \dots t_nw_n \in L\}$. For a fixed n, we define the *n-volume* of L as follows: $V_n(L) = \sum_{w\in\Sigma^n} \operatorname{Vol} P_w^L$, where Vol stands for the standard Euclidean volume in $\mathbb{R}^n$. In case of regular timed languages, the sets P_w^L are union of polytopes, and hence their volumes (finite or infinite) are well-defined. For a timed language $L(\mathcal{A})$ recognized by a timed automaton $\mathcal{A}$ we will just write $V_n(\mathcal{A})$ (or even V_n) instead of $V_n(L(\mathcal{A}))$.

Similarily to the discrete case, the *(volumetric) entropy* of a timed language L is defined as $\mathcal{H}(L) = \limsup_{n\to\infty} \frac{\log V_n}{n}$.

3.2 Bounded Deterministic Timed Automata

We briefly fix notations and recall definitions about timed automata ([1]).

We fix a natural constant M which upper bounds all the constants in the automaton. A *clock* is a variable ranging over $\mathbb{R}_{\geq 0}$. A *clock constraint* over a set of clocks C is a finite conjunction of inequalities of the form $x \sim c$ or $x \sim y$, where x and y are clocks, $\sim \in \{<, \leq, =, \geq, >\}$ and $c \in 0..M$. A clock reset $\mathfrak{r}$ is a function $\mathbb{R}^C \to \mathbb{R}^C$ which sets to 0 the clocks in some fixed subset of C and does not modify the values of the others.

A timed automaton (TA) is a tuple $\mathcal{A} = (Q, \Sigma, C, \Delta, q_0, F)$. Its elements are respectively the finite set of locations, the finite alphabet, the finite set of clocks (let its cardinality be d), the transition relation, the initial location, and the final condition. A state of $\mathcal{A}$ is a pair $(q, \mathbf{x})$ of a control location $q \in Q$ and a vector of clock values $\mathbf{x} \in \mathbb{R}^d$. An element of Δ is written as $\delta = (q, a, \mathfrak{g}, \mathfrak{r}, q')$ meaning a transition from q to q' with label a, guard $\mathfrak{g}$ (which is a clock constraint) and reset $\mathfrak{r}$. An element of F has a form $(q, \mathfrak{g})$ meaning that an accepting run can terminate by a transition to q with clocks respecting the clock constraint $\mathfrak{g}$.

A *run* of $\mathcal{A}$ along a *path* $\pi = \delta_1 \ldots \delta_n \in \Delta^n$ has the form $(q_{i_0}, \mathbf{x}_0) \xrightarrow{t_1 a_1} (q_{i_1}, \mathbf{x}_{i_1}) \xrightarrow{t_2 a_2} \cdots \xrightarrow{t_n a_n} (q_{i_n}, \mathbf{x}_n)$ where for all $j \in 1..n$, $\delta_j = (q_{i_{j-1}}, a_j, \mathfrak{g}, \mathfrak{r}, q_{i_j}) \in \Delta$, $\mathbf{x}_{j-1} + t_j \mathbf{1} \models \mathfrak{g}$ and $\mathbf{x}_j = \mathfrak{r}(\mathbf{x}_{j-1} + t_j \mathbf{1})$. In this case we use the notation $\mathbf{x}_1 \xrightarrow{t_1 \ldots t_n, \pi} \mathbf{x}_n$ to say that such a run exists.

When $q_{i_0} = q_0$ is the initial state, $\mathbf{x}_0 = 0$ and F contains a couple $(q, \mathfrak{g})$ with $q_{i_n} = q$ and $\mathbf{x}_n$ satisfying $\mathfrak{g}$, then the timed word $t_1 a_1 \ldots t_n a_n$ is said to be *accepted* by $\mathcal{A}$. The set of all such words is the language $L(\mathcal{A})$ accepted by $\mathcal{A}$.

Several objects are naturally associated with a path. Given a path and two clock vectors, a polytope of all the timings of the path can be defined: $P(\pi, \mathbf{x}, \mathbf{x}') = \{\boldsymbol{t} \mid \mathbf{x} \xrightarrow{\boldsymbol{t}, \pi} \mathbf{x}'\}$. We also define the reachability relation: $\mathtt{Reach}(\pi) = \{(\mathbf{x}, \mathbf{x}') \mid \exists \boldsymbol{t}, \mathbf{x} \xrightarrow{\boldsymbol{t}, \pi} \mathbf{x}'\}$.

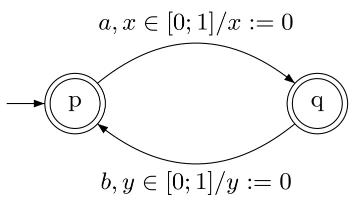
A TA is *deterministic* if for any two transitions with the same source and the same label the guards are disjoint. It is *bounded* whenever every guard upper bounds at least one clock.

In the rest of the paper, we compute volumes and entropy for regular timed languages recognized by deterministic timed automata (DTA). Moreover, if some guards in the automaton were unbounded, the volume would be infinite, which is beyond the reach of our approach. Thus we concentrate on Bounded Deterministic Timed Automata (BDTA).

Remark 1. Most of known techniques to compute entropy of untimed regular languages work on deterministic automata. In fact, these techniques count paths in the automaton, and only in the deterministic case their number coincides with the number of accepted words. The same is true for volumes in timed automata.

3.3 A Running Example

To illustrate the notions of volume and entropy, we consider the languages recognized by the BDTA $\mathcal{E}$ on the left of Fig. 3.



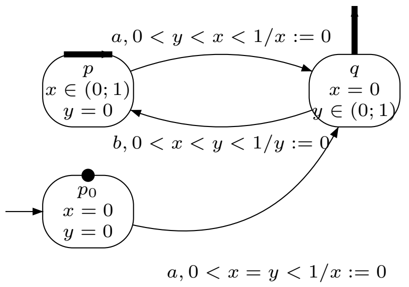

$a, 0 < y < x < 1/x := 0$
p
$x \in (0;1)$
$y = 0$
q
$x = 0$
$y \in (0;1)$
$b, 0 < x < y < 1/y := 0$
p_0
$x = 0$
$y = 0$
$a, 0 < x = y < 1/x := 0$

Fig. 3. A simple timed automaton $\mathcal{E}$ (left) and its fleshy region-split form (right)

The language recognized by this automaton contains the words of the form $t_1at_2bt_3at_4b\ldots$ with $t_i + t_{i+1} \in [0;1]$. Notice that the automaton has two clocks that are never reset together. The geometric form of possible delay vectors $(t_1 \ldots t_n)$ in $\mathbb{R}^n$ is defined by overlapping constraints $t_i + t_{i+1} \in [0;1]$. A systematic method to compute the volume of this polytope is described below in Sect. 4.1. It gives the sequence of volumes: $1; 1/2; 1/3; 5/24; 2/15; 61/720; 17/315; 277/8064; \ldots$ As shown in [4], the entropy of this language is $\log(2/\pi)$.

3.4 Preprocessing Timed Automata

In order to compute volumes V_n and entropy $\mathcal{H}$ of the language of a BDTA, we first transform this automaton into a normal form, a (timed) variant of the region graph defined in [1]. We recall that a subset of $\mathbb{R}^d$ defined by a clock constraint is called a *zone*. Smallest (by inclusion) zones are called *regions*. We say that a BDTA $\mathcal{A} = (Q, \Sigma, C, \delta, q_0, F)$ is in a *region-split form* if properties B1-3, below, hold. Furthermore, with additional property B4, such an automaton is called *fleshy*.

B1. Each location and each transition of $\mathcal{A}$ is visited by some accepting run.
B2. For every location $q \in Q$ a unique clock region $\mathbf{r}_q$ (called its *entry region*) exists, such that the set of clock values with which q is entered is exactly $\mathbf{r}_q$. For the initial location q_0, its entry region is the singleton $\{0\}$.
B3. The guard $\mathfrak{g}$ of every transition $\delta = (q, a, \mathfrak{g}, \mathfrak{r}, q') \in \Delta$ is just one region. All the clock values satisfying $\mathfrak{g}$ are time-reachable from $\mathbf{r}_q$.
B4. For every transition δ its guard $\mathfrak{g}$ has no constraints of the form $x = c$.

By the fundamental property of region abstraction, any path of the underlying graph of a region-split TA is realizable as a run that follows the same edges.

Proposition 6 ([4]). *Given a BDTA accepting a language L, a fleshy region-split TA accepting a language $L' \subset L$ with $V_n(L') = V_n(L)$ and $\mathcal{H}(L') = \mathcal{H}(L)$ can be constructed.*

From now on, we suppose w.l.o.g. that the automaton $\mathcal{A}$ is in a fleshy region-split form (see Fig. 3, right).

3.5 Thick Languages and Forgetful Automata

A timed language is called *thin* if its entropy is $-\infty$ and *thick* otherwise ($\mathcal{H} > -\infty$). In [8] we have characterized timed automata recognizing thick languages in the following way. A path π is called *forgetful* if $\mathtt{Reach}(\pi) = \mathbf{r} \times \mathbf{r}'$. Informally, after reading $(\boldsymbol{t}, \pi)$ from a state s_0, the reached state s does not depend on s_0 (π forgets its starting state). Forgetful paths satisfy the *progress* condition: along these paths each clock is reset at least once. We will use the key characterization of thickness in terms of forgetfulness:

Theorem 1 ([8]). *For a BDTA in region split form, $\mathcal{H} > -\infty$ if and only if there exists a forgetful cycle.*

The automaton of the running example is thick; one of its forgetful cycles is ab. The proof of Thm. 1 is based on a kind of pumping lemma (also used below).

Theorem 2 ([8]). *For every BDTA $\mathcal{A}$ and $\eta > 0$, there exists N_η such that any path π longer than N_η with $\mathtt{Vol}(L_\pi) \geq \eta^{|\pi|}$ contains a forgetful cycle.*

This means any long path with non-vanishing volume contains a forgetful cycle.

4 Timed Automata: Operators

We adapt several definitions and results of [4] to our more general setting.

4.1 Recurrent Equations on Volume Functions

Given a BDTA $\mathcal{A}$, we want to compute its entropy based on its n-volumes V_n. In order to obtain recurrent equations on these volumes, we need to take into account all possible initial locations and clock configurations. For every state $(q, \mathbf{x})$, let $L(q, \mathbf{x})$ be the set of all the timed words corresponding to the runs of the automaton starting at this state, let $L_n(q, \mathbf{x})$ be its sublanguage consisting of its words of length n, and $v_n(q, \mathbf{x})$ the volume of this sublanguage. Similarly we define for a path π its volume function, $v_\pi(\mathbf{x}) = \mathtt{Vol}(L_\pi(\mathbf{x}))$.

By definition of runs of a timed automaton, we obtain the following language equations for $q \in Q$ and $\mathbf{x} \in \mathbf{r}_q$:

$$L_0(q, \mathbf{x}) = \{\varepsilon\} \text{ if } q \text{ is final;} \quad L_0(q, \mathbf{x}) = \emptyset \text{ otherwise;}$$

$$L_{k+1}(q, \mathbf{x}) = \bigcup_{(q,a,\mathfrak{g},\mathfrak{r},q') \in \Delta} \; \bigcup_{\tau : \mathbf{x}+\tau \in \mathfrak{g}} \tau a L_k(q', \mathfrak{r}(\mathbf{x} + \tau)).$$

Since the automaton is deterministic, the union over transitions (the first $\bigcup$ in the formula) is disjoint. Hence, it is easy to pass to volumes:

$$v_0(q, \mathbf{x}) = 1_F(q, \mathbf{x});$$

$$v_{k+1}(q, \mathbf{x}) = \sum_{(q,a,\mathfrak{g},\mathfrak{r},q') \in \Delta} \int_{\tau : \mathbf{x}+\tau \in \mathfrak{g}} v_k(q', \mathfrak{r}(\mathbf{x} + \tau)) \, d\tau, \tag{1}$$

where 1_F is the indicator function of the final states F.

4.2 Operator Ψ and Its Link to Volumes and Entropy

The recurrent formula (1) has the form $v_{k+1} = \Psi v_k$, (and hence $v_n = \Psi^n 1_F$), where Ψ is the operator defined by the equation:

$$\Psi f(q, \mathbf{x}) = \sum_{(q,a,\mathfrak{g},\mathfrak{r},q')\in\Delta} \int_{\mathbf{x}+\tau\in\mathfrak{g}} f(q', \mathfrak{r}(\mathbf{x}+\tau))\, d\tau. \tag{2}$$

Now we must define the functional space where the function v_n lies and where the operator acts. We slightly modify the functional space of [4]. We define S as the disjoint union of all the (closures of) entry regions of all the states of $\mathcal{A}$. Formally, $S = \{(q, \mathbf{x}) \mid \mathbf{x} \in \bar{\mathbf{r}}_q\}$. The elements of the space $\mathcal{F}$ are continuous functions from S to $\mathbb{R}$. The uniform norm $\|u\| = \sup_{\xi\in S} |u(\xi)|$ can be defined on $\mathcal{F}$, yielding a Banach space structure. We can compare two functions in $\mathcal{F}$ pointwise, thus we write $u \le v$ if $\forall\xi \in S : u(\xi) \le v(\xi)$. For a function $f \in \mathcal{F}$ we sometimes denote $f(p, x)$ by $f_p(x)$. Thus, any function $f \in \mathcal{F}$ can be seen as a finite collection of functions f_p defined on entry regions $\bar{\mathbf{r}}_p$ of locations of $\mathcal{A}$. When restricted to one location q the volume functions $\mathbf{x} \mapsto v_n(q, \mathbf{x})$ are polynomial on $\mathbf{r}_q$ [4] and can thus be prolongated to its closure $\bar{\mathbf{r}}_q$. We conclude that $v_n \in \mathcal{F}$ for all $n \in \mathbb{N}$.

Proposition 7. *The operator Ψ is a linear bounded positive operator on the Banach space $\mathcal{F}$.*

The problem of computing volumes and entropy is now phrased as studying iterations of operator Ψ on a functional space $\mathcal{F}$. The theory of positive operators guarantees, that under some hypotheses, v_n is close in direction to a positive eigenvector v^* of Ψ, corresponding to its leading eigenvalue ρ. Moreover, values of v_n will grow/decay exponentially like ρ^n. The eigenvalue ρ and the corresponding eigenvector can be computed using natural iterative procedures. In the sequel we apply this general scheme to the operator Ψ, referring to the book [13] when a result concerning positive operators is needed.

The following theorem was the main result of [4], it still holds for our more general class of timed automata.

Theorem 3 (adapted from [4]). *For any BDTA $\mathcal{H} = \log \rho(\Psi)$.*

The concluding result of the present paper is a procedure to compute the spectral radius of Ψ and thus the entropy $\mathcal{H}$ based on the spectral gap. To prove existence of this spectral gap we will use the kernel form of path operators and properties of thick automata.

4.3 Path Operators and Their Kernel Form

Equation (2) can be rewritten as:

$$(\Psi f)_q(\mathbf{x}) = \sum_{\delta=(q,\ldots,q')\in\Delta} (\psi_\delta f_{q'})(\mathbf{x}). \tag{3}$$

where for $\delta = (q, a, \mathfrak{g}, \mathfrak{r}, q')$ the operator ψ_δ acts from the space $C(\bar{\mathbf{r}}_{q'})$ (of continuous functions on the target region) to the space $C(\bar{\mathbf{r}}_q)$. It is defined by the integral: $\psi_\delta f(\mathbf{x}) = \int_{\mathbf{x}+\tau\in\mathfrak{g}} f(\mathfrak{r}(\mathbf{x}+\tau))\,d\tau$. Iterating (3), we obtain a formula for powers of operator Ψ

$$(\Psi^k f)_p(\mathbf{x}) = \sum_{\delta_1\ldots\delta_k \text{ from } p \text{ to } p'} (\psi_{\delta_1}\ldots\psi_{\delta_k} f_{p'})(\mathbf{x}). \quad (4)$$

For a path $\pi = \delta_1\ldots\delta_k \in \Delta^k$ starting in a state p and leading to a state q, we define $\psi_\pi = \psi_{\delta_1}\ldots\psi_{\delta_k}$, this operator acts from $C(\bar{\mathbf{r}}_q)$ to $C(\bar{\mathbf{r}}_p)$. Let D be the dimension of $\mathbf{r}_q$. When the path π satisfies the progress condition, for $(\mathbf{x},\mathbf{x}') \in \mathbf{r}_p\times\mathbf{r}_q$ the polytope $P(\pi,\mathbf{x},\mathbf{x}')$, is either empty or of dimension $(n-D)$, and we denote by $v_\pi(\mathbf{x},\mathbf{x}')$ its $(n-D)$-dimensional volume. We have the following representation of ψ_π.

Theorem 4 (kernel form,[7]). *When π is a progress path, the function v_π is a kernel for ψ_π:*

$$\psi_\pi(f)(\mathbf{x}) = \int_{\mathbf{r}_q} v_\pi(\mathbf{x},\mathbf{x}')f(\mathbf{x}')d\mu_q(\mathbf{x}'). \quad (5)$$

The kernel v_π is piecewise polynomial, strictly positive and continuous on $\mathtt{Reach}(\pi)$*; it is zero outside of* $\mathtt{Reach}(\pi)$*.*

The measure $\mu_q(\mathbf{x}')$ in the theorem is a D-dimensional Lebesgue measure on $\mathbf{r}_q$. For example, if $\mathbf{r}_q$ is defined by the clock constraint $0 = x_1 < x_2 - 1 = x_3 - 1 < x_4 - 2 < 1$ then $\int_{\mathbf{r}_q} f(\mathbf{x})d\mu_q(\mathbf{x}) = \int_1^2 \left(\int_{x_2+1}^3 f(0,x_2,x_2,x_4)dx_4\right)dx_2$. The theorem applied to forgetful paths ensures that $v_\pi(\mathbf{x},\mathbf{x}') > 0$ on $\mathbf{r}_p\times\mathbf{r}_q$.

Example 1. Let us apply the theorem to the forgetful cycle ab of our running example. We have $x \xrightarrow{t_1 a t_2 b} x'$ if and only if (x, t_1, t_2, x') satisfy the set of inequations $(I) = \{0 < x < 1, 0 < t_1, 0 < t_2, x + t_1 < 1, t_1 + t_2 < 1\}$ and $x' = t_1 + t_2$. We instantiate (I) with $t_1 = x' - t_2$, and obtain the set of inequations $(I') = \{t_2 < x', 0 < t_2, x + x' - 1 < t_2\}$. The kernel of ψ_{ab} is $v_{ab}(x,x') = \mathtt{Vol}\{t_2 \mid (x,t_2,x') \models (I')\} = \min(x', 1-x)$. Thus $\psi_{ab}(f)(x) = \int_0^1 \min(x',1-x)f(x')dx' = (1-x)\int_{1-x}^1 f(x')dx' + \int_0^{1-x} x'f(x')dx'$.

4.4 Two Decompositions Again

As in the untimed case (Sect. 2.1) we decompose the automaton and the operator. First we partition the location set Q (and thus the automaton $\mathcal{A}$) in strongly connected components. We only consider non-trivial (i.e. containing a cycle) components $\mathcal{A}_c$. The following result mimics Prop. 2.

Proposition 8 ([6,8]). *$\mathcal{H}(\mathcal{A}) = \max_c \mathcal{H}(\mathcal{A}_c)$.*

Thus, like in the discrete case, we can restrict ourselves to the study of operators of strongly connected automata. Since the entropy of thin SCC is $-\infty$, we will only consider thick components.

Given a strongly-connected (region-split fleshy) timed automaton $\mathcal{A}$, we define its period $\mathfrak{p}$ as the greatest common divisor of the lengths of its cycles. Then, as in Prop. 3, the location set Q can be split into $\mathfrak{p}$ periodic components Q_i, for $i \in 0..\mathfrak{p}-1$, and the Banach space $\mathcal{F}$ into a direct sum of the corresponding subspaces $\mathcal{F}_i$. Each $\mathcal{F}_i$ consists of functions in $\mathcal{F}$ vanishing outside of Q_i.

Operator Ψ maps each $\mathcal{F}_i$ to $\mathcal{F}_{i-1 \bmod \mathfrak{p}}$; hence each $\mathcal{F}_i$ is invariant under $\Psi^{\mathfrak{p}}$. We denote the restriction of $\Psi^{\mathfrak{p}}$ to $\mathcal{F}_i$ by $\Psi_i^{\mathfrak{p}}$. The following result mimics Prop. 4.

Proposition 9. *For all $i \in 0..\mathfrak{p}-1$, $\rho(\Psi_i^{\mathfrak{p}}) = \rho(\Psi^{\mathfrak{p}})$.*

Thus we can concentrate our effort on the operator restricted to one periodic component: $\Psi_i^{\mathfrak{p}}$.

5 Spectral Gap

It is well-known that computation of the spectral radius of an operator (as well as other convergence properties) is substantially simplified by the existence of spectral gap in the operator, as defined in Subsect. 2.2. Here we show that every periodic component of the operator $\Theta = \Psi_i^{\mathfrak{p}}$ with $\mathfrak{p}$ the period of the automaton has such a gap. This result will be used in the next section to ensure convergence of a numerical algorithm for entropy computation.

We are ready to formulate the main result of this article.

Theorem 5 (spectral gap). *For any region-split strongly connected thick timed automaton $\mathcal{A}$ of period $\mathfrak{p}$ the operator $\Theta = \Psi_i^{\mathfrak{p}}$ has a spectral gap.*

The proof of this result is quite technical, and we only present the logical structure of the proof and some of its ideas. The proof is based on Perron-Frobenius theory for acute operators as in [13].

The idea of acuteness can be explained as follows. Let v be a non-zero vector in the functional space $\mathcal{F}$, and let h be a non-zero covector (a functional) in the dual space $\mathcal{F}$. The angle α between them can be naturally defined as follows:

$$\cos\alpha = \frac{\langle h, v\rangle}{\|h\| \cdot \|v\|} \text{ with } 0 \leq \alpha \leq \pi$$

(for h and v two vectors in Euclidean $\mathbb{R}^n$ this is the usual angle). For non-negative h and v the angle is always between 0 and $\pi/2$.

A linear positive operator $A : \mathcal{F} \to \mathcal{F}$ is called *acute* if applying it to any non-negative non-zero h and v yields A^*h and Av forming an acute angle smaller than some fixed acute ϕ. Formally, we say that A is acute with a cosine $\cos\phi$ where $\phi \in (0, \pi/2)$ whenever

$$\forall \text{ non-zero } h, v \geq 0 : \ \cos\phi \leq \frac{\langle A^*h, Av\rangle}{\|A^*h\| \cdot \|Av\|}. \tag{6}$$

We are interested in acuity since it is a sufficient condition for existence of a spectral gap:

Lemma 1 ([13], Thm. 12.3). *A positive acute operator A with cosine $\cos\phi$ has a β-gap with $\beta = 1 - \tan\phi/2$ (i.e. $\beta = \Omega(\cos\phi)$ whenever $\cos\phi$ is small).*

In order to prove that the operator Ψ has a spectral gap we first concentrate on two adjacent forgetful paths π_1 and π_2, and prove that the angle between $\psi^*_{\pi_1}h$ and $\psi_{\pi_2}v$ is acute, and its cosine admits an exponential lower bound.

Lemma 2 (angle between two forgetful paths). *Let π_1 and π_2 be forgetful paths of length n from p to q and from q to r respectively. Let $h \in C^*(\mathbf{r}_p)$ and $v \in C(\mathbf{r}_r)$ be both non-negative and non-zero. Then, for some $\alpha > 0$ depending only on the automaton, the following inequality holds:*

$$\frac{\langle \psi^*_{\pi_1}h, \psi_{\pi_2}v\rangle}{\|\psi^*_{\pi_1}h\| \cdot \|\psi_{\pi_2}v\|} \geq \alpha^n. \tag{7}$$

We give a very rough idea of the proof using the kernel form given by Thm. 4. We sketch the proof of the following sufficient condition for (7):

$$\forall \mathbf{x} \in \mathbf{r}_p, \forall \mathbf{z} \in \mathbf{r}_r,\ \frac{\int_{\mathbf{r}_q} v_{\pi_1}(\mathbf{x},\mathbf{y})v_{\pi_2}(\mathbf{y},\mathbf{z})d\mathbf{y}}{\int_{\mathbf{r}_q} v_{\pi_1}(\mathbf{x},\mathbf{y})d\mathbf{y}\, \sup_{\mathbf{y}\in\mathbf{r}_q} v_{\pi_2}(\mathbf{y},\mathbf{z})} \geq \alpha^n. \tag{8}$$

For each timed run following the path $\pi_1\pi_2$ from $\mathbf{x} \in \mathbf{r}_p$ to $\mathbf{z} \in \mathbf{r}_q$, we consider separately its first part, over path π_1, and its second part, over π_2. We transform the first part so that it reaches a point inside some shrunk version $\mathbf{r}_q^-$ of the clock region at the end of π_1, closer to its barycenter (it is important that this transformation does not change too much the volumes). Then we change its second part, making it start from the point of the shrunk region that would minimize path volumes over π_2. After this transformation, the integral corresponding to the numerator splits into a product of two factors $\int_{\mathbf{r}_q^-} v_{\pi_1}(\mathbf{x},\mathbf{y})d\mathbf{y}$ and $\min_{\mathbf{y}\in\mathbf{r}_q^-} v_{\pi_2}(\mathbf{y},\mathbf{z})$ proportional to the two factors of the denominator. Thus the fraction simplifies and we get the required estimate.

Now the properties of the periodic decomposition enter into the play.

Lemma 3. *In any periodic component of a strongly connected and thick automaton, there exists a natural ℓ (multiple of the period), such that for every states p and q in this component there exists a forgetful path θ_{pq} of length exactly ℓ.*

We call a path π of length n *good* if its last $n-\ell$ transitions form a forgetful path (where ℓ comes from the previous lemma). Of course, a good path is forgetful.

As we know from Eq. (4), restricted to one periodic component, the operator Θ^n admits the following matrix representation:

$$(\Theta^n f)_p = ((\Psi_i^{\mathfrak{p}})^n f)_p = \sum_{p\xrightarrow{\pi}q, |\pi|=n\mathfrak{p}} \psi_\pi f_q,$$

where locations p and q belong to the periodic component Q_i and $f \in \mathcal{F}_i$. We will split it into two operators: $\Theta^n = \Phi_n + \Xi_n$ where Φ_n corresponds to good paths and Ξ_n to bad ones.

The following lemma states that the huge majority of paths are good.

Lemma 4 (size of good and bad paths). *Bad paths have a volume smaller than any exponent, while good are at least exponential:*

Bad are small: *For every $\iota > 0$ there exists N such that for all $n > N$ it holds that $\|\Xi_n\| < \iota^n$.*

Good are large: *There exists $\nu > 0$ and N such that for all $n > N$ it holds that $\rho(\Phi_n) > \nu^n$.*

The first item is an immediate corollary of Thm. 2.

We know from Lemma 2, that operators corresponding to two adjacent forgetful paths form an acute angle. It is possible to deduce that operator Φ_n (which is a sum of many operators of forgetful paths) is also acute:

Lemma 5 (good part is acute). *The operator Φ_n is acute, and its cosine admits an exponential lower bound:* $\cos\phi = \Omega(\gamma^n)$ *with some* $\gamma > 0$.

We have thus decomposed the operator Θ^n into an acute operator Φ_n and a small operator Ξ_n. By Lemma 1, Φ_n has a spectral gap. We need some results from perturbation theory to establish that the influence of Ξ_n on the spectrum is negligible and thus $\Psi_{\mathcal{A}}^n$ also has a gap.

For an operator A with β-gap and spectral radius ρ consider a ring on the complex plane: $\Gamma = \{\zeta | (1-3\beta/4)\rho \leq |\zeta| \leq (1-\beta/4)\rho\}$ (see Fig. 2, right). By definition of the gap, all ζ in this ring do not belong to the spectrum of A, thus the resolvent operator $(A-\zeta)^{-1}$ is well defined. Let δ be the maximal norm of this resolvent: $\delta = \sup_{\zeta\in\Gamma} \left\|(A-\zeta)^{-1}\right\|$.

Lemma 6 (small perturbation preserves spectral gap). *Let A be a linear operator with gap β. Let B satisfy $\|B\| < \delta^{-1}$. Then $A+B$ also has a gap $\beta/2$.*

This is a well-known fact of perturbation theory (see e.g. [12]).

It turns out that for an acute operator the parameter δ can be estimated. This allows combining Lemmata 1, 6.

Lemma 7 (resolvent norm for acute operators). *Let A be a linear positive acute operator with cosine* $\cos\phi$ *and spectral radius ρ. Then the parameter δ described above satisfies $\delta = O((\cos\phi)^{-6}\rho^{-1})$.*

Lemma 8. *For a thick strongly connected BDTA $\mathcal{A}$ there exists N such that for all $n \geq N$ the operator Θ^n has a spectral gap.*

We have proved the gap property for the automaton operator in high powers: Θ^n for $n \geq N$. Based on the following lemma, we can deduce the same property for Θ (this ends the proof of Thm. 5).

Lemma 9. *Let A be a positive operator. If both operators A^N and A^{N+1} have gaps then operator A also has a gap.*

Table 1. Iterative algorithm: approximating $\mathcal{H}$

```
1. Transform $\mathcal{A}$ into the fleshy region-split form.
2. Decompose it into strongly-connected components $A_c$.
3. For every thick $A_c$ find its operator $\Psi_c$ and period $\mathfrak{p}_c$
4. Compute the sequence of functions $g_{c,0} = 1;\quad g_{c,n+1} = \Psi_c^{\mathfrak{p}_c} g_{c,n}$.
5. Compute the approximations $\rho_{c,n} = \|g_{c,n+1}\| / \|g_{c,n}\|$.
6. Compute $\mathcal{H} = \max_c\{\log \rho_c/\mathfrak{p}_c\} \approx \max_c\{\log \rho_{c,n}/\mathfrak{p}_c\}$.
```

6 Computing the Entropy

In this section, we present a typical application of spectral gap results (Thm. 5): a very simple procedure for numeric approximation of the entropy of any BDTA, as characterized in Thm. 3.

Let A be a positive aperiodic linear operator with a spectral gap larger than β. Our aim is to compute its spectral radius. For this we iterate the operator: $g_0 = 1;\quad g_{n+1} = Ag_n$. An approximation of $\rho(A)$ can be computed as follows: $\rho_n = \|g_{n+1}\|/\|g_n\|$. As stated in [13, (15.16)], for some constant C, the following exponential error estimate holds: $|\rho_n - \rho(A)| < C(1-\beta)^n$.

Combining with the results of the Sect. 4 we obtain the algorithm to compute the entropy of a timed automaton presented in Table 1. We summarize with the following result:

Theorem 6. *The algorithm in Table 1 computes the entropy of a BDTA with an exponentially small error (wrt the number of iterations n).*

Example 2. Applying the method to the running example, we first restrict the study to the cycle ab which is the only non-trivial strongly connected component. Its period is 2 and thus we must compute $\Psi^{2n}(1)$ for $n = 0, 1, 2, \ldots$ restricted to one periodic component p (or q). Table 2 contains the four first iterations of Ψ^2. In this table we present $g_n(x) = \Psi^{2n}(1)(p, x) = \psi_{(ab)^n}(1)(x) = v_{(ab)^n}(x)$, its norm and $\rho_{n-1} = \|g_n\|/\|g_{n-1}\|$ (which is an approximation of $\rho(\Psi^2) = \rho(\Psi)^2$. This yields the following approximation of the entropy $\mathcal{H} \approx (\log \rho_3)/2 \approx -0.6512$ which is close to the true value (see [4]) $\mathcal{H} = \log(2/\pi) \approx -0.6515$.

7 Perspectives

We believe that the techniques based on linear operators and their spectral gaps can be applied to other problems on timed automata, such as convergence of probabilistic timed automata to steady-state distributions, but this will be the subject of further research.

Table 2. Iterating the operator Ψ^2 of the running example

n	$g_n(x) = v_{(ab)^n}(x)$	$\|g_n\|$	ρ_{n-1}
0	1	1	
1	$1 - x - (1-x)^2/2$	1/2	0.5
2	$(1-x)/3 + (1-x)^4/24 - (1-x)^3/6$	5/24	0.41667
3	$\frac{2}{15}(1-x) - (1-x)^6/720 + (1-x)^5/120 - (1-x)^3/18$	61/720	0.40667
4	$\frac{17}{315}(1-x) + (1-x)^8/40320 - (1-x)^7/5040+$ $(1-x)^5/360 - (1-x)^3/45$	277/8064	0.40544

References

1. Alur, R., Dill, D.L.: A theory of timed automata. TCS 126, 183–235 (1994)
2. Asarin, E., Basset, N., Béal, M.-P., Degorre, A., Perrin, D.: Toward a timed theory of channel coding. In: Jurdziński, M., Ničković, D. (eds.) FORMATS 2012. LNCS, vol. 7595, pp. 27–42. Springer, Heidelberg (2012)
3. Asarin, E., Basset, N., Degorre, A., Perrin, D.: Generating functions of timed languages. In: Rovan, B., Sassone, V., Widmayer, P. (eds.) MFCS 2012. LNCS, vol. 7464, pp. 124–135. Springer, Heidelberg (2012)
4. Asarin, E., Degorre, A.: Volume and entropy of regular timed languages: Analytic approach. In: Ouaknine, J., Vaandrager, F.W. (eds.) FORMATS 2009. LNCS, vol. 5813, pp. 13–27. Springer, Heidelberg (2009)
5. Asarin, E., Degorre, A.: Volume and entropy of regular timed languages: Discretization approach. In: Bravetti, M., Zavattaro, G. (eds.) CONCUR 2009. LNCS, vol. 5710, pp. 69–83. Springer, Heidelberg (2009)
6. Asarin, E., Degorre, A.: Two size measures for timed languages. In: FSTTCS. LIPIcs, vol. 8, pp. 376–387 (2010)
7. Basset, N.: A maximal entropy stochastic process for a timed automaton. In: Fomin, F.V., Freivalds, R., Kwiatkowska, M., Peleg, D. (eds.) ICALP 2013, Part II. LNCS, vol. 7966, pp. 61–73. Springer, Heidelberg (2013)
8. Basset, N., Asarin, E.: Thin and thick timed regular languages. In: Fahrenberg, U., Tripakis, S. (eds.) FORMATS 2011. LNCS, vol. 6919, pp. 113–128. Springer, Heidelberg (2011)
9. Bertrand, N., Bouyer, P., Brihaye, T., Markey, N.: Quantitative model-checking of one-clock timed automata under probabilistic semantics. In: QEST, pp. 55–64 (2008)
10. Chomsky, N., Miller, G.A.: Finite state languages. Information and Control 1(2), 91–112 (1958)
11. Denardo, E.: Periods of connected networks and powers of nonnegative matrices. Mathematics of Operations Research 2(1), 20–24 (1977)
12. Katō, T.: Perturbation Theory for Linear Operators. Springer (1995)
13. Krasnosel'skij, M., Lifshits, E., Sobolev, A.: Positive Linear Systems: The method of positive operators. Heldermann Verlag, Berlin (1989)
14. Lind, D., Marcus, B.: An introduction to symbolic dynamics and coding. Cambridge University Press (1995)
15. Maler, O., Larsen, K.G., Krogh, B.H.: On zone-based analysis of duration probabilistic automata. In: INFINITY. EPTCS, vol. 39, pp. 33–46 (2010)
16. Sassoli, L., Vicario, E.: Close form derivation of state-density functions over DBM domains in the analysis of non-Markovian models. In: QEST, pp. 59–68 (2007)
17. Seneta, E.: Non-Negative Matrices and Markov Chains. Springer (2006)

Robust Weighted Timed Automata and Games*

Patricia Bouyer, Nicolas Markey, and Ocan Sankur

LSV, CNRS & ENS Cachan, France

Abstract. Weighted timed automata extend timed automata with cost variables that can be used to model the evolution of various quantities. Although cost-optimal reachability is decidable (in polynomial space) on this model, it becomes undecidable on weighted timed games. This paper studies cost-optimal reachability problems on weighted timed automata and games under robust semantics. More precisely, we consider two perturbation game semantics that introduce imprecisions in the standard semantics, and bring robustness properties w.r.t. timing imprecisions to controllers. We give a polynomial-space algorithm for weighted timed automata, and prove the undecidability of cost-optimal reachability on weighted timed games, showing that the problem is robustly undecidable.

1 Introduction

Weighted timed automata. Weighted timed automata [4,7] are timed automata [2] enriched with observer variables whose values grow with given constant derivatives in each location. This allows one to describe systems with timing constraints while specifying the evolution of some resources, such as energy. The cost-optimal reachability problem was studied in [4,7,8], and the problem was shown to be PSPACE-complete [8]. The model naturally extends to timed games. Although some partial decidability results for weighted timed games have been proposed in [1,10], the problem is undecidable in the general case [13,9] even for a fixed number of clocks.

Robustness. Timed automata and their extensions are abstract formalisms to describe models of real-time systems. Consequently, these formalisms make idealistic assumptions, such as the perfect continuity of the clocks, their high precision, and instantaneous reaction of the system. One side effect of such idealistic semantics is that they allow easily encoding of undecidable problems. In fact, the undecidability proofs on weighted timed games of [13,9] rely on the ability of timed automata to distinguish infinitely precise clock values, and modify these values with high precision. It is believed that some of the undecidability results in the literature can be overcome by introducing fuzziness in the semantics, making it impossible to encode complex (undecidable) languages, see for instance [3,5,18].

In this paper, we investigate how adding imprecisions to the semantics affects the (un)decidability of the cost-optimal reachability problem in weighted timed

* This work was partly supported by ANR project ImpRo (ANR-10-BLAN-0317), by ERC Starting grant EQualIS (308087) and by European project Cassting (FP7-ICT-601148).

automata and games. We consider two prominent perturbation semantics from [15,11,23], where perturbations are modeled as a game between a controller which chooses delays and edges, and an environment which perturbs the chosen delay by a (parametrized) bounded amount. The problem consists in deciding whether the controller has a winning strategy ensuring a given objective for a sufficiently small value of the parameter. Such a winning strategy is then *robust*, since it can ensure winning even if the moves it suggests are perturbed by a bounded amount. Two variants have been considered: in the *excess-perturbation semantics*, the controller is only required to suggest delays and edges whose guards are satisfied after the delay, while the guard can be violated after perturbations. This semantics allows one to design systems with simple timing constraints (for instance, using equalities), and synthesize robust strategies afterwards, taking into account new behaviors due to imprecisions. The *conservative perturbation semantics* requires the guards to be satisfied after *any* perturbation. This semantics appears naturally for instance in applications where lower and upper bounds on task execution times are given and need to be respected strictly.

For timed automata, robust reachability is EXPTIME-complete for the excess-perturbation semantics [11], while robust reachability and safety are PSPACE-complete for the conservative perturbation semantics [23]. Here, we apply both semantics to weighted timed automata and games and study the problem of deciding an upper bound on the *limit-cost* of a winning strategy for reachability objectives for the controller. The limit-cost refers to the limit of the cost achieved by a given strategy, when the bound δ on the perturbations goes to zero. In fact, the cost of a given path can slightly increase (or decrease) due to perturbations on the delays, but only by an amount proportional to δ. Thus, the limit-cost allows us to concentrate on the bound that could be achieved if this effect is discarded. On weighted timed automata, we prove that the problem is PSPACE-complete in the conservative perturbation semantics, by adapting the corner-point abstraction [8]. In the excess-perturbation semantics, we show, perhaps surprisingly, that the problem becomes undecidable on weighted timed automata. On weighted timed games, our results are negative: in both excess- and conservative perturbation semantics, cost optimal reachability remains undecidable on weighted timed games, even when the number of clocks is fixed and the constants are bounded. We also prove the undecidability of the problem for fixed parameter δ on weighted timed games. Hence, similarly to "robust undecidability" results of [20] on timed automata, we establish that cost-optimal reachability on weighted timed games is robustly undecidable, for the considered semantics.

Related Work. The work [19] attempted to introduce fuzziness in the semantics of timed automata, via a topological semantics, with the hope of extending the decidability results. However timed language inclusion turned out to be still undecidable [20]. Another related line of work is that of [21,17,16], which consists in modeling imprecisions by *enlarging* all clock constraints of the automaton by some parameter δ, transforming each constraint of the form $x \in [a, b]$ into $x \in [a - \delta, b + \delta]$. One analyzes the resulting system with a worst case approach. The game semantics we consider allow the system to observe and react to perturbations. The dual notion of *shrinking* was considered in [22] in order to study whether any significant behavior is lost when guards are shrunk.

The long version [12] of this paper contains the detailed proofs.

2 Preliminaries

Game structures. A *(two-player) game structure* is a tuple $\mathcal{T} = (S, s_0, M_1, M_2, T_1, T_2, \mathsf{jt})$, where S is a set of *states*, $s_0 \in S$ is the *initial state*, M_i is the set of *moves* of Player i, $T_i \subseteq S \times M_i$ is the *enabling condition* for Player i, and $\mathsf{jt}\colon S \times M_1 \times M_2 \to S$ the *joint transition function*. We assume that each M_i contains a distinguished element $\perp$ called the *empty move*, and that $\mathsf{jt}(s, \perp, \perp) = s$ for any $s \in S$. A *run* of $\mathcal{T}$ is a finite or infinite sequence $q_1 e_1 q_2 e_2 \ldots$ of alternating states $q_i \in S$, and pairs of moves $e_i = (m_1, m_2) \in M_1 \times M_2$, such that for all $i \geq 1$, we have $(q_i, m_\iota) \in T_\iota$ for $\iota \in \{1, 2\}$, and $q_{i+1} = \mathsf{jt}(q_i, e_i)$. For any finite run ρ, let $|\rho|$ denote its *length*, that is, the number of states it contains. For any natural number $1 \leq i \leq |\rho|$, let $\mathsf{state}_i(\rho)$ the i-th state of ρ, and $\mathsf{trans}_i(\rho)$ its i-th transition. We let $\mathsf{first}(\rho) = \mathsf{state}_1(\rho)$, and $\mathsf{last}(\rho) = \mathsf{state}_{|\rho|}(\rho)$. We also denote by $\rho_{i\ldots j}$ the sub-run of ρ between states of indices i and j.

A *strategy* for Player i is a function f that maps each finite run h to a move M_i, such that $(\mathsf{last}(h), f(h)) \in T_i$. A run ρ is *compatible* with strategies f and g of Players 1 and 2, if $\mathsf{state}_{i+1}(\rho) = \mathsf{jt}(\rho_{1\ldots i}, (f(\rho_{1\ldots i}), g(\rho_{1\ldots i})))$ for all $i \geq 1$. Given strategies f and g for Players 1 and 2 resp., the *outcome of the pair* (f, g) *in* $\mathcal{T}$, denoted by $\mathsf{Outcome}_{\mathcal{T}}(f, g)$ is the unique infinite run that is compatible with both strategies. Let $S_i(\mathcal{T})$ denote the set of the strategies of Player i in $\mathcal{T}$.

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

An atomic clock constraint is a formula of the form $k \preceq x \preceq' l$ or $k \preceq x - y \preceq' l$ where $x, y \in \mathcal{C}$, $k, l \in \mathbb{Z} \cup \{-\infty, \infty\}$ and $\preceq, \preceq' \in \{<, \leq\}$. A *guard* is a conjunction of atomic clock constraints. A valuation ν satisfies a guard g, denoted $\nu \models g$, if all constraints are satisfied when each $x \in \mathcal{C}$ is replaced with $\nu(x)$. We write $\Phi_{\mathcal{C}}$ for the set of guards built on $\mathcal{C}$.

Definition 1. *A* weighted timed game *(WTG) $\mathcal{A}$ is a tuple $(\mathcal{L}, \ell_0, \mathcal{C}, \mathcal{I}, E_1, E_2, \mathcal{S})$, where $\mathcal{L}$ is a finite set of locations, $\mathcal{C}$ is a finite set of clocks, $\mathcal{I} : \mathcal{L} \to \Phi_{\mathcal{C}}$ assigns an invariant to every location, $E_1, E_2 \subseteq \mathcal{L} \times \Phi_{\mathcal{C}} \times 2^{\mathcal{C}} \times \mathcal{L}$ are sets of edges, $\ell_0 \in \mathcal{L}$ is the initial location, and $\mathcal{S}\colon \mathcal{L} \to \mathbb{Z}$ is the slope function*[1]. *For any edge $e = (\ell, g, R, \ell')$, g is the guard of the edge, and R its reset set. An edge $e = (\ell, g, R, \ell')$ is also written as $\ell \xrightarrow{g,R} \ell'$. A* weighted timed automaton *(WTA) is a WTG of the form $(\mathcal{L}, \ell_0, \mathcal{C}, \mathcal{I}, E_1, \emptyset, \mathcal{S})$.*

[1] We do not introduce discrete weights on transitions, but all our results would also hold in that setting.

Weighted timed games define game structures similarly to timed games [6]: the guards of the edges enable or disable the transitions, and the reset set determines the update after the transition is taken by resetting the clocks belonging to the set. Intuitively, the edges E_i are controlled by Player i. Moreover, the state space contains the value of a cost variable, denoted cost, which grows with derivative $\mathcal{S}(\ell)$ at any location ℓ.

In this paper, we assume that all clocks are bounded above by a constant, *i.e.*, the invariant at each location imposes some upper bound on all clocks.

Formally, the *exact semantics* of a WTG $\mathcal{A}$ is a game structure $[\![\mathcal{A}]\!] = (S, s_0, M_1, M_2, T_1, T_2, \mathsf{jt})$. The state space is $S = \{(\ell, \nu, c) \mid \ell \in \mathcal{L}, \nu \in \mathbb{R}^{\mathcal{C}}_{\geq 0}, c \in \mathbb{R}, \nu \models \mathcal{I}(\ell)\}$. The initial state is $s_0 = (\ell_0, \mathbf{0}, 0)$. The moves are defined by $M_i = \mathbb{R} \times E_i \cup \{\bot\}$ whose components are pairs of a delay and a Player-i edge. A pair $(d, e) \in \mathbb{R} \times E_i$ is said enabled at $(\ell, \nu) \in \mathcal{L} \times \mathbb{R}^{\mathcal{C}}_{\geq 0}$ whenever, writing $e = (\ell_1, g, R, \ell_2)$, we have $\ell = \ell_1$, $\nu \models \mathcal{I}(\ell_1)$, $\nu + d \models \mathcal{I}(\ell_1)$, $\nu + d \models g$, and $(\nu + d)[R \leftarrow 0] \models \mathcal{I}(\ell_2)$. The enabling condition $T_i((\ell, \nu, c), (d, e))$ holds if, and only if (d, e) is enabled at (ℓ, ν). Note that $T_i((\ell, \nu, c), \bot)$ holds at any state (ℓ, ν, c). The joint transition function is defined as follows. Given $d_i \geq 0$, and edges $e_i = (\ell, g_i, R_i, \ell_i) \in E_i$, we have $\mathsf{jt}((\ell, \nu, c), (d_1, e_1), (d_2, e_2)) = (\ell_{i_0}, (\nu + d_{i_0})[R_{i_0} \leftarrow 0], c + d_{i_0}\mathcal{S}(\ell))$, where $i_0 = 1$ if $d_1 \leq d_2$, and $i_0 = 2$ otherwise. Moreover, $\mathsf{jt}((\ell, \nu, c), (d_1, e_1), \bot) = (\ell_1, (\nu + d_1)[R_1 \leftarrow 0], c + d_1\mathcal{S}(\ell))$, and symmetrically.

Example 1. Figure 1 displays an example of a weighted timed game. Plain (resp. dashed) arrows are for Player 1 (resp. Player 2) edges. The slopes are indicated above each state. A strategy for Player 1 is to suggest a delay of 1 and choose the edge from ℓ_1 to ℓ_2. This prevents Player 2 from going down to location ℓ_5, where the cost of accepting is $12 - o(\delta)$. From location ℓ_2, Player 1 can go to ℓ_4, from where a target location is reached with cost 7.

Regions, Vertices. We assume familiarity with *regions* (see [2]). We recall that the *region automaton* of a timed automaton $\mathcal{A}$ is a finite automaton, denoted $\mathcal{R}(\mathcal{A})$, with states (ℓ, r), where ℓ is a location, and r a region. Let us write $\mathsf{loc}((\ell, r)) = \ell$. There is a transition $(\ell, r) \xrightarrow{\mathsf{delay}} (\ell, r')$ iff some valuation in r' can be reached by a time delay from some valuation in r. We have $(\ell, r) \xrightarrow{e} (\ell', r')$, for an edge $e = (\ell, g, R, \ell')$ iff all valuations of r satisfy g, and $r' = r[R \leftarrow 0]$. A path of $\mathcal{R}(\mathcal{A})$ is a sequence $q_1 t_1 q_2 t_2 \ldots$ where for all $i \geq 1$, $q_i = (\ell_i, r_i)$ for some location ℓ_i and region r_i, and t_i is either an edge or delay. We say that a run ρ of $\mathcal{A}$ follows a path $\pi = q_1 t_1 \ldots$ of $\mathcal{R}(\mathcal{A})$ if for any $i \geq 1$, if we write $(\ell_i, r_i) = \mathsf{state}_i(\pi)$,

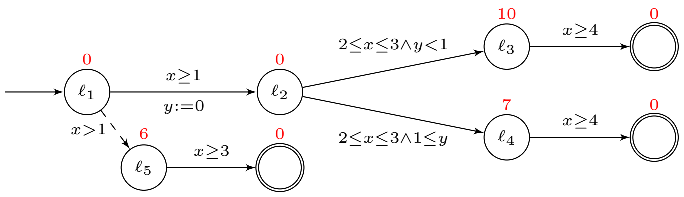


Fig. 1. Example of a WTG

then $\mathsf{state}_i(\rho) = (\ell_i, \nu_i)$ for some $\nu_i \in r_i$, and moreover, $\mathsf{trans}_i(\pi) = \mathsf{delay}$ implies that $\mathsf{trans}_i(\rho) \in \mathbb{R}_{\geq 0}$, and $\mathsf{trans}_i(\pi) = \mathsf{trans}_i(\rho)$ otherwise.

A valuation with integer coordinates is called a *vertex*. For any region r, let us denote by $\mathcal{V}(r)$ the set of vertices that belong the topological closure of r. A region is *non-punctual* if for some $\nu \in r$, and $\epsilon > 0$, $\nu + [-\epsilon, \epsilon] \subseteq r$. It is *punctual* otherwise. A *non-punctual path* of the region automaton is a path of $\mathcal{R}(\mathcal{A})$ where all regions reached after a delay are non-punctual.

3 Robust Cost-Optimal Reachability

We now define two perturbed semantics for weighted timed games. These semantics were first studied in [14,11] for timed automata and games in order to synthesize robust strategies. We adapt these here to WTG and formalize cost-optimal reachability problems.

Perturbed semantics. We will call Player 1 Controller, and Player 2 Perturbator. The idea behind the perturbed semantics is to give Perturbator the additional power of perturbing the delays chosen by Controller by some bounded amount δ (in that sense, the perturbed semantics becomes asymmetric). In this setting, a winning strategy for Controller is then robust to perturbations in the time delays. Informally, at any state (ℓ, ν, c), both players suggest a delay and an edge. If Perturbator suggests a shorter delay, then the suggested delay and edge is taken. If Controller suggests the shorter delay d and edge e, then the system moves to an intermediate state (ℓ, ν, c, d, e), from which Perturbator chooses $\epsilon \in [-\delta, \delta]$, and the edge e is taken after a delay of $d + \epsilon$; the cost then increases by $(d + \epsilon) \cdot \mathcal{S}(\ell)$. We require both players to only suggest delays no smaller than δ, to model the fact that the system is not infinitely fast. We will formally define two perturbed semantics based on the above ideas; they will differ on the satisfaction or not of the guards after the delay has been perturbed by ϵ.

Formally, given $\delta > 0$, the *δ-excess perturbation semantics* of a WTG $\mathcal{A} = (\mathcal{L}, \ell_0, \mathcal{C}, \mathcal{I}, E_1, E_2, \mathcal{S})$ is a game structure $\mathcal{G}^{e}_{\delta}(\mathcal{A}) = (S', s'_0, M'_1, M'_2, T'_1, T'_2, \mathsf{jt}')$, where $S' = S \cup S \times \mathbb{R}_{\geq 0} \times E_1$, with $S = \{(\ell, \nu, c) \mid \ell \in \mathcal{L}, \nu \in \mathbb{R}^{\mathcal{C}}_{\geq 0}, c \in \mathbb{R}, \nu \models \mathcal{I}(\ell)\}$. The initial state is $s'_0 = (\ell_0, \mathbf{0}, 0)$. We have $M'_1 = [\delta, \infty) \times E_1$, and $M'_2 = [\delta, \infty) \times E_2 \cup [-\delta, \delta]$. The enabling conditions T'_i are as follows. For any $i \in \{1, 2\}$, from any state $(\ell, \nu, c) \in S'$, we have $((\ell, \nu, c), d, e) \in T'_i$ for any $d \geq \delta$ and $e \in E_i$ whenever (d, e) is enabled at (ℓ, ν). We also have $(\ell, \nu, c), \bot) \in T'_i$. For states $(\ell, \nu, c, d, e) \in S'$, we have $((\ell, \nu, c, d, e), \bot) \in T'_1$ and $((\ell, \nu, c, d, e), \epsilon) \in T'_2$ for any $\epsilon \in [-\delta, \delta]$. The joint transition function δ respects the shorter delay:

$$\mathsf{jt}'((\ell, \nu, c), (d_1, e_1), (d_2, e_2)) = \begin{cases} (\ell, \nu, c, d_1, e_1) & \text{if } d_1 \leq d_2, \\ (\ell_2, (\nu + d_2)[R' \leftarrow 0], c + d_2 \cdot \mathcal{S}(\ell)) & \text{if } d_1 > d_2. \end{cases}$$

Moreover, in case one player plays $\bot$, we let $\mathsf{jt}'((\ell, \nu, c), (d, e_1), \bot) = (\ell, \nu, c, d, e_1)$ and $\mathsf{jt}'((\ell, \nu, c), \bot, (d', e_2)) = (\ell_2, (\nu + d')[R' \leftarrow 0], c + d' \cdot \mathcal{S}(\ell))$, as expected. Last, we let $\mathsf{jt}'((\ell, \nu, c, d, e_1), \bot, \epsilon) = (\ell_1, (\nu + d + \epsilon)[R \leftarrow 0], c + (d + \epsilon) \cdot \mathcal{S}(\ell))$.

Thus, the cost variable grows with derivative $\mathcal{S}(\ell)$ at location ℓ, and the sojourn time is either the delay suggested by Perturbator if it is shorter, or the delay suggested by Controller and perturbed by Perturbator otherwise. Notice that, in this semantics the guard of an edge that is taken need not be satisfied after a perturbation (hence the term *excess*).

We also consider another natural semantics for perturbation, which we call the *δ-conservative perturbation semantics* and denote $\mathcal{G}^{c}_{\delta}(\mathcal{A})$. This semantics is defined similarly with the only difference that the enabling condition for Controller from states (ℓ, ν, c) are defined as follows. From any state $(\ell, \nu, c) \in S'$, we have $((\ell, \nu, c), d, e) \in T'_1$ for any $d \geq \delta$ and $e \in E_1$ whenever $(d + \epsilon, e)$ is enabled at (ℓ, ν) for every $\epsilon \in [-\delta, \delta]$. In other terms, Controller should only suggest delays and edges whose guards are enabled under any perturbation of the delay. Consequently, this semantics forbids equality constraints, since these are never enabled.

Example 2. Figure 2 explains the differences between our two perturbation semantics: Controller has to suggest a delay such that the resulting valuation does not end up in the grey area; Perturbator can then shift this delay by $[-\delta, \delta]$. In the conservative semantics, no new behaviors are added, because the final delay chosen by Perturbator will satisfy the guard; in the excess semantics, new behaviors may occur because neighboring regions can be reached.

Example 1 (Cont'd). We come back to the WTG of Fig. 1, to illustrate the differences between the two perturbed semantics. Under the excess-perturbation semantics, as in the exact case, Controller can suggest a delay of 1 and choose the edge from ℓ_1 to ℓ_2. The location ℓ_5 can thus be avoided. Now, one can see that the move of Perturbator determines the next location to be visited: if Perturbator adds a positive perturbation (*i.e.* if the delay is in $[1, 1 + \delta]$), then only location ℓ_3 is reachable. Conversely, a negative perturbation enables only location ℓ_4. To maximize the cost, Perturbator will force the play to ℓ_3, so Controller can only ensure a cost of $10 + \Theta(\delta)$.

We now focus on the conservative semantics. The above strategy is no more valid since in this case, Controller can only suggest delays of at least $1 + \delta$. Then Perturbator can force the play to ℓ_5. Here, the cost of winning is $12 + \Theta(\delta)$.

Cost-optimal reachability. We define cost-optimal reachability problems that take into account the perturbed semantics. We are interested in computing strategies for reachability which minimize the cost when the parameter δ goes to 0.

First notice that $S_1(\mathcal{G}^{e}_{\delta}(\mathcal{A}))$ does not depend on δ, since Controller only has to suggest moves that satisfy the guards (a *winning* strategy will depend on δ

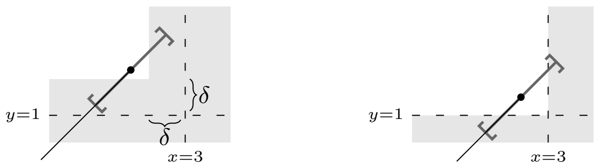


Fig. 2. The conservative- (left) and excess semantics (right) for a transition guarded with $x \leq 3 \wedge y \geq 1$. The grey area corresponds to forbidden delays, the bullet corresponds to the choice of Controller, and the segment indicates the possible choices for Perturbator.

though). In contrast, $S_1(\mathcal{G}^c_\delta(\mathcal{A}))$ does depend on δ since Controller is required to satisfy the guards even after perturbations. It is easy to see that $S_1(\mathcal{G}^c_\delta(\mathcal{A})) \subseteq S_1(\mathcal{G}^c_{\delta'}(\mathcal{A}))$ for any $\delta' < \delta$. We denote $S_1(\mathcal{G}^c(\mathcal{A})) = \bigcup_{\delta>0} S_1(\mathcal{G}^c_\delta(\mathcal{A}))$.

Let us write $(\ell, \nu, c)|_{\mathsf{cost}} = c$, and $(\ell, \nu, c, d, e)|_{\mathsf{cost}} = c$, the projection of a state to the cost value. For any run ρ of the exact or perturbed semantics, we define the *cost of ρ w.r.t. a location ℓ* as, $\mathsf{cost}^\ell(\rho) = \inf\{\mathsf{state}_i(\rho)|_{\mathsf{cost}} \mid 1 \leq i < |\rho| + 1, \mathsf{loc}(\mathsf{state}_i(\rho)) = \ell\}$ (Note that the definition includes the case where $|\rho| = \infty$). Hence, if ℓ is never reached, then the cost is ∞. Otherwise it is the infimum of the costs observed at location ℓ. Given $\delta > 0$, a pair of strategies $(\sigma, \sigma') \in S_1(\mathcal{G}^e_\delta(\mathcal{A})) \times S_2(\mathcal{G}^e_\delta(\mathcal{A}))$, and location ℓ, we define $\mathsf{cost}^\ell_{\sigma,\sigma'}(\mathcal{G}^e_\delta(\mathcal{A})) = \mathsf{cost}^\ell(\mathsf{Outcome}_{\mathcal{G}^e_\delta(\mathcal{A})}(\sigma, \sigma'))$. We define similarly $\mathsf{cost}^\ell_{\sigma,\sigma'}(\mathcal{G}^c_\delta(\mathcal{A}))$. Given a strategy $\sigma \in S_1(\mathcal{G}^e_\delta(\mathcal{A})$, we define the *limit-cost of σ* as $\mathsf{lim\text{-}cost}^{\mathrm{exs}}_\sigma(\mathcal{A}, \ell) = \lim_{\delta\to 0} \sup_{\sigma' \in S_2(\mathcal{G}^e_\delta(\mathcal{A}))} \mathsf{cost}^\ell_{\sigma,\sigma'}(\mathcal{G}^e_\delta(\mathcal{A})))$. The limit is well defined since strategy σ is valid for any $\delta > 0$. Similarly, for $\sigma \in \mathcal{G}^c_{\delta_0}(\mathcal{A})$, we let $\mathsf{lim\text{-}cost}^{\mathrm{cons}}_\sigma(\mathcal{A}, \ell) = \lim_{\substack{\delta\to 0\\ 0<\delta<\delta_0}} \sup_{\sigma' \in S_1(\mathcal{G}^c_\delta(\mathcal{A}))} \mathsf{cost}^\ell_{\sigma,\sigma'}(\mathcal{G}^c_\delta(\mathcal{A}))$. Here, we take the limit for $0 < \delta < \delta_0$, such that $\sigma \in S_2(\mathcal{G}^c_{\delta_0}(\mathcal{A}))$ so that the strategy is valid for any considered δ. Thus, the limit-cost of a Controller's strategy σ is the cost it guarantees in the limit, against any strategy of Perturbator, when δ goes to 0.

We are interested in deciding whether Controller has a strategy that guarantees an upper bound on the limit-cost for a reachability objective.

Definition 2. *The* limit strategy (strict) upper-bound problem for the excess perturbation semantics *asks, given a weighted timed game $\mathcal{A}$, a location ℓ, and a rational λ, whether there exists a strategy $\sigma \in S_1(\mathcal{G}^e_\delta(\mathcal{A}))$ such that $\mathsf{lim\text{-}cost}^{exs}_\sigma(\mathcal{A}, \ell) \leq \lambda$ (resp. $\mathsf{lim\text{-}cost}^{exs}_\sigma(\mathcal{A}, \ell) < \lambda$). Similarly, we define the* limit strategy (strict) upper-bound problem for the conservative perturbation semantics.

We define the *limit-value* as the infimum of the limit-cost that can be guaranteed by Controller: $\mathsf{lim\text{-}value}^{\mathrm{exs}}(\mathcal{A}, \ell) = \inf_{\sigma \in S_1(\mathcal{G}^e_\delta(\mathcal{A}))} \mathsf{lim\text{-}cost}^{\mathrm{exs}}_\sigma(\mathcal{A}, \ell)$, and $\mathsf{lim\text{-}value}^{\mathrm{cons}}(\mathcal{A}, \ell) = \inf_{\sigma \in S_1(\mathcal{G}^c(\mathcal{A}))} \mathsf{lim\text{-}cost}^{\mathrm{cons}}_\sigma(\mathcal{A}, \ell)$. We also consider deciding upper bounds on values:

Definition 3. *The* limit value upper-bound problem for the excess (resp. conservative) perturbation semantics *asks whether given a weighted timed automaton $\mathcal{A}$, a target location ℓ, and a rational λ, it holds $\mathsf{lim\text{-}value}^{exs}(\mathcal{A}, \ell) \leq \lambda$ (resp. $\mathsf{lim\text{-}value}^{cons}(\mathcal{A}, \ell) \leq \lambda$)?*

Notice that the strategy strict upper-bound problem is equivalent to deciding whether the strict upper bound holds for the *value*, that is the infimum of the limit cost over all possible strategies. We will consider the restriction of both problems on WTA.

Example 1 (Cont'd). We come back to the WTG of Fig. 1. We have seen the impact of the choice of the semantics on the possible behaviors of the system. In particular, the limit-optimal cost in the conservative (resp. excess) semantics is equal to 12 (resp. 10).

Theorem 1. *The limit strategy upper-bound problem for WTA (and WTG) is undecidable under the excess perturbation semantics, for a fixed number of clocks.*

Theorem 1 is a rather surprising result. It reveals that adding perturbations can render problems intractable, which is the opposite of a common belief [5,18]. In this case, optimal reachability is PSPACE-complete for weighted timed automata under the exact semantics, but becomes undecidable under the excess perturbation semantics.

The conservative robust semantics is more restrictive than the excess perturbation semantics. In timed automata, reachability under the conservative semantics is PSPACE-complete [23], in contrast with the EXPTIME-completeness under the excess perturbation semantics [11]. For weighted timed automata, the conservative semantics renders the problem tractable; the limit value upper-bound problem is PSPACE-complete:

Theorem 2. *The limit value upper-bound problem is* PSPACE-*complete on WTA under the conservative perturbation semantics.*

The algorithm is based on the corner-point abstraction [8], but requires eliminating *punctual* regions, following the ideas in [23]. However the conservative semantics does not allow the treatment of weighted timed games:

Theorem 3. *The limit strategy strict upper-bound problem is undecidable for WTG under the conservative perturbation semantics, for a fixed number of clocks.*

The undecidability also holds in both semantics on WTGs when δ is fixed:

Theorem 4. *The following problem is undecidable: For any fixed $0 \leq \delta \leq \frac{1}{3}$, given a WTG $\mathcal{A}$, a target location ℓ, and a rational λ, decide whether it holds $\inf_{\sigma \in S_1(G)} \sup_{\sigma' \in S_2(G)} \mathsf{cost}^{\ell}_{\sigma,\sigma'}(G) < \lambda$, where G denotes either $\mathcal{G}^e_\delta(\mathcal{A})$ or $\mathcal{G}^c_\delta(\mathcal{A})$.*

In Section 4, we present our results on WTA, that is, the algorithm of Theorem 2 and the undecidability result of Theorem 1. The long version [12] of this paper contains the detailed proofs of these results, and of Theorems 3 and 4.

4 Weighted Timed Automata

4.1 Algorithm in the Conservative Semantics

We present a polynomial-space algorithm for the limit value upper-bound problem, based on a variant of the corner-point abstraction [8]. The idea behind our algorithm is that Perturbator can always avoid punctual regions by adding an infinitesimal perturbation. Thus, one needs to remove punctual delay transitions from the corner-point abstraction. It turns out that the resulting construction suffices to solve the limit-cost value for a given WTA.

A *finite weighted automaton* over $(\mathbb{Z}, +)$ is a tuple $\mathcal{F} = (S, s_0, \Sigma, T, W)$, where S is the set of states, $s_0 \in S$ the initial state, $T \subseteq S \times \Sigma \times S$ the set of transitions, and $W \colon E \to \mathbb{Z}$ is the *weight function.* A *path* (or *run*) of a finite weighted automaton is a (finite or infinite) sequence $q_1 t_1 q_2 t_2 \ldots$ alternating states and transitions and such that for all $i \geq 1$, $t_i = (q_i, \sigma_i, q_{i+1})$ for some σ_i. We write $\mathsf{Runs}(\mathcal{F})$ for the set of runs of $\mathcal{F}$ starting in the initial state s_0. The length, first and last states and sub-runs are defined in the same way as for runs of a game structure. A finite weighted automaton then associates to any finite path the sum of the weights of the edges it visits. Given any path $\pi = q_1 t_1 q_2 \dots q_n$, the *weight of* π is defined as $W(\pi) = \sum_{1 \leq i < n} W(t_i)$.

Let us consider a weighted timed automaton $\mathcal{A} = (\mathcal{L}, \ell_0, \mathcal{C}, E_1, \emptyset, \mathcal{S})$. Notice that, following Def. 1, we write it as a weighted timed games with no edges belonging to Player 2. The *corner-point abstraction* of $\mathcal{A}$ is a finite weighted automaton, denoted $\mathcal{R}_{\mathsf{cp}}(\mathcal{A})$. The states of $\mathcal{R}_{\mathsf{cp}}(\mathcal{A})$ are triples (ℓ, r, v), where ℓ is a location, r a region, and $v \in \mathcal{V}(r)$ a vertex of r. Edges are defined as follows: we have $(\ell, r, v) \xrightarrow{\mathsf{delay}} (\ell, r', v')$ if $(\ell, r) \xrightarrow{\mathsf{delay}} (\ell, r')$ in the region automaton, and $v' = v + k$ for some natural number k. In other terms, v' is a time-successor of v, and is a vertex of region r'. The weight associated to this transition is $k \times \mathcal{S}(\ell)$. Further, we have an edge $(\ell, r, v) \xrightarrow{e} (\ell', r', v')$ if $e = (\ell, g, \sigma, R, \ell')$ is an edge of $\mathcal{A}$ such that $r \models g$, and $r' = r[R \leftarrow 0]$, $v' = v[R \leftarrow 0]$. Such an edge has weight 0. Observe that $\mathcal{R}_{\mathsf{cp}}(\mathcal{A})$ is finite since all clocks are assumed to be bounded. Notice that a path in the corner-point abstraction corresponds to a path of the WTA that runs arbitrarily close to vertices of the regions it visits.

Let the *non-punctual corner-point abstraction*, denoted $\mathcal{R}^{\mathsf{np}}_{\mathsf{cp}}(\mathcal{A})$, be the finite weighted automaton obtained from the corner-point abstraction by removing any transition of the form $(\ell, r, v) \xrightarrow{\mathsf{delay}} (\ell, r', v')$, where r' is punctual. Thus, any path in the non-punctual corner-point abstraction corresponds to a non-punctual path in the region automaton.

For any path π of the region automaton $\mathcal{R}(\mathcal{A})$ of $\mathcal{A}$, we denote by $\mathsf{Runs}(\pi)$, the set of runs of $[\![\mathcal{A}]\!]$ that follow π. If π is a path of the corner-point abstraction $\mathcal{R}_{\mathsf{cp}}(\mathcal{A})$, then we say that a run follows π if it follows the path projected to $\mathcal{R}(\mathcal{A})$ (that is, obtained by removing vertices in each state). We extend the notation $\mathsf{Runs}(\pi)$ to paths π of the corner-point abstraction. For any path π of $\mathcal{R}(\mathcal{A})$ or $\mathcal{R}_{\mathsf{cp}}(\mathcal{A})$, let us define $\bar{\pi}$ obtained from π by replacing all regions by their topological closures. We will consider $\mathsf{Runs}(\bar{\pi})$ which is the set of runs visiting the topological closures of the regions of π. In other terms, this is the topological closure of the set $\mathsf{Runs}(\pi)$.

We define $\mathsf{value}(\mathcal{F}, s)$ for a finite weighted automaton $\mathcal{F}$ and state s as the cost of the shortest path from the initial state to s. Formally, for any finite weighted automaton $\mathcal{F} = (S, s_0, \Sigma, T, W)$, and $s \in S$, we let $\mathsf{value}(\mathcal{F}, s) = \inf\{W(\pi) \mid \pi \in \mathsf{Runs}(\mathcal{F}), \mathsf{last}(\pi) = s\}$. For corner-point abstractions, we extend this notation to locations: $\mathsf{value}(\mathcal{R}_{\mathsf{cp}}(\mathcal{A}), \ell) = \inf\{W(\pi) \mid \pi \in \mathsf{Runs}(\mathcal{R}_{\mathsf{cp}}(\mathcal{A})), \mathsf{loc}(\mathsf{last}(\pi)) = \ell\}$.

In the exact semantics, results of [8] show that the infimum of the cost of the runs of a WTA following a given path π of the corner-point abstraction is achieved by a run that follows $\bar{\pi}$, and only visits vertices. Hence, to compute the infimum cost, it suffices to compute the value of the corner-point abstraction. In the conservative perturbed case, we prove that the same algorithm can be applied, once we discard punctual paths.

Lemma 1. *For any weighted timed automaton $\mathcal{A}$ and target location ℓ, we have $\textsf{lim-value}^{cons}(\mathcal{A}, \ell) = \textsf{value}(\mathcal{R}^{np}_{cp}(\mathcal{A}), \ell)$.*

Theorem 2 follows from the previous lemma. In fact, to compute the optimal cost on $\mathcal{A}$, it suffices to consider the finite weighted automaton $\mathcal{R}^{\mathsf{np}}_{\mathsf{cp}}(\mathcal{A})$, and find the shortest path to location ℓ. To decide whether the limit value is less than some given constant λ, one can guess a path in $\mathcal{R}^{\mathsf{np}}_{\mathsf{cp}}(\mathcal{A})$ in polynomial-space (such a path can be constructed on-the-fly in polynomial space, see *e.g.* [8]), and check whether its weight is less than or equal to λ. Note that the problem is PSPACE-hard since it already is in the unweighted case.

4.2 Undecidability under Excess Perturbation

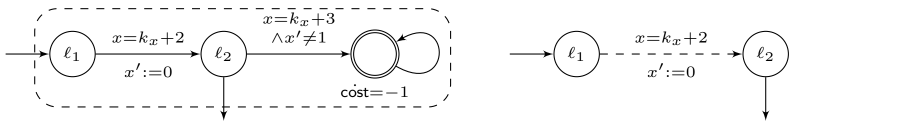
We present the proof of Theorem 1, showing the undecidability of the limit strategy upper-bound problem for WTA with excess perturbation. Our proof is based on a reduction from the halting problem of Minsky machines, following the encoding of [9]. Compared to the reductions of earlier work [13,9], special care needs to be taken when dealing with perturbations, since the present semantics disables precise moves.

We consider a Minsky machine with counters c_1 and c_2, and a list of instructions $I_1, \ldots, I_n$. Here, each instruction I_i, for $1 \le i \le n-1$, is an *incrementation for* c_b as, $c_b = c_b + 1$; `goto` I_j, for $b = 1$ or 2, or a *decrementation with zero-test for* c_b as, `if` $(c_b = 0)$ `goto` I_j `else` $c_i = c_i - 1$; `goto` $I_{j'}$. The instruction I_n is the *ending instruction*, that is, the final state. The halting problem asks whether the instruction I_n is reachable, starting from the configuration $c_1 = 0$, $c_2 = 0$, at instruction I_1.

Our reduction uses 10 clocks $x, x', y, y', u, u', t, t', z, z'$. A counter of a Minsky machine with value n will be encoded by a pair of clocks x, x' with values $k_x + \frac{1}{2^n}$ and $k_{x'} + \frac{1}{2^n}$ for some integers $k_x, k_{x'}$. Here, k_x is called the *shift* of x. If α denotes the clock x', we let $\alpha' = x$, and $\alpha'' = x'$, and similarly for other clocks. A configuration of a Minsky machine with counter values $n, m \ge 0$, is entirely encoded by four clocks:

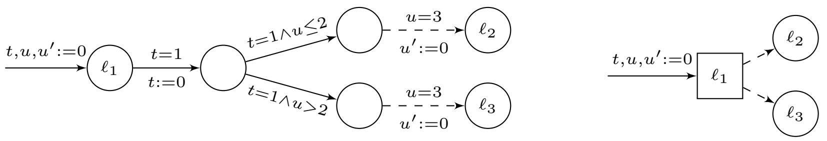
$$x = k_x + \frac{1}{2^n} \qquad x' = k_{x'} + \frac{1}{2^n} \qquad y = k_y + \frac{1}{2^m} \qquad y' = k_{y'} + \frac{1}{2^m} \tag{1}$$

for some shifts $k_x, k_{x'}, k_y, k_{y'}$. The redundancy in this encoding is necessary to cope with perturbations; this will be clear in the constructions. We denote by $\boldsymbol{k}$ the vector of shifts, for all clocks, and by $\mathsf{code}_{\boldsymbol{k}}(n, m)$ the set of valuations satisfying (1). We also define $\mathsf{code}^{\epsilon}_{\boldsymbol{k}}(n, m)$ the set of valuations ν such that $\nu + \eta \in \mathsf{code}_{\boldsymbol{k}}(n, m)$, where $|\eta(\alpha)| \le \epsilon$, $\eta(\alpha) = \eta(\alpha')$ for all clocks α, and moreover $\eta(x) = \eta(x') = 0$ whenever $n = 0$, and $\eta(y) = \eta(y') = 0$ whenever $m = 0$. In other terms, $\mathsf{code}^{\epsilon}_{\boldsymbol{k}}(n, m)$ is the set of valuations that encode a configuration with an error bounded by ϵ, except that the encoding of the counter value 0 is exact. Given shifts $\boldsymbol{k}$ and a valuation $\nu \in \mathsf{code}^{\epsilon}_{\boldsymbol{k}}(n, m)$, we denote $\mathsf{fracp}(\nu) = \nu - \boldsymbol{k}$. This gives the fractional part of the clocks, except when $n = 0$, in which case the



(a) An unperturbed edge. (b) A simpler representation of that edge.

Fig. 3. Unperturbed edges. Perturbator has interest in not perturbing these transitions, since otherwise Controller can go to the target location and win with cost $-\infty$.



(a) Letting Perturbator decide. (b) A simpler representation.

Fig. 4. Module that lets Perturbator decide a successor among ℓ_2 and ℓ_3

components x and x' are equal to 1, and similarly for y. We say that a valuation ν encodes a configuration (n, m) of the machine if it satisfies (1) for some $\boldsymbol{k}$.

We define modules for incrementation and decrementation with zero-test instructions, which will, once combined, yield the reduction. The modules will be defined on a given list of clocks. For instance, if we describe a module $M(x, y, z, u, t)$ that uses the clocks x, y, z, u, and t in its definition, then $M(z, y, x, t, u)$ is obtained simply by exchanging x and z, and u and t.

Unperturbed edges. Let us first present a construction that prevents Perturbator from perturbing the delays along an edge. The construction only applies to deterministic transitions (with equality constraints) and requires resetting one of the two clocks used in the encoding of a counter. Consider the timed automaton of Fig. 3(a). At ℓ_2, Controller can go to the accepting state where the cost decreases to $-\infty$ if, and only if Perturbator has perturbed by a nonzero amount the transition from ℓ_1 to ℓ_2. Thus, Perturbator does not have interest in perturbing since its objective is to maximize the cost. If there has been no perturbation, the clock values are only increased or decreased by some integers. More precisely, the shifts of all clocks but x' increase by 2, and the shift of x' becomes 0. In the rest, we will use this trick extensively to construct our modules. For better readability, we will represent such *unperturbed edges* by dashed arrows; when clear from context, we may omit the edges leading to accepting sink states (see Fig. 3(b)).

Ask-Perturbator module. In weighted timed automata, unlike in weighted timed games, Perturbator cannot suggest moves since it controls no edges. However, a special construction allows letting Perturbator decide the successor location. We describe in Fig. 4(a) such a construction, which also ensures that the configuration is preserved, up to shifts.

The first edge is deterministic, and Perturbator *can* add any perturbation. Controller then distinguishes between positive and negative perturbations, and only has one possible move accordingly. We disallow perturbations (using

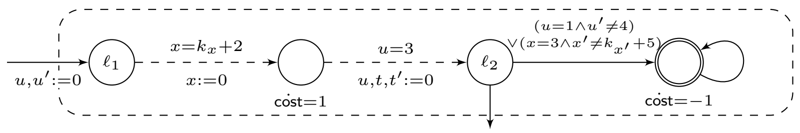


Fig. 5. Module $\mathsf{Add}_{\boldsymbol{k}}^{1+x}(x,u,t)$

unperturbed edges) at the edges leading to ℓ_2 or ℓ_3, so that the configuration is preserved up to shifts. More precisely, the shifts of all clocks x, x', y, y' increase by 3. To simplify the presentation of more complex modules, we will represent this module more compactly as in Fig. 4(b).

Reduction module. In the above modules, we have seen that configurations are preserved up to shifts, but shifts could grow. We present a module that reduces the shifts of all clocks. The module $\mathsf{Reduce}_{\boldsymbol{k}}(x,y,u,t)$ is constructed for each shift vector $\boldsymbol{k}$ (there will be a finite number of these), is deterministic, and constructed using unperturbed edges. The definition of the module is omitted (see [12]); the following lemma summarizes its properties.

Lemma 2. *Let $\delta < \frac{1}{2}$. Assume $\mathsf{Reduce}_{\boldsymbol{k}}(x,y,u,t)$ is entered with valuation $\nu \in \mathsf{code}^{\epsilon}_{\boldsymbol{k}}(n,m)$ for some $\epsilon < \frac{1}{2}$. Controller has a strategy to either go to the target location with cost $-\infty$ or to reach location ℓ_2 with valuation ν' satisfying $\nu' = \mathsf{fracp}(\nu) + \boldsymbol{k'}$ where $\boldsymbol{k'}$ is defined as follows: $k'_x = 6, k'_{x'} = 2, k'_y = 5, k'_{y'} = 1$.*

Test Module. In order to verify the incrementation and decrementation, we use the cost variable. We first show how one can add $1 + \mathsf{fracp}(x)$ and $2 - \mathsf{fracp}(x)$ to the cost variable, without changing the configuration. The construction is similar to [9]; we adapt it using unperturbed edges. The module $\mathsf{Add}_{\boldsymbol{k}}^{1+x}(x,u,t)$ depicted in Fig. 5 adds $1+\mathsf{fracp}(x)$ to the cost, leaving the configuration unchanged (up to shifts).

Lemma 3. *Let $\delta < \frac{1}{2}$. Assume module $\mathsf{Add}_{\boldsymbol{k}}^{1+x}(x,u,t)$ is entered with valuation $\nu \in \mathsf{code}^{\epsilon}_{\boldsymbol{k}}(n,m)$ for some $\epsilon < \frac{1}{2}$. Controller has a strategy that ensures either reaching a target location with cost $-\infty$ or location ℓ_2 with valuation ν' satisfying $\nu' = \mathsf{fracp}(\nu) + \boldsymbol{k'}$ where $k'_x = 1$, and $k'_\alpha = k_\alpha + 3$ for all $\alpha \in \{x', y, y', z, z'\}$, while the cost increases by $1 + \mathsf{fracp}(x)$.*

We define similarly a module $\mathsf{Add}_{\boldsymbol{k}}^{2-x}(x,u,t)$ that adds $2 - \mathsf{fracp}(x)$ to the cost variable. The module is similar to the one of Fig. 5, except that cost only increases (with slope 1) at location ℓ_1.

Lemma 4. *Let $\delta < \frac{1}{2}$. Assume module $\mathsf{Add}_{\boldsymbol{k}}^{2-x}(x,u,t)$ is entered with valuation $\nu \in \mathsf{code}^{\epsilon}_{\boldsymbol{k}}(n,m)$ for some $\epsilon < \frac{1}{2}$. Controller has a strategy that ensures either reaching a target location with cost $-\infty$ or location ℓ_2 with valuation ν' satisfying $\nu' = \mathsf{fracp}(\nu) + \boldsymbol{k'}$ where $k'_x = 1$, and $k'_\alpha = k_\alpha + 3$ for all $\alpha \in \{x', y, y', z, z'\}$, while the cost increases by $2 - \mathsf{fracp}(x)$.*

Concatenating modules $\mathsf{Add}_{\boldsymbol{k}}^{1+x}(x,u,t)$, $\mathsf{Add}_{\boldsymbol{k'}}^{2-x}(z,t,u)$ and $\mathsf{Add}_{\boldsymbol{k''}}^{2-x}(z,u,t)$, we obtain the module $\mathsf{Add}_{\boldsymbol{k}}^{5+x-2z}(x,z,u,t)$, which adds $5 + \mathsf{fracp}(x) - 2\mathsf{fracp}(z)$ to the cost and leads to a target location (we make the last location accepting). Similarly, using concatenation, we define $\mathsf{Add}_{\boldsymbol{k}}^{4+2z-x}(x,z,u,t)$, which increases the cost by $4 + 2\mathsf{fracp}(z) - \mathsf{fracp}(x)$. One can append at the end of module $\mathsf{Add}_{\boldsymbol{k}}^{4+2z-x}(x,z,u,t)$ an edge that increments the cost by 1, which yields module $\mathsf{Add}_{\boldsymbol{k}}^{5+2z-x}(x,z,u,t)$. Finally, module $\mathsf{Test}^{2z=x}(x,z,u,t)$ is defined by letting Perturbator choose whether to go to $\mathsf{Add}_{\boldsymbol{k'}}^{5+2z-x}(x,z,t,u)$ or to $\mathsf{Add}_{\boldsymbol{k'}}^{5+x-2z}(x,z,t,u)$ (here the new shift vector $\boldsymbol{k'}$ is due to the shift added by the ask-Perturbator module). Note that in both cases, the run ends in a target location. The following property follows from Lemmas 3 and 4.

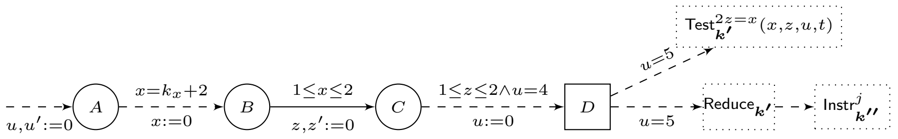
Lemma 5. *Let $\delta < \frac{1}{2}$. If module $\mathsf{Test}^{2z=x}(x,z,u,t)$ is entered with valuation $\nu \in \mathsf{code}_{\boldsymbol{k}}^{\epsilon}(n,m)$ for some $\epsilon < \frac{1}{2}$, Controller has a strategy to ensure reaching a target location with cost at most $5 + |2\mathsf{fracp}(z) - \mathsf{fracp}(x)|$.*

Incrementation Module $\mathsf{Aut}_{\boldsymbol{k}}{}^{c_1,+}$. We define module $\mathsf{Aut}_{\boldsymbol{k}}{}^{c_1,+}$, given in Fig. 6, which simulates the incrementation of counter c_1. We assume first that there are no perturbation. When the module is entered with a valuation in $\mathsf{code}_{\boldsymbol{k}}(n,m)$, we expect Controller to choose the delays so that $z = 1 + \frac{\mathsf{fracp}(x)}{2}$ at location D. From this point on, the clocks z and z' will switch roles with x and x'. Thus, this corresponds to incrementing the counter c_1 by 1. At location D, Perturbator can either decide to "test" the incrementation has been correctly performed by going to the test module, or to continue the simulation by first passing through the reduction module. Here, $\mathsf{Instr}_{\boldsymbol{k''}}^{j}$ refers to a module among $\mathsf{Aut}_{\boldsymbol{k}}{}^{c_b,+}$, $\mathsf{Aut}_{\boldsymbol{k}}{}^{c_b,-}$ for $b \in \{1,2\}$ (to be defined next) according to the instruction I_j. Now, in the presence of perturbations, Perturbator can perturb the value of z chosen by Controller by δ. So at D, if Perturbator goes to the test module the cost is $5 + O(\delta)$, provided that Controller has played correctly. Otherwise, the simulation carries on with $\left| z - \frac{\mathsf{fracp}(x)}{2} \right| \leq \delta$. The following lemma states this formally.

Lemma 6. *Let $\delta < \frac{1}{2}$. Assume module $\mathsf{Aut}_{\boldsymbol{k}}{}^{c_1,+}$ is entered with valuation $\nu \in \mathsf{code}_{\boldsymbol{k}}^{\epsilon}(n,m)$ for some $\epsilon > 0$ with $\delta + \epsilon/2 < \frac{1}{2}$, and cost 0. Then, Controller has a strategy that ensures that either module $\mathsf{Instr}_{\boldsymbol{k}}{}^{j}$ is entered with a valuation $\nu' \in \mathsf{code}_{\boldsymbol{k'}}^{\delta+\epsilon/2}(n+1,m)$ for some $\boldsymbol{k'}$, or the target location is reached with cost at most $5 + 2\delta$.*

A decrementation module can be defined following the same ideas.

Lemma 7. *Assume module $\mathsf{Aut}_{\boldsymbol{k}}{}^{c_1,-}$ is entered with valuation $\nu \in \mathsf{code}_{\boldsymbol{k}}^{\epsilon}(n,m)$ and cost 0. Then,*



$\mathsf{Test}_{\boldsymbol{k'}}^{2z=x}(x,z,u,t)$
$u{=}5$
$x{=}k_x{+}2$
$1{\leq}x{\leq}2$
$1{\leq}z{\leq}2 \wedge u{=}4$
A
B
C
D
$\mathsf{Reduce}_{\boldsymbol{k'}}$
$\mathsf{Instr}_{\boldsymbol{k''}}^{j}$
$u,u'{:=}0$
$x{:=}0$
$z,z'{:=}0$
$u{:=}0$
$u{=}5$

Fig. 6. Module $\mathsf{Aut}_{\boldsymbol{k}}{}^{c_1,+}(x,y,z,u,t)$. The module $\mathsf{Reduce}_{\boldsymbol{k'}}$ refers to $\mathsf{Reduce}_{\boldsymbol{k'}}(z,y,u,t)$, and the module $\mathsf{Instr}_{\boldsymbol{k''}}^{j}$ to $\mathsf{Instr}_{\boldsymbol{k''}}^{j}(z,y,x,u,t)$.

– *If $n = 0$, Controller has a strategy to ensure reaching either $\mathsf{Instr}_{\boldsymbol{k}}{}^{j}$ with the same configuration up to shifts and cost, or an accepting location with cost 0.*
– *If $n \geq 1$, and $\epsilon < \frac{1}{2}$, the play cannot reach $\mathsf{Instr}_{\boldsymbol{k}}{}^{j}$.*
– *If $n \geq 1$, Controller has a strategy that ensures that either module $\mathsf{Instr}_{\boldsymbol{k}}{}^{j'}$ is reached with a valuation $\nu' \in \mathsf{code}_{\boldsymbol{k'}}^{\delta+2\epsilon}(n-1, m)$ for some $\boldsymbol{k'}$, or the target location is reached with cost at most $5 + \epsilon + 2\delta$. Moreover, if $n = 1$, then Controller can ensure that $\nu'(x) = k'_x + 1$.*

Complete reduction. To construct the complete reduction, we define for each instruction I_j of the Minsky machine, a module $\mathsf{Instr}_{\boldsymbol{k}}{}^{j}$ as one of the incrementation or decrementation modules according to the type of I_i. We mark the first location of $\mathsf{Instr}_{\boldsymbol{k}}{}^{1}$ as the initial location. The halting state $\mathsf{Instr}_{\boldsymbol{k}}{}^{n}$ of the machine is an accepting location of the timed automaton. For any machine M, let $\mathcal{A}_M$ denote the weighted timed automaton constructed in this manner, and let ℓ denote the target location obtained by merging all target locations presented in the above construction. Theorem 1 follows from the following proposition.

Proposition 1. *The Minsky machine M halts if, and only if, there is a strategy $\sigma \in S_C(\mathcal{G}^{\mathsf{exs}}(\mathcal{A}_M))$ such that $\mathsf{lim\text{-}cost}_{\sigma}^{exs}(\mathcal{A}_M, \ell) \leq 5$.*

Discussion. One can argue that the undecidability in the excess perturbation game semantics is due to the ability of Controller to test clock values with precision using equality constraints, and in particular in detecting perturbations. This allows for instance letting Perturbator make a discrete choice, as in the above reduction. Hence, this ability and the possibility of disallowing perturbations on some edges make the semantics of weighted timed automata somehow close to that of two-player weighted timed games in the exact semantics for which the optimal-cost reachability is undecidable.

The conservative perturbation game semantics disallows both abilities since Controller is required to suggest delays whose perturbations satisfy the guard of the chosen edge. This excludes equality constraints from guards. Therefore, one cannot encode unperturbed edges nor define the *ask-Perturbator* module as previously. The decidability proof presented in Section 4 confirms these intuitions.

5 Conclusion

In this paper, we showed "robust undecidability" results for weighted timed games: optimal reachability problems remain undecidable under perturbation game semantics. Moreover, the problem even becomes undecidable for weighted timed automata in the excess perturbation game semantics. The undecidability in both cases is due to the ability of either of the players to play precisely, and the other one to check previous delays with precision. We conclude that game semantics does not introduce enough "fuzziness" in the semantics to avoid encoding undecidable languages.

We did not study the value upper-bound problems for the excess perturbation game semantics; we conjecture that it should be undecidable. We believe we could also recover decidability by restricting to closed guards, since then players would not be able to check the non-equality of the clock values.

References

1. 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)
2. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science 126(2), 183–235 (1994)
3. Alur, R., La Torre, S., Madhusudan, P.: Perturbed timed automata. In: Morari, M., Thiele, L. (eds.) HSCC 2005. LNCS, vol. 3414, pp. 70–85. Springer, Heidelberg (2005)
4. Alur, R., La Torre, S., Pappas, G.J.: Optimal paths in weighted timed automata. In: Di Benedetto, M.D., Sangiovanni-Vincentelli, A.L. (eds.) HSCC 2001. LNCS, vol. 2034, pp. 49–62. Springer, Heidelberg (2001)
5. Asarin, E., Bouajjani, A.: Perturbed Turing machines and hybrid systems. In: LICS 2001, pp. 269–278. IEEE Comp. Soc. Press (2001)
6. Asarin, E., Maler, O., Pnueli, A., Sifakis, J.: Controller synthesis for timed automata. In: SSC 1998, pp. 469–474. Elsevier Science (1998)
7. Behrmann, G., Fehnker, A., Hune, T., Larsen, K.G., Pettersson, P., Romijn, J., Vaandrager, F.: Minimum-cost reachability for priced timed automata. In: Di Benedetto, M.D., Sangiovanni-Vincentelli, A.L. (eds.) HSCC 2001. LNCS, vol. 2034, pp. 147–161. Springer, Heidelberg (2001)
8. Bouyer, P., Brihaye, T., Bruyère, V., Raskin, J.-F.: On the optimal reachability problem of weighted timed automata. Formal Methods in System Design 31(2), 135–175 (2007)
9. Bouyer, P., Brihaye, T., Markey, N.: Improved undecidability results on weighted timed automata. Information Processing Letters 98(5), 188–194 (2006)
10. 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)
11. Bouyer, P., Markey, N., Sankur, O.: Robust reachability in timed automata: A game-based approach. In: Czumaj, A., Mehlhorn, K., Pitts, A., Wattenhofer, R. (eds.) ICALP 2012, Part II. LNCS, vol. 7392, pp. 128–140. Springer, Heidelberg (2012)
12. Bouyer, P., Markey, N., Sankur, O.: Robust weighted timed automata and games. Research Report LSV-13-10, Laboratoire Spécification et Vérification, ENS Cachan, France, 25 p. (2013)
13. Brihaye, T., Bruyère, V., Raskin, J.-F.: On optimal timed strategies. In: Pettersson, P., Yi, W. (eds.) FORMATS 2005. LNCS, vol. 3829, pp. 49–64. Springer, Heidelberg (2005)
14. Chatterjee, K., Henzinger, T.A., Piterman, N.: Strategy logic. Information and Computation 208(6), 677–693 (2010)
15. Chatterjee, K., Henzinger, T.A., Prabhu, V.S.: Timed parity games: Complexity and robustness. Logicical Methods in Computer Science 7(4) (2011)
16. De Wulf, M., Doyen, L., Markey, N., Raskin, J.-F.: Robust safety of timed automata. Formal Methods in System Design 33(1-3), 45–84 (2008)
17. De Wulf, M., Doyen, L., Raskin, J.-F.: Almost ASAP semantics: From timed models to timed implementations. Formal Aspects of Computing 17(3), 319–341 (2005)
18. Fränzle, M.: Analysis of hybrid systems: An ounce of realism can save an infinity of states. In: Flum, J., Rodríguez-Artalejo, M. (eds.) CSL 1999. LNCS, vol. 1683, pp. 126–140. Springer, Heidelberg (1999)

19. Gupta, V., Henzinger, T.A., Jagadeesan, R.: Robust timed automata. In: Maler, O. (ed.) HART 1997. LNCS, vol. 1201, pp. 331–345. Springer, Heidelberg (1997)
20. Henzinger, T.A., Raskin, J.-F.: Robust undecidability of timed and hybrid systems. In: Lynch, N.A., Krogh, B.H. (eds.) HSCC 2000. LNCS, vol. 1790, pp. 145–159. Springer, Heidelberg (2000)
21. Puri, A.: Dynamical properties of timed systems. Discrete Event Dynamic Systems 10(1-2), 87–113 (2000)
22. Sankur, O., Bouyer, P., Markey, N.: Shrinking timed automata. In: FSTTCS 2011. LIPIcs, vol. 13, pp. 375–386. Leibniz-Zentrum für Informatik (2011)
23. Sankur, O., Bouyer, P., Markey, N., Reynier, P.-A.: Robust controller synthesis in timed automata. In: Aires, B., D'Argenio, P.R., Melgratti, H. (eds.) CONCUR 2013. LNCS, vol. 8052, pp. 546–560. Springer, Heidelberg (2013)

On MITL and Alternating Timed Automata[⋆]

Thomas Brihaye[1], Morgane Estiévenart[1,⋆⋆], and Gilles Geeraerts[2]

[1] Département de Mathématiques, Université de Mons (UMONS), Belgium
[2] Département d'Informatique, Université Libre de Bruxelles (U.L.B.), Belgium

Abstract. *One clock alternating timed automata* (OCATA) have been recently introduced as natural extension of (one clock) timed automata to express the semantics of MTL [12]. We consider the application of OCATA to problem of model-checking MITL formulas (a syntactic fragment of MTL) against timed automata. We introduce a new semantics for OCATA where, intuitively, clock valuations are *intervals* instead of *single values in* $\mathbb{R}$. Thanks to this new semantics, we show that we can *bound* the number of clock copies that are necessary to allow an OCATA to recognise the models of an MITL formula. Equipped with this technique, we propose a new algorithm to translate an MITL formula into a timed automaton, and we sketch several ideas to define new model checking algorithms for MITL.

1 Introduction

Automata-based model-checking [5,16] is a well-established technique for proving the correctness of computer systems. In this framework, the system to analyse is modeled by means of a *finite automaton* A whose accepted language consists of all the traces of the system. The *property* to prove is usually expressed using a *temporal logic formula* Φ, whose set of models is the language of all *correct executions*. For instance, the LTL formula $\Box(p \Rightarrow \Diamond q)$ says that every p-event should eventually be followed by a q-event. Then, establishing correctness of the system amounts to showing that the language $L(A)$ of the automaton is included in the language $[\![\Phi]\!]$ of the formula. In practice, automata-based model checking algorithms first negate the formula and translate $\neg\Phi$ into an automaton $A_{\neg\Phi}$ that recognises the *complement* of $[\![\Phi]\!]$, i.e., the set of all *erroneous traces*. Then, the algorithm computes the synchronous product $A \times A_{\neg\Phi}$ and check whether $L(A \times A_{\neg\Phi}) = \emptyset$, in which case the system respects the property.

While those techniques are now routinely used to prove the correctness of huge systems against complex properties [3], the model of finite automata and the classical temporal logics such as LTL are sometimes not *expressive* enough because they can model the *possible sequences of events*, but cannot express *quantitative properties about the (real) time elapsing between successive events*. To overcome these weaknesses, Alur and Dill [1] have proposed the model of *timed automata*, that extends finite automata

⋆ This work has been partly supported by a grant from the National Bank of Belgium, the ARC project (number AUWB-2010-10/15-UMONS-3), the FRFC project (number 2.4545.11) and a 'Crédit aux chercheurs' of the Belgian National Fund for Scientific Reserach FRS – F.N.R.S.
⋆⋆ This author has been supported by a FRIA scholarship.

with a finite set of (real valued) clocks. A real-time extension of LTL is the *Metric Temporal Logic* (MTL) that has been proposed by Koymans [9] and consists in labeling the modalities with time intervals. For instance $\Box(p \Rightarrow \Diamond_{[1,2]} q)$ means 'all p-event must be followed by a q-event that occurs *between* 1 *and* 2 *time units later*'. Unfortunately, the satisfiability and model-checking of MTL are undecidable on infinite words [8], and non-primitive recursive on finite words [13], with the pointwise semantics.

An interesting alternative is the Metric Interval Temporal Logic (MITL), that has been proposed by Alur *et al.* [2]. MITL is a syntactic fragment of MTL where singular intervals are disallowed on the modalities. Thanks to this restriction, MITL model-checking is EXPSPACE-c, even on infinite words. MITL thus seems a good compromise between *expressiveness* and *complexity*. In their seminal work, Alur *et al.* provide a construction to translate an MITL formula Φ into a timed automaton $\mathcal{B}_\Phi$, from which the automaton-based model checking procedure sketched above can be applied. Although this procedure is foundational from the theoretical point of view, it does not seem easily amenable to efficient implementation: the construction is quite involved, and requests that $\mathcal{B}_\Phi$ be completely built before the synchronous product with the system's model can be explored. Note that an alternative technique, based on the notion of *signal* has been proposed by Maler *et al.* [11]. However the semantics used there *slightly* differs from that of [2], whereas we stick to the classical pointwise semantics.

Since MITL is a *syntactic fragment* of MTL, all the techniques developed by Ouaknine and Worrell [12] for MTL can be applied to MITL. Their technique relies on the notion of *alternating timed automaton with one clock* (OCATA), an extension of timed automata. Intuitively an OCATA can create several copies of itself that run in parallel and must *all* accept the suffix of the word. For example, Fig. 1 displays an OCATA. Observe that the *arc* starting from ℓ_0 has two destinations: ℓ_0 and ℓ_1. When the automaton is in ℓ_0 with clock valuation v, and reads a σ, it spawns two copies of itself: the first reads the suffix of the word from (ℓ_0, v), and the latter from $(\ell_1, 0)$ (observe that the clock is reset on the branch to ℓ_1). Then, every MITL formula Φ can be translated into an OCATA $\mathcal{A}_\Phi$ that recognises its models [12]. The translation has the advantage of being very simple and elegant, and the size of $\mathcal{A}_\Phi$ is linear in the size of Φ. Unfortunately, one cannot bound a priori the number of clock copies that need to be remembered at all times along runs of an OCATA. Hence, OCATA cannot, in general, be translated to timed automata [10]. Moreover, the model-checking algorithm of [12] relies on well-quasi ordering to ensure termination, and has non-primitive recursive complexity.

In the present work, we consider only the point-wise semantics and exploit the translation of MITL formulas into OCATA [12] to devise new, optimal, – hopefully – elegant and simple algorithms to translate an MITL formula into a timed automaton. To achieve this, we rely on two technical ingredients. We first propose (in Section 3) a novel *interval-based* semantics for OCATA. In this semantics, clock valuations can be regarded as *intervals* instead of *single points*, thus our semantics generalises the standard one [12]. Intuitively, a state (ℓ, I) of an OCATA in the interval-based semantics (where ℓ is a location and I is an interval) can be regarded as an *abstraction* of all the (possibly unbounded) sets of states $\{(\ell, v_1), (\ell, v_2), \ldots, (\ell, v_n)\}$ of the standard semantics with $v_i \in I$ for all i. Then, we introduce a family of so-called *approximation function* that, roughly speaking, associate with each configuration C of the OCATA in the interval-based semantics, a set of configurations that are obtained from C by merging selected intervals in C. We rely on approximation functions to *bound* the number of clock copies that are present in all configurations. Our main technical contribution (Section 4) then consists in showing that, when considering an OCATA $\mathcal{A}_{\Phi}$ obtained from an MITL formula Φ, combining the interval semantics and a well-chosen approximation function is *sound*, in the sense that the resulting semantics recognises $L(\Phi)$, while requesting only *a bounded number of clock copies*. Thanks to this result, we provide an algorithm to translate $\mathcal{A}_{\Phi}$ into a plain timed automaton that accepts $L(\Phi)$.

Observe that the idea of *grouping clock values within intervals* to define the semantics of an MITL formula is not new. Because the syntax of MITL forbids punctual intervals, it is easy to observe that, if a formula of the form $\varphi_1 U_{[a,b]} \varphi_2$ holds at some instant t, it will also hold in other points in time, namely in some (non-singular) interval $[t', t' + b - a]$ that contains the instant t. This crucial observation is arguably the main idea that has lead to the definition of MITL, and is formalised in the seminal paper of Alur *et al.* [2], by the notion of *witnessing interval*, which is central to prove the correctness of the construction. The same observation has been relied upon in several other works on MITL or similar logics, such as the Event-Clock Logic (see for instance [15,14]). Our contribution is thus rather to show how this observation can be formalised in the context of OCATA, and to provide a systematic way to bound the number of clock copies necessary to recognise the models of an MITL formula with such an automaton.

From our point of view, the *benefits* of this new approach are as follows. From the *theoretical point of view*, our construction is the first that relies on OCATA to translate MITL formulas into timed automata. We believe our construction is easier to describe (and thus, hopefully, easier to implement) than the previous approaches. The translation from MITL to OCATA is very straightforward. The intuitions behind the translation of the OCATA into a timed automaton are also quite natural (although the proof of correctness requires some technicalities). From the *practical point of view*, our approach allows us, as we briefly sketch in Section 5, to envision *efficient model checking algorithms for MITL*, in the same spirit of the *antichain approach* [6] developed for LTL model checking. Note that the key ingredient to enable this *antichain* approach is the use of *alternating automata* to describe the LTL formula. Our contribution thus lay the *necessary theoretical basis* to enable a similar approach in a real-time setting.

Remark. Owing to lack of space, the reader is referred to the companion technical report [4] for the missing proofs and additional details.

2 Preliminaries

Basic notions. Let $\mathbb{R}$ ($\mathbb{R}^+$, $\mathbb{N}$) denote resp. the sets of real (non-negative real, natural) numbers. We call ***interval*** a convex subset of $\mathbb{R}$. We rely on the classical notation $\langle a, b \rangle$ for intervals, where $\langle$ is (or [, $\rangle$ is) or], $a \in \mathbb{R}$ and $b \in \mathbb{R} \cup \{+\infty\}$. For an interval $I = \langle a, b \rangle$, we let $\inf(I) = a$ be the *infimum* of I, $\sup(I) = b$ be its *supremum* (a and b are called the *endpoints* of I) and $|I| = \sup(I) - \inf(I)$ be its *length*. We note $\mathcal{I}(\mathbb{R})$ the set of all intervals. Similarly, we note $\mathcal{I}(\mathbb{R}^+)$ (resp. $\mathcal{I}(\mathbb{R}_{\mathbb{N}})$) the set of all intervals whose endpoints are in $\mathbb{R}^+$ (resp. in $\mathbb{N} \cup \{+\infty\}$). Let $I \in \mathcal{I}(\mathbb{R})$ and $t \in \mathbb{R}$, we note $I + t$ for $\{i + t \in \mathbb{R} \mid i \in I\}$. Let I and J be two intervals, we let $I < J$ iff

$\forall i \in I, \forall j \in J : i < j$. For $I \in \mathcal{I}(\mathbb{R})$, $v \in \mathbb{R}$ and $\bowtie \in \{<,>\}$, we note: $I \bowtie v$ iff $\forall i \in I, i \bowtie v$.

Let Σ be a finite alphabet. A *word* on a set S is a finite sequence $s = s_1 \ldots s_n$ of elements in S. We denote by $|s| = n$ the length of s. A time sequence $\bar{\tau} = \tau_1\tau_2\tau_3 \ldots \tau_n$ is a word on $\mathbb{R}^+$ s.t. $\forall i < |\bar{\tau}|, \tau_i \leq \tau_{i+1}$. A *timed word* over Σ is a pair $\theta = (\bar{\sigma}, \bar{\tau})$ where $\bar{\sigma}$ is a word over Σ, $\bar{\tau}$ a time sequence and $|\bar{\sigma}| = |\bar{\tau}|$. We also note θ as $(\sigma_1, \tau_1)(\sigma_2, \tau_2)(\sigma_3, \tau_3) \ldots (\sigma_n, \tau_n)$, and let $|\theta| = n$. A *timed language* is a (possibly infinite) set of timed words.

Metric Interval Temporal Logic. Given a finite alphabet Σ, the formulas of MITL are defined by the following grammar, where $\sigma \in \Sigma$, $I \in \mathcal{I}(\mathbb{R}_{\mathbb{N}})$:

$$\varphi := \top \quad | \quad \sigma \quad | \quad \varphi_1 \wedge \varphi_2 \quad | \quad \neg\varphi \quad | \quad \varphi_1 U_I \varphi_2.$$

We rely on the following usual shortcuts $\Diamond_I \varphi$ stands for $\top U_I \varphi$, $\Box_I \varphi$ for $\neg\Diamond_I \neg\varphi$, $\varphi_1 \tilde{U}_I \varphi_2$ for $\neg(\neg\varphi_1 U_I \neg\varphi_2)$, $\Box\varphi$ for $\Box_{[0,\infty)}\varphi$ and $\Diamond\varphi$ for $\Diamond_{[0,\infty)}\varphi$.

Given an MITL formula Φ, we note $Sub(\Phi)$ the set of all subformulas of Φ, i.e. : $Sub\,(\Phi) = \{\Phi\}$ when $\Phi \in \{\top\} \cup \Sigma$, $Sub\,(\neg\varphi) = \{\neg\varphi\} \cup Sub\,(\varphi)$ and $Sub\,(\Phi) = \{\Phi\} \cup Sub\,(\varphi_1) \cup Sub\,(\varphi_2)$ when $\Phi = \varphi_1 U_I \varphi_2$ or $\Phi = \varphi_1 \wedge \varphi_2$. We let $|\Phi|$ denote the *size of* Φ, defined as the *number* of U or $\tilde{U}$ modalities it contains.

Definition 1 (Semantics of MITL). *Given a timed word $\theta = (\bar{\sigma}, \bar{\tau})$ over Σ, a position $1 \leq i \leq |\theta|$ and an MITL formula Φ, we say that θ satisfies Φ from position i, written $(\theta, i) \models \Phi$ iff the following holds :*

- $(\theta, i) \models \top$
- $(\theta, i) \models \sigma \Leftrightarrow \sigma_i = \sigma$
- $(\theta, i) \models \varphi_1 \wedge \varphi_2 \Leftrightarrow (\theta, i) \models \varphi_1$ *and* $(\theta, i) \models \varphi_2$
- $(\theta, i) \models \neg\varphi \Leftrightarrow (\theta, i) \not\models \varphi$
- $(\theta, i) \models \varphi_1 U_I \varphi_2 \Leftrightarrow \exists i \leq j \leq |\theta|$, *such that* $(\theta, i) \models \varphi_2, \tau_j - \tau_i \in I$ *and* $\forall i \leq k < j, (\theta, k) \models \varphi_1$

*We say that θ **satisfies** Φ, written $\theta \models \Phi$, iff $(\theta, 1) \models \Phi$. We note $[\![\Phi]\!] = \{\theta \mid \theta \models \Phi\}$.*

Observe that, for all MITL formula Φ, $[\![\Phi]\!]$ is a timed language and that we can transform any MITL formula in an equivalent MITL formula in *negative normal form* (in which negation can only be present on letters $\sigma \in \Sigma$) using the operators : $\wedge, \vee, \neg, U_I$ and $\tilde{U}_I$.

Example 2. We can express the fact that '*every occurrence of p is followed by an occurrence of q between 2 and 3 time units later*' by: $\Box(p \Rightarrow \Diamond_{[2,3]} q)$. Its negation, $\neg\big(\Box(p \Rightarrow \Diamond_{[2,3]} q)\big)$, is equivalent to the following negative normal form formula: $\top U_{[0,+\infty)}(p \wedge \bot\, \tilde{U}_{[2,3]} \neg q)$.

Alternating timed automata. Let us now recall [13] the notion of (one clock) *alternating timed automaton* (OCATA for short). As we will see, OCATA define timed languages, and we will use them to express the semantics of MITL formula. Let $\Gamma(L)$ be a set of formulas defined by the following grammar:

$$\gamma := \top \quad | \quad \bot \quad | \quad \gamma_1 \vee \gamma_2 \quad | \quad \gamma_1 \wedge \gamma_2 \quad | \quad \ell \quad | \quad x \bowtie c \quad | \quad x.\gamma$$

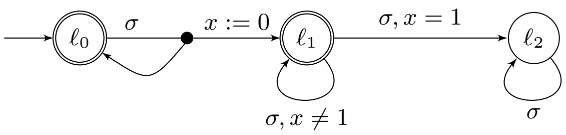


Fig. 1. OCATA $\mathcal{A}$

where $c \in \mathbb{N}$, $\bowtie \in \{<, \leq, >, \geq\}$ and $\ell \in L$. We call $x \bowtie c$ a *clock constraint*. Intuitively, the expression $x.\gamma$ means that clock x must be reset to 0.

Definition 3 ([13]). *A* ***one-clock alternating timed automaton*** *(OCATA) is a tuple $\mathcal{A} = (\Sigma, L, \ell_0, F, \delta)$ where Σ is a finite alphabet, L is a finite set of locations, ℓ_0 is the initial location, $F \subseteq L$ is a set of accepting locations, $\delta : L \times \Sigma \to \Gamma(L)$ is the transition function.*

We assume that, for all γ_1, γ_2 in $\Gamma(L)$: $x.(\gamma_1 \vee \gamma_2) = x.\gamma_1 \vee x.\gamma_2$, $x.(\gamma_1 \wedge \gamma_2) = x.\gamma_1 \wedge x.\gamma_2$, $x.x.\gamma = x.\gamma$, $x.(x \bowtie c) = 0 \bowtie c$, $x.\top = \top$ and $x.\ \bot{=}\bot$. Thus, we can write any formula of $\Gamma(L)$ in disjunctive normal form, and, from now on, we assume that $\delta(\ell, \sigma)$ is written in disjunctive normal form. That is, for all ℓ, σ, we have $\delta(\ell, \sigma) = \bigvee_j \bigwedge_k A_{j,k}$, where each term $A_{j,k}$ is of the form ℓ, $x.\ell$, $x \bowtie c$ or $0 \bowtie c$, with $\ell \in L$ and $c \in \mathbb{N}$. We call *arc* of the OCATA $\mathcal{A}$ a triple $(\ell, \sigma, \bigwedge_k A_{j,k})$ s.t. $\bigwedge_k A_{j,k}$ is a disjunct in $\delta(\ell, \sigma)$.

Example 4. As an example, consider the OCATA $\mathcal{A}$ in Fig. 1, over the alphabet $\Sigma = \{\sigma\}$. $\mathcal{A}$ has three *locations* ℓ_0, ℓ_1 and ℓ_2, such that ℓ_0 is initial and ℓ_0 and ℓ_1 are final. $\mathcal{A}$ has a unique clock x and its transition function is given by : $\delta(\ell_0, \sigma) = \ell_0 \wedge x.\ell_1$, $\delta(\ell_1, \sigma) = (\ell_2 \wedge x = 1) \vee (\ell_1 \wedge x \neq 1)$ and $\delta(\ell_2, \sigma) = \ell_2$. The *arcs* of $\mathcal{A}$ are thus $(\ell_0, \sigma, \ell_0 \wedge x.\ell_1)$, $(\ell_1, \sigma, \ell_2 \wedge x = 1)$, $(\ell_1, \sigma, \ell_1 \wedge x \neq 1)$ and (ℓ_2, σ, ℓ_2). Observe that, in the figure we represent the (conjunctive) arc $(\ell_0, \sigma, \ell_0 \wedge x.\ell_1)$ by an arrow splitting in two branches connected resp. to ℓ_0 and ℓ_1 (possibly with different resets: the reset of clock x is depicted by $x := 0$). Intuitively, *taking the arc* $(\ell_0, \sigma, \ell_0 \wedge x.\ell_1)$ means that, when reading a σ from location ℓ_0 and clock value v, the automaton should start *two copies of itself*, one in location ℓ_0, with clock value v, and a second in location ℓ_1 with clock value 0. Both copies should accept the suffix for the word to be accepted. This notion will be defined formally in the next section.

3 An Intervals Semantics for OCATA

The *standard* semantics for OCATA [12,10] is defined as an infinite transition system whose *configurations* are finite sets of pairs (ℓ, v), where ℓ is a location and v is the valuation of the (unique) clock. Intuitively, each configuration thus represents the current state of all the copies (of the unique clock) that run in parallel in the OCATA. The transition system is *infinite* because one cannot bound, a priori, the number of different clock valuations that can appear in a single configuration, thereby requiring peculiar techniques, such as well-quasi orderings (see [13]) to analyse it. In this section, we introduce a *novel* semantics for OCATA, in which *configurations* are sets of *states* (ℓ, I), where ℓ is a location of the OCATA and I is an *interval*, instead of a single point in $\mathbb{R}^+$. Intuitively, a state (ℓ, I) is an abstraction of all the states (ℓ, v) with $v \in I$, in the standard semantics. We further introduce the notion of *approximation function*. Roughly speaking, an approximation function associates with each configuration C (in the interval semantics), a set of configurations that *approximates* C (in a sense that will be made precise later), *and contains less states* than C. In section 4, we will show that the interval semantics, combined to a proper approximation function, allows us to build, from all MITL formula Φ, an OCATA $\mathcal{A}_\Phi$ accepting $[\![\Phi]\!]$, and whose *reachable configurations contain a bounded number of intervals*. This will be the basis of our algorithm to build a *timed automaton* recognising Φ (and hence performing automata-based model-checking of MITL).

We call *state* of an OCATA $\mathcal{A} = (\Sigma, L, \ell_0, F, \delta)$ a couple (ℓ, I) where $\ell \in L$ and $I \in \mathcal{I}(\mathbb{R}^+)$. We note $S = L \times \mathcal{I}(\mathbb{R}^+)$ the state space of $\mathcal{A}$. A state (ℓ, I) is *accepting* iff $\ell \in F$. When $I = [v, v]$ (sometimes denoted $I = \{v\}$), we shorten (ℓ, I) by (ℓ, v). A *configuration* of an OCATA $\mathcal{A}$ is a (possibly empty) finite set of states of $\mathcal{A}$ whose intervals associated with a same location are disjoint. In the rest of the paper, we sometimes see a configuration C as a function from L to $2^{\mathcal{I}(\mathbb{R}^+)}$ s.t. for all $\ell \in L$: $C(\ell) = \{I \mid (\ell, I) \in C\}$. We note $\mathrm{Config}(\mathcal{A})$ the set of all configurations of $\mathcal{A}$. The *initial configuration* of $\mathcal{A}$ is $\{(\ell_0, 0)\}$. A configuration is *accepting* iff all the states it contains are accepting (in particular, the empty configuration is accepting). For a configuration C and a delay $t \in \mathbb{R}^+$, we note $C+t$ the configuration $\{(\ell, I+t) | (\ell, I) \in C\}$. From now on, we assume that, for all configurations C and all locations ℓ: when writing $C(\ell)$ as $\{I_1, \ldots, I_m\}$ we have $I_i < I_{i+1}$ for all $1 \le i < m$. Let E be a finite set of intervals from $\mathcal{I}(\mathbb{R}^+)$. We let $\|E\| = |\{[a, a] \in E\}| + 2 \times |\{I \in E \mid \inf(I) \neq \sup(I)\}$ denote the *number of clock copies* of E. Intuitively, $\|E\|$ is the number of *individual clocks* we need to encode all the information present in E, using one clock to track singular intervals, and two clocks to retain $\inf(I)$ and $\sup(I)$ respectively for non-singular intervals I. For a configuration C, we let $\|C\| = \sum_{\ell \in L} \|C(\ell)\|$.

Interval semantics. Our definition of the *interval semantics* for OCATA follows the definition of the *standard* semantics as given by Ouaknine and Worrell [12], adapted to cope with intervals. Let $M \in \mathrm{Config}(\mathcal{A})$ be a configuration of an OCATA $\mathcal{A}$, and $I \in \mathcal{I}(\mathbb{R}^+)$. We define the satisfaction relation "$\models_I$" on $\Gamma(L)$ as:

$M \models_I \top$		$M \models_I \ell$	iff $(\ell, I) \in M$
$M \models_I \gamma_1 \wedge \gamma_2$	iff $M \models_I \gamma_1$ and $M \models_I \gamma_2$	$M \models_I x \bowtie c$	iff $\forall x \in I, x \bowtie c$
$M \models_I \gamma_1 \vee \gamma_2$	iff $M \models_I \gamma_1$ or $M \models_I \gamma_2$	$M \models_I x.\gamma$	iff $M \models_{[0,0]} \gamma$

We say that M is a *minimal model* of the formula $\gamma \in \Gamma(L)$ with respect to the interval $I \in \mathcal{I}(\mathbb{R}^+)$ iff $M \models_I \gamma$ and there is no $M' \subsetneq M$ such that $M' \models_I \gamma$. Remark that a formula γ can admit several minimal models (one for each disjunct in the case of a formula of the form $\gamma = \bigvee_j \bigwedge_k A_{j,k}$). Intuitively, for $\ell \in L, \sigma \in \Sigma$ and $I \in \mathcal{I}(\mathbb{R}^+)$, a minimal model of $\delta(\ell, \sigma)$ with respect to I represents a configuration the automaton can reach from state (ℓ, I) by reading σ. The definition of $M \models_I x \bowtie c$ only allows to take a transition $\delta(\ell, \sigma)$ from state (ℓ, I) if all the values in I satisfy the clock constraint $x \bowtie c$ of $\delta(\ell, \sigma)$.

Example 5. Let us consider again the OCATA of Fig. 1. A minimal model M of $\delta(\ell_1, \sigma)$ with respect to [1.5,2] must be such that : $M \models_{[1.5,2]} (\ell_1 \wedge x \neq 1) \vee (\ell_2 \wedge x = 1)$. As $\exists v \in [1.5, 2]$ s.t. $v \neq 1$, it is impossible that $M \models_{[1.5,2]} x = 1$. However, as $\forall v \in [1.5, 2], v \neq 1$, $M \models_{[1.5,2]} x \neq 1$ and so $M \models_{[1.5,2]} (\ell_1 \wedge x \neq 1) \vee (\ell_2 \wedge x = 1)$ iff $M \models_{[1.5,2]} \ell_1$, i.e. $(\ell_1, [1.5, 2]) \in M$. So, $\{(\ell_1, [1.5, 2])\}$ is the unique *minimal model* of $\delta(\ell_1, \sigma)$ wrt $[1.5, 2]$.

Approximation functions. As stated before, our goal is to define a semantics for OCATA that enables to bound the number of clock copies. To this end, we define the notion of approximation function: we will use such functions to reduce the number of clock copies associated with each location in a configuration. An approximation function associates with each configuration C a set of configurations C' s.t. $\|C'(\ell)\| \leq \|C(\ell)\|$ and s.t. the intervals in $C'(\ell)$, *cover* those of $C(\ell)$, for all ℓ. Then, we define the semantics of an OCATA $\mathcal{A}$ by means of a transition system $\mathcal{T}_{\mathcal{A},f}$ whose definition is *parametrised by an approximation function f.*

Definition 6. *Let $\mathcal{A}$ be an OCATA $\mathcal{A}$. An* approximation function *is a function f :* $\text{Config}(\mathcal{A}) \mapsto 2^{\text{Config}(\mathcal{A})}$ *s.t. for all configurations C, for all $C' \in f(C)$, for all locations $\ell \in L$: (i) $\|C'(\ell)\| \leq \|C(\ell)\|$, (ii) for all $I \in C(\ell)$, there exists $J \in C'(\ell)$ s.t. $I \subseteq J$, (iii) for all $J \in C'(\ell)$, there are $I_1, I_2 \in C(\ell)$ s.t.* $\inf(J) = \inf(I_1)$ *and* $\sup(J) = \sup(I_2)$. *We note $APP_{\mathcal{A}}$ the set of approximation functions for $\mathcal{A}$.*

Definition 7. *Let $\mathcal{A}$ be an OCATA and let $f \in APP_{\mathcal{A}}$ be an approximation function. The f*-semantics *of $\mathcal{A}$ is the transition system $\mathcal{T}_{\mathcal{A},f} = (\text{Config}(\mathcal{A}), \rightsquigarrow, \longrightarrow_f)$ on configurations of $\mathcal{A}$ defined as follows:*

- *the transition relation $\rightsquigarrow$ takes care of the elapsing of time : $\forall t \in \mathbb{R}^+, C \overset{t}{\rightsquigarrow} C'$ iff $C' = C + t$. We let $\rightsquigarrow = \bigcup_{t \in \mathbb{R}^+} \overset{t}{\rightsquigarrow}$.*
- *the transition relation $\longrightarrow$ takes care of discrete transitions between locations and of the approximation : $C = \{(\ell_k, I_k)_{k \in K}\} \xrightarrow{\sigma} C'$ iff there exists a configuration $C" = \bigcup_{k \in K} M_k$ s.t. (i) for all k: M_k is a minimal model of $\delta(\ell_k, \sigma)$ with respect to I_k, and (ii) $C' \in f(C")$. We let $\longrightarrow_f = \bigcup_{\sigma \in \Sigma} \xrightarrow{\sigma}_f$.*

We can now define the accepted language of an OCATA (parametrised by an approximation function f). Let $\theta = (\bar{\sigma}, \bar{\tau})$ be a timed word s.t. $|\theta| = n$, and let $f \in APP_{\mathcal{A}}$ be an approximation function. Let us note $t_i = \tau_i - \tau_{i-1}$ for all $1 \leq i \leq |\theta|$, assuming $\tau_0 = 0$. An *f-run* of $\mathcal{A}$ on θ is a finite sequence of discrete and continuous transitions in $\mathcal{T}_{\mathcal{A},f}$ that is labelled by θ, i.e. a sequence of the form: $C_0 \overset{t_1}{\rightsquigarrow} C_1 \xrightarrow{\sigma_1}_f C_2 \overset{t_2}{\rightsquigarrow} C_3 \xrightarrow{\sigma_2}_f \dots \overset{t_n}{\rightsquigarrow} C_{2n-1} \xrightarrow{\sigma_n}_f C_{2n}$. We say that an f-run is *accepting* iff its last configuration C_{2n} is accepting and we say that a timed word is f-accepted by $\mathcal{A}$ iff there exists an accepting f-run of $\mathcal{A}$ on this word. We note $L_f(\mathcal{A})$ the language of all finite timed words f-accepted by $\mathcal{A}$. In the rest of the paper, we (sometimes) use the abbreviation $C_i \xrightarrow{t,\sigma}_f C_{i+2}$ for $C_i \overset{t}{\rightsquigarrow} C_{i+1} = C_i + t \xrightarrow{\sigma}_f C_{i+2}$.

Observe that this interval semantics generalises the standard OCATA semantics [12]. This standard semantics can be recovered by considering $\mathcal{T}_{\mathcal{A},Id}$, where Id is the approximation function such that $Id(C) = \{C\}$ for all C. Indeed, in $\mathcal{T}_{\mathcal{A},Id}$, all the reachable configurations contain only states of the form $(\ell, [a, a])$, i.e., all intervals are singular. So, each state $(\ell, [a, a])$ can be naturally mapped to a state (ℓ, a) in the standard semantics. From now on, we denote $L_{Id}(\mathcal{A})$ by $L(\mathcal{A})$.

Example 8. Let us consider again the OCATA $\mathcal{A}$ in Fig. 1, and the timed word $\theta = (\sigma, 0)(\sigma, 0.2)(\sigma, 0.5)$, with $|\theta| = 3$. Let f be the approximation function s.t. for all $C \in \text{Config}(\mathcal{A})$: $f(C) = \big\{C(\ell_0) \cup C(\ell_2) \cup \{(\ell_1, [inf(I_1), sup(I_m)])\}\big\}$ if $C(\ell_1) = \{I_1, I_2, \dots I_m\} \neq \emptyset$ (assuming, as mentioned before, that $I_1 < I_2 < \cdots < I_m$); and $f(C) = \{C\}$ if $C(\ell_1) = \emptyset$. Thus, roughly speaking, $f(C)$ always contains one configuration, which is obtained from C by merging all the intervals in $C(\ell_1)$ and keeping the rest of the configuration untouched. Then, an f-run on θ is: $\rho_1 = \{(\ell_0, 0)\} \xrightarrow{0,\sigma} \{(\ell_0, 0), (\ell_1, 0)\} \xrightarrow{0.2,\sigma} \{(\ell_0, 0.2), (\ell_1, [0, 0.2])\} \xrightarrow{0.3,\sigma} \{(\ell_0, 0.5), (\ell_1, [0, 0.5])\}$. Also, an Id-run on θ is: $\rho_2 = \{(\ell_0, 0)\} \xrightarrow{0,\sigma} \{(\ell_0, 0), (\ell_1, 0)\} \xrightarrow{0.2,\sigma} \{(\ell_0, 0.2), (\ell_1, 0), (\ell_1, 0.2)\} \xrightarrow{0.3,\sigma} \{(\ell_0, 0.5), (\ell_1, 0), (\ell_1, 0.3), (\ell_1, 0.5)\}$. Now, consider the timed word $\theta' = \theta(\sigma, 1.1)$. An Id-run on θ' is $\rho_3 = \rho_2 \xrightarrow{0.6,\sigma} \{(\ell_0, 1.1), (\ell_1, 0), (\ell_1, 0.6), (\ell_1, 0.9), (\ell_1, 1.1)\}$ (hence θ' is Id-accepted by $\mathcal{A}$), but $\mathcal{A}$ has no f-run on θ'. Indeed, letting 0.6 t.u. elapse from ρ_1's last configuration yields $\{(\ell_0, 1.1), (\ell_1, [0.6, 1.1])\}$ from which no transition can be fired, because $[0.6, 1.1]$ satisfies neither $x \neq 1$ nor $x = 1$, which are the respective guards of the arcs from ℓ_1.

In the rest of the paper we will rely mainly on approximation functions that enable to bound the number of clock copies in all configurations along all runs of an OCATA $\mathcal{A}$. Let $k \in \mathbb{N}$ be a constant. We say that $f_k \in APP_{\mathcal{A}}$ is a *k-bounded approximation function* iff for all $C \in \text{Config}(\mathcal{A})$, for all $C' \in f_k(C)$: $\|C'\| \leq k$.

Accepted language and approximations. Let us now study the relationship between the standard semantics of OCATA and the family of semantics obtained when relying on an approximation function that is different from Id. We show that introducing approximations does not increase the accepted language:

Proposition 9. *For all OCATA $\mathcal{A}$, for all $f \in APP_{\mathcal{A}}$: $L_f(\mathcal{A}) \subseteq L(\mathcal{A})$.*

Proof (sketch). Let $C_0 \overset{t_1}{\rightsquigarrow} C_1 \xrightarrow{\sigma_1}_f C_2 \overset{t_2}{\rightsquigarrow} C_3 \cdots \xrightarrow{\sigma_n}_f C_{2n}$ be an *accepting* f-run of $\mathcal{A}$ on θ, and let us build, inductively, an accepting Id-run $D_0 \overset{t_1}{\rightsquigarrow} D_1 \xrightarrow{\sigma_1}_{Id} D_2 \overset{t_2}{\rightsquigarrow} D_3 \cdots \xrightarrow{\sigma_n}_{Id} D_{2n}$ on θ s.t. the following *invariant* holds: for all $0 \leq i \leq 2n$, for all $(\ell, [v, v]) \in D_i$, there is $(\ell, I) \in C_i$ s.t. $v \in I$. The *base case* is trivial since $C_0 = D_0$. For the *inductive case*, we first observe that the elapsing of time maintains the invariant. Thus, we have to show that each discrete step in the f-run can be simulated by a discrete step in the Id-run that maintains the invariant. A σ labeled discrete step from some configuration C_{2j+1} in the f-run consists in selecting an arc a_s of the form (ℓ, σ, γ) for each $s = (\ell, I)$ in C_{2j+1}, whose guard is satisfied by I. Then, firing all these arcs yields a configuration E, and $C_{2j+2} \in f(E)$. From each $s' = (\ell, [v, v])$ in D_{2j+1}, we fire the arc a_s where $s = (\ell, I)$ is a state in C_{2j+1} s.t $v \in I$. Such an s exists by induction hypothesis. Since the effects of the arcs are the same, and by properties of the approximation function, we conclude that D_{2j+2} and C_{2j+2} respect the invariant. In particular D_{2n} and C_{2n} respect it, hence D_{2n} is accepting. □

4 From MITL to Timed Automata

In this section, we present our new technique to build, from any MITL formula Φ, a *timed automaton* that accepts $[\![\Phi]\!]$. Our technique relies on two ingredients. First, we recall [13] how to build, from all MITL formula Φ, and OCATA $\mathcal{A}_\Phi$ s.t. $L(\mathcal{A}_\Phi) = [\![\Phi]\!]$. This is not sufficient to obtain a timed automaton, as, in general, the semantics of an OCATA needs an *unbounded* number of clock copies, which prevents us from translating all OCATA into timed automata. The second ingredient is the definition of a family of *bounded approximation functions* $f^\star_\Phi$, s.t., for all MITL formula Φ, $L_{f^\star_\Phi}(\mathcal{A}_\Phi) = L(\mathcal{A}_\Phi)$. Since each $f^\star_\Phi$ is a *bounded approximation function*, the number of clock copies in the $f^\star_\Phi$-semantics of $\mathcal{A}_\Phi$ is *bounded*, which allows us to build a *timed automaton* $\mathcal{B}_\Phi$ *with the same semantics* (thus, $\mathcal{B}_\Phi$ accepts $[\![\Phi]\!]$).

From MITL to OCATA. We begin by recalling[1] [13] how to build, from any MITL formula Φ (in negative normal form), an OCATA $\mathcal{A}_\Phi$ s.t. $L(\mathcal{A}_\Phi) = [\![\Phi]\!]$. We let $\mathcal{A}_\Phi = (\Sigma, L, \ell_0, F, \delta)$ where: L is the set containing the initial copy of Φ, noted 'Φ_{init}', and all the formulas of $Sub(\Phi)$ whose outermost connective is 'U' or '$\tilde{U}$'; $\ell_0 = \Phi_{init}$; F is the set of the elements of L of the form $\varphi_1 \tilde{U}_I \varphi_2$. Finally δ is defined[2] by induction on the structure of Φ:

- $\delta(\Phi_{init}, \sigma) = x.\delta(\Phi, \sigma)$
- $\delta(\varphi_1 \vee \varphi_2, \sigma) = \delta(\varphi_1, \sigma) \vee \delta(\varphi_2, \sigma)$; $\delta(\varphi_1 \wedge \varphi_2, \sigma) = \delta(\varphi_1, \sigma) \wedge \delta(\varphi_2, \sigma)$
- $\delta(\varphi_1 U_I \varphi_2, \sigma) = (x.\delta(\varphi_2, \sigma) \wedge x \in I) \vee (x.\delta(\varphi_1, \sigma) \wedge \varphi_1 U_I \varphi_2 \wedge x \leq sup(I))$
- $\delta(\varphi_1 \tilde{U}_I \varphi_2, \sigma) = (x.\delta(\varphi_2, \sigma) \vee x \notin I) \wedge (x.\delta(\varphi_1, \sigma) \vee \varphi_1 \tilde{U}_I \varphi_2 \vee x > sup(I))$
- $\forall \sigma_1, \sigma_2 \in \Sigma$: $\delta(\sigma_1, \sigma_2) = \begin{cases} \text{true} & \text{if } \sigma_1 = \sigma_2 \\ \text{false} & \text{if } \sigma_1 \neq \sigma_2 \end{cases}$ and $\delta(\neg\sigma_1, \sigma_2) = \begin{cases} \text{false} & \text{if } \sigma_1 = \sigma_2 \\ \text{true} & \text{if } \sigma_1 \neq \sigma_2 \end{cases}$
- $\forall \sigma \in \Sigma$: $\delta(\top, \sigma) = \top$ and $\delta(\bot, \sigma) = \bot$.

To simplify the following proofs, we deviate slightly from that definition, and assume that if a formula of type $\varphi_1 U_I \varphi_2$ or $\varphi_1 \tilde{U}_I \varphi_2$ appears more than once as a sub-formula of Φ, the occurrences of this formula are supposed different and are encoded as different locations. With this definition, we have:

Theorem 10 ([13]). *For all MITL formula Φ: $L(\mathcal{A}_\Phi) = [\![\Phi]\!]$.*

[1] Remark that in [13], the authors are concerned with MTL, but since MITL is a syntactic fragment of MTL, the procedure applies here.

[2] Remark that the $x \leq sup(I)$ and $x > sup(I)$ conditions in the resp. definitions of $\delta(\varphi_1 U_I \varphi_2, \sigma)$ and $\delta(\varphi_1 \tilde{U}_I \varphi_2, \sigma)$ have been added here for technical reasons. This does not modify the accepted language. Indeed, in [13], these conditions are given in the infinite word semantics of OCATA.

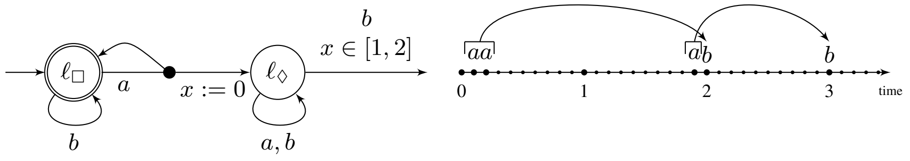
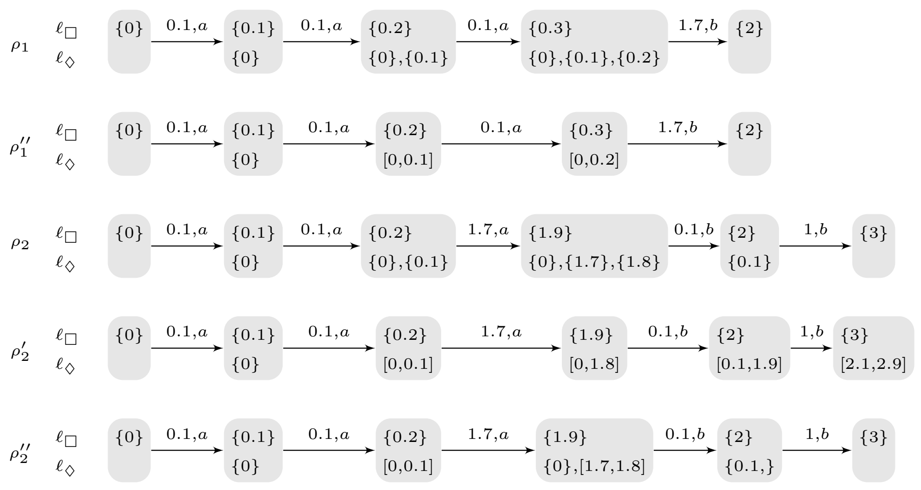


Fig. 2. (left) OCATA $\mathcal{A}_{\Phi_1}$ with $\Phi_1 = \Box(a \Rightarrow \Diamond_{[1,2]}b)$. (right) The grouping of clocks.

Example 11. As an example consider the formula $\Phi_1 = \Box(a \Rightarrow \Diamond_{[1,2]}b)$, which is a shorthand for $\bot\tilde{U}_{[0,+\infty)}\big(a \Rightarrow (\top U_{[1,2]}b)\big)$. The OCATA $\mathcal{A}_{\Phi_1}$ is given in Fig. 2 (left), where the location $\ell_\Box$ corresponds to Φ_1 and the location $\ell_\Diamond$ corresponds to $\top U_{[1,2]}b$. One can check that this automaton follows strictly the above definition, after simplification of the formulas, *except* that we have remove the Φ_{init} location and used the $\ell_\Box$ location as initial location instead, to enhance readability of the example (this does not modify the accepted language, in the present case). Observe the edge labeled by $b, x \in [1, 2]$ from $\ell_\Diamond$, without target state: it depicts the fact that: $\delta(\top U_{[1,2]}b, b) = (x \in [1, 2]) \vee (\top U_{[1,2]}b)$. Intuitively, when the automaton has a copy in location '$\Diamond$' with a clock valuation in $[1, 2]$, the copy can be *removed*, because a minimal model of $x \in [1, 2]$ wrt to a valuation v with $v \in [1, 2]$ is $\emptyset$.

To help us build an intuition of the $f^\star_{\Phi_1}$ function, let us consider the Id-run ρ_1 of $\mathcal{A}_{\Phi_1}$ on $\theta_1 = (a, 0.1)(a, 0.2)(a, 0.3)(b, 2)$ depicted in Fig. 3. Observe that $\theta_1 \models \Phi_1$, and that ρ_1's last configuration is indeed accepting. Also note that, as in the example of Fig. 1, the number of clock copies necessary in the Id-semantics cannot be bounded. Now, let us discuss the intuition behind $f^\star_{\Phi_1}$ by considering θ_1 again. Consider ρ'_1 the run prefix of ρ_1 ending in $\{(\ell_\Box, 0.2), (\ell_\Diamond, 0), (\ell_\Diamond, 0.1)\}$. Clearly, the last configuration of ρ'_1 can be *over-approximated* by *grouping* the two clock values 0 and 0.1 into the *smallest interval that contains them both*, i.e. $[0, 0.1]$. This intuitions is compatible with the definition of *bounded approximation function*, and yields the *accepting* run ρ''_1 depicted in Fig. 3. Nevertheless, we must be be careful when grouping clock copies. Let us consider $\theta_2 = (a, 0.1)(a, 0.2)(a, 1.9)(b, 2)(b, 3) \in [\![\Phi_1]\!]$, as witnessed by ρ_2 depicted in Fig. 3. When grouping *in the same interval*, the three clock copies created in $\ell_\Diamond$ (along ρ_2) by the reading of the three a's (and letting further 0.1 time unit elapse) yields the run prefix of ρ'_2 depicted in Fig. 3 ending in $\{(\ell_\Box, 2), (\ell_\Diamond, [0.1, 1.9])\}$. From the last configuration of this run, the edge with guard $x \in [1, 2]$ and origin $\ell_\Diamond$ cannot be taken. Thus, the only way to extend this prefix is through ρ'_2 (depicted in Fig. 3) which yields a run that *does not accept* θ_2. Obviously, by grouping the two clock copies created in $\ell_\Diamond$ by the two first a's, and by keeping the third one apart, one obtains the accepting run ρ''_2 (depicted in Fig. 3). Fig. 2 (right) shows the intuition behind the grouping of clocks. The two first positions (with $\sigma_1 = \sigma_2 = a$) of the word satisfy Φ_1, *because of the b in position* 4 (with $\tau_4 = 2$), while position 3 (with $\sigma_3 = a$) satisfies Φ_1 *because of the b in position* 5 (with $\tau_5 = 3$). This explains why we group the two first copies (corresponding to the two first a's) and keep the third one apart.



ρ1 ℓ□ ℓ◇ {0} 0.1,a {0.1} {0} 0.1,a {0.2} {0},{0.1} 0.1,a {0.3} {0},{0.1},{0.2} 1.7,b {2}
ρ1'' ℓ□ ℓ◇ {0} 0.1,a {0.1} {0} 0.1,a {0.2} [0,0.1] 0.1,a {0.3} [0,0.2] 1.7,b {2}
ρ2 ℓ□ ℓ◇ {0} 0.1,a {0.1} {0} 0.1,a {0.2} {0},{0.1} 1.7,a {1.9} {0},{1.7},{1.8} 0.1,b {2} {0.1} 1,b {3}
ρ2' ℓ□ ℓ◇ {0} 0.1,a {0.1} {0} 0.1,a {0.2} [0,0.1] 1.7,a {1.9} [0,1.8] 0.1,b {2} [0.1,1.9] 1,b {3} [2.1,2.9]
ρ2'' ℓ□ ℓ◇ {0} 0.1,a {0.1} {0} 0.1,a {0.2} [0,0.1] 1.7,a {1.9} {0},[1.7,1.8] 0.1,b {2} {0.1,} 1,b {3}

Fig. 3. Several OCATA runs

The approximation functions $f^\star_\Phi$. Let us now formally define the family of *bounded* approximation functions that will form the basis of our translation to timed automata. We first give an *upper bound* $M(\Phi)$ on the number of clock copies (intervals) we need to consider in the configurations to recognise an MITL formula Φ. The precise definition of the bound $M(\Phi)$ is technical and is given by induction on the structure of the formula. It can be found in the companion technical report [4]. Yet, for all MITL formula Φ:

$$M(\Phi) \leq |\Phi| \times \max_{I \in \mathcal{I}_\Phi} \left(4 \times \left\{ \left\lceil \frac{\inf(I)}{|I|} \right\rceil \right\} + 2, 2 \times \left\{ \left\lceil \frac{\sup(I)}{|I|} \right\rceil \right\} + 2 \right)$$

where $\mathcal{I}_\Phi$ is the set of all the intervals that occur in Φ[3].

Equipped with this bound, we can define the $f^\star_\Phi$ function. Throughout this description, we assume an OCATA $\mathcal{A}$ with set of locations L. Let $S = \{(\ell, I_0), (\ell, I_1), \ldots, (\ell, I_m)\}$ be a set of states of $\mathcal{A}$, all in the same location ℓ, with, as usual $I_0 < I_1 < \cdots < I_m$. Then, we let $\mathsf{Merge}\,(S) = \{(\ell, [0, sup(I_1)]), (\ell, I_2), \ldots, (\ell, I_m)\}$ if $I_0 = [0,0]$ and $\mathsf{Merge}\,(S) = S$ otherwise, i.e., $\mathsf{Merge}\,(S)$ is obtained from S by *grouping* I_0 and I_1 iff $I_0 = [0,0]$, otherwise $\mathsf{Merge}\,(S)$ does not modify S. Observe that, in the former case, if I_1 is not a singleton, then $\|\mathsf{Merge}\,(S)\| = \|S\| - 1$. Now, we can lift the definition of Merge to configurations. Let C be a configuration of $\mathcal{A}$ and let $k \in \mathbb{N}$. We let:

$$\mathsf{Merge}\,(C, k) = \left\{C' \mid \|C'\| \leq k \text{ and } \forall \ell \in L : C'(\ell) \in \{\mathsf{Merge}\,(C(\ell))\,, C(\ell)\}\right\}$$

Observe that $\mathsf{Merge}\,(C, k)$ is a (possibly empty) *set of configurations*, where each configuration (i) has at most k clock copies, and (ii) can be obtained by applying (if possible) or not the Merge function to each $C(\ell)$. Let us now define a family of k-bounded

[3] The first component of the maximum comes from U and the second from $\tilde{U}$.

approximation functions, based on Merge. Let $k \geq 2 \times |L|$ be a bound and let C be a configuration, assuming that $C(\ell) = \{I_1^\ell, \dots, I_{m_\ell}^\ell\}$ for all $\ell \in L$. Then:

$$F^k(C) = \begin{cases} \mathsf{Merge}\,(C, k) & \text{If } \mathsf{Merge}\,(C, k) \neq \emptyset \\ \{(\ell, [inf(I_1^\ell), sup(I_{m_\ell}^\ell)]) \mid \ell \in L\} & \text{otherwise} \end{cases}$$

Roughly speaking, the $F^k(C)$ function tries to obtain configurations C' that approximate C and s.t. $\|C'\| \leq k$, using the Merge function. If it fails to, i.e., when $\mathsf{Merge}\,(C, k) = \emptyset$, $F^k(C)$ returns a single configuration, obtained from C by grouping all the intervals in each location. The latter case occurs in the definition of F^k for the sake of completeness. When the OCATA $\mathcal{A}$ has been obtained from an MITL formula Φ, and for k big enough (see hereunder) each $\theta \in [\![\Phi]\!]$ will be recognised by at least one F^k-run of $\mathcal{A}$ that traverses only configurations obtained thanks to Merge. We can now finally define $f_\Phi^\star$ for all MITL formula Φ, by letting $f_\Phi^\star = F^K$, where $K = \max\{2 \times |L|, M(\Phi)\}$. It is easy to observe that $f_\Phi^\star$ is indeed a *bounded approximation function*. Then, we can show that, for all MITL formula Φ, the $f_\Phi^\star$-semantics of $\mathcal{A}_\Phi$ accepts exactly $[\![\Phi]\!]$. To obtain this result, we rely on the following proposition[4]:

Proposition 12. *Let Φ be an MITL formula, let K be a set of index and, $\forall k \in K$, let $\Phi_k = \varphi_{1,k} U_{I_k} \varphi_{2,k}$ be subformulas of Φ. For all $k \in K$, let ℓ_{Φ_k} be their associated locations in $\mathcal{A}_\Phi$. Let $\theta = (\bar{\sigma}, \bar{\tau})$ be a timed word and let $J_k \in \mathcal{I}(\mathbb{R}^+)$ be closed intervals. The automaton $\mathcal{A}_\Phi$ Id-accepts θ from configuration $\{(\ell_{\Phi_k}, J_k)_{k \in K}\}$ iff $\forall k \in K, \exists m_k \geq 1 : (\theta, m_k) \models \varphi_2 \wedge \tau_{m_k} \in I_k - \inf(J_k) \wedge \tau_{m_k} \in I_k - \sup(J_k) \wedge \forall 1 \leq m'_k < m_k : (\theta, m'_k) \models \varphi_1$.*

To illustrate this proposition, let us consider $\Phi_2 \equiv \top U_{[2,3]} b$, the associated automaton $\mathcal{A}_{\Phi_2}$ and the timed word $\theta = (a, 0)(b, 1)(b, 2)$. Assume that $\mathcal{A}_{\Phi_2}$ is in configuration $C = \{(\ell_{\Phi_2}, [0, 2])\}$ and we must read θ from C. Observe that for all value $y \in [0, 2]$, there is a position m in θ s.t. $\tau_m \in [2, 3] - y$ (i.e., τ_m satisfies the temporal constraint of the modality), $(\theta, m) \models b$ and all intermediate positions satisfy $\top$: $\forall 1 \leq m' < m, (\theta, m') \models \top$. In other words, $\forall y \in [0, 2], (\theta, 1) \models \top U_{[2,3]-y} b$. Yet, the conditions of the propositions are not satisfied and, indeed, there is no accepting Id-run of $\mathcal{A}_{\Phi_2}$ from C. Indeed, after reading the first (resp. second) b, the resulting configuration contains $(\ell_{\Phi_2}, [1, 3])$ (resp. $(\ell_{\Phi_2}, [2, 4])$). In both cases, the interval associated with ℓ_{Φ_2} does not satisfy the clock constraint $x \in [2, 3]$ of the transition $x.\delta(\top, b) \wedge x \in [2, 3]$ of $\mathcal{A}_{\Phi_2}$ that enables to leave location ℓ_{Φ_2}. This example also shows that Proposition 12 cannot be obtained as a corollary of the results by Ouaknine and Worrell [13] and deserves a dedicated proof as part of our contribution.

The property given by Proposition 12 is thus crucial to determine, given an accepting run, whether we can *group* several intervals and retain an accepting run or not. This observation will be central to the proof of our main theorem:

Theorem 13. *For all MITL formula Φ, $f_\Phi^\star$ is a* bounded approximation function *and $L_{f_\Phi^\star}(\mathcal{A}_\Phi) = L(\mathcal{A}_\Phi) = [\![\Phi]\!]$.*

Proof (sketch). By definition, $f_\Phi^\star$ is a *bounded approximation function*. Hence, by Proposition 9, $L_{f_\Phi^\star}(\mathcal{A}_\Phi) \subseteq L(\mathcal{A}_\Phi)$. Let $\theta = (\bar{\sigma}, \bar{\tau})$ be a timed word in $L(\mathcal{A}_\Phi)$, and let us show

[4] Stated here for the U modality, a similar proposition holds for $\tilde{U}$.

that $\theta \in L_{f^\star_\Phi}(\mathcal{A}_\Phi)$, by building an accepting $f^\star_\Phi$-run ρ' on θ. Since, $\theta \in L(A_\Phi)$, there is an accepting Id-run ρ of $\mathcal{A}_\Phi$ on θ. We assume that $\rho = C_0 \xrightarrow{\tau_1,\sigma_1}_{Id} C_1 \xrightarrow{\tau_2-\tau_1,\sigma_2}_{Id} C_2 \cdots \xrightarrow{\tau_n-\tau_{n-1},\sigma_n} C_n$.

We build, by induction on the length of ρ, a sequence of runs $\rho_0, \rho_1, \ldots, \rho_n$ s.t. for all $0 \leq j \leq n$, $\rho_j = D^j_0 \xrightarrow{\tau_1,\sigma_1}_{Id} \cdots \xrightarrow{\tau_n-\tau_{n-1},\sigma_n} D^j_n$ is an accepting run on θ with the following properties: (i) for all $0 \leq k \leq j$: $\left\|D^j_k\right\| \leq 4 \times \left\lceil \frac{\inf(I_i)}{|I_i|} \right\rceil + 2$, and (ii) assuming $\tau_0 = 0$: $D^j_j \xrightarrow{\tau_{j+1}-\tau_j,\sigma_{j+1}}_{Id} \cdots \xrightarrow{\tau_n-\tau_{n-1},\sigma_n}_{Id} D^j_n$ is an accepting Id-run on $\theta^{j+1} = (\sigma_{j+1}\sigma_{j+2}\ldots\sigma_n, \tau')$, where $\tau' = (\tau_{j+1}-\tau_j)(\tau_{j+2}-\tau_j)\ldots(\tau_n-\tau_j)$, assuming $\tau_{-1} = 0$, i.e., θ^j is the suffix of length $n-j$ of θ, where all the timed stamps have been decreased by τ_j. Clearly, letting $\rho_0 = \rho$ satisfied these properties. We build ρ_{k+1} from ρ_k, by first letting $D^{k+1}_0, D^{k+1}_1, \ldots, D^{k+1}_k = D^k_0, D^k_1, \ldots, D^k_k$, and then showing how to build D^{k+1}_{k+1} from D^k_{k+1} by merging intervals. Let $\ell \in L$. We use the criterion given by Proposition 12 to decide when to group intervals in $D^k_{k+1}(\ell)$. Assume $D^k_{k+1}(\ell) = \{J_1, J_2, \ldots, J_m\}$. Then:

- If $D^k_{k+1}(\ell)$ is either empty, or a singleton, we let $D^{k+1}_{k+1}(\ell) = D^k_{k+1}(\ell)$.
- Else, if $J_1 \neq [0,0]$, then the reading of σ_{k+1} has not created a new copy in ℓ and we let $D^{k+1}_{k+1}(\ell) = D^k_{k+1}(\ell)$ too.
- Else, $J_1 = [0,0]$ and we must decide whether we group this clock copy with J_2 or not. Assume[5] ℓ corresponds to the sub-formula $\varphi_1 U_I \varphi_2$. Then:
  1. if $\exists m \geq 1$ such that : $(\theta^{k+1}, \tau^{k+1}_m) \models \varphi_2 \wedge \tau^{k+1}_m \in I - \sup(J_2) \wedge \tau^{k+1}_m \in I \wedge \forall 1 \leq m' < m : (\theta^{k+1}, \tau^{k+1}_{m'}) \models \varphi_1$, then, we let $D^{k+1}_{k+1}(\ell) = \mathsf{Merge}\left(D^k_{k+1}(\ell)\right)$,
  2. else, we let $D^{k+1}_{k+1}(\ell) = D^k_{k+1}(\ell)$.

We finish the construction of ρ_{k+1} by firing, from D^{k+1}_{k+1} the same arcs as in the D^k_{k+1} $D^k_{k+2} \ldots D^k_n$ suffix of ρ_k, using the Id-semantics. Proposition 12 guarantees that we have grouped the intervals in such a way that this suffix is an Id-accepting run on θ^{k+1}. Finally, we let $\rho' = \rho_n$ which is an accepting run on θ. We finish the proof by a technical discussion showing that ρ_n is an $f^\star_\Phi$-run. $\square$

From OCATA to timed automata. Let us show how we can now translate $\mathcal{A}_\Phi$ into a *timed automaton* that accepts $[\![\Phi]\!]$. The crucial point is to define a *bound*, $M(\Phi)$, on the number of clocks that are necessary to recognise models of Φ.

A timed automaton (TA) is a tuple $\mathcal{B} = (\Sigma, L, \ell_0, X, F, \delta)$, where Σ is a finite *alphabet*, L is finite set of *locations*, $\ell_0 \in L$ is the *initial location*, X is a finite set of *clocks*, $F \subseteq L$ is the set of *accepting locations*, and $\delta \subseteq L \times \Sigma \times \mathcal{G}(X) \times 2^X \times L$ is a finite set of *transitions*, where $\mathcal{G}(X)$ denotes the set of *guards on* X, i.e. the set of all finite conjunctions of *clock constraints* on clocks from X. For a transition $(\ell, \sigma, g, r, \ell')$, we say that g is its *guard*, and r its *reset*. A *configuration* of a TA is a pair (ℓ, v), where $v : X \mapsto \mathbb{R}^+$ is a *valuation* of the clocks in X. We denote by Config $(\mathcal{B})$ the set of all configurations of $\mathcal{B}$, and we say that (ℓ, v) is *accepting* iff $\ell \in F$. For all $t \in \mathbb{R}^+$, we

[5] When ℓ corresponds to the sub-formula $\varphi_1 \tilde{U}_I \varphi_2$, we use the proposition similar to Proposition 12 for $\tilde{U}$ to decide whether we group J_1 with J_2 or not.

have (time successor) $(\ell, v) \overset{t}{\rightsquigarrow} (\ell', v')$ iff $\ell = \ell'$ and $v' = v + t$ where $v + t$ is the valuation s.t. for all $x \in X$: $(v + t)(x) = v(x) + t$. For all $\sigma \in \Sigma$, we have (discrete successor) $(\ell, v) \xrightarrow{\sigma} (\ell', v')$ iff there is $(\ell, \sigma, g, r, \ell') \in \delta$ s.t. $v \models g$, for all $x \in r$: $v'(x) = 0$ and for all $x \in X \setminus r$: $v'(x) = v(x)$. We write $(\ell, v) \xrightarrow{t,\sigma} (\ell', v')$ iff there is $(\ell'', v'') \in \text{Config}\,(\mathcal{B})$ s.t. $(\ell, v) \overset{t}{\rightsquigarrow} (\ell'', v'') \xrightarrow{\sigma} (\ell', v')$. A timed word $\theta = (\bar{\sigma}, \bar{\tau})$ with $\bar{\sigma} = \sigma_1\sigma_2 \cdots \sigma_n$ and $\bar{\tau} = \tau_1\tau_2 \cdots \tau_n$ is *accepted* by $\mathcal{B}$ iff there is an *accepting run* of $\mathcal{B}$ on θ, i.e. a sequence of configurations (ℓ_1, v_1),..., (ℓ_n, v_n) s.t. $\ell_n \in F$ and for all $0 \leq i \leq n - 1$: $(\ell_i, v_i) \xrightarrow{\tau_i - \tau_{i-1}, \sigma_i} (\ell_{i+1}, v_{i+1})$, where v_0 assigns 0 to all clocks, and assuming that τ_{-1} denotes 0. We denote by $L(\mathcal{B})$ the *language* of $\mathcal{B}$ (set of words accepted by $\mathcal{B}$).

We can now sketch the translation (see [4] for the details). Let Φ be an MITL formula, and assume $\mathcal{A}_\Phi = (\Sigma, L^\Phi, \ell_0^\Phi, F^\Phi, \delta^\Phi)$. Let us show how to build the TA $\mathcal{B}_\Phi = (\Sigma, L, \ell_0, X, F, \delta)$ s.t. $L(\mathcal{B}_\Phi) = L_{f_\Phi^\star}(\mathcal{A}_\Phi)$. The TA $\mathcal{B}_\Phi$ is built as follows. For a set of clocks X, we let $\mathsf{loc}(X)$ be the set of functions S that associate with each $\ell \in L^\Phi$ a finite sequence $(x_1, y_1), \ldots, (x_n, y_n)$ of pairs of clocks from X, s.t. each clock occurs only once in all the $S(\ell)$. Then, $L = \mathsf{loc}(X)$. Observe that L is indeed a finite set. Intuitively, a configuration (S, v) of $\mathcal{B}_\Phi$ encodes the configuration C of $\mathcal{A}_\Phi$ s.t. for all $\ell \in L^\Phi$: $C(\ell) = \{[v(x), v(y)] \mid (x, y) \in S(\ell)\}$. The other components of $\mathcal{B}_\Phi$ are defined as follows. ℓ_0 is s.t. $\ell_0(\ell_0^\Phi) = (x, y)$, where x and y are two clocks arbitrarily chosen from X, and $\ell_0(\ell) = \emptyset$ for all $\ell \in L^\Phi \setminus \{\ell_0^\Phi\}$. X is a set of clocks s.t. $|X| = M(\Phi)$. F is the set of all locations S s.t. $\{\ell \mid S(\ell) \neq \emptyset\} \subseteq F^\Phi$. Finally, δ allows $\mathcal{B}_\Phi$ to simulate the $f_\Phi^\star$-semantics of $\mathcal{A}_\Phi$: the non determinism of δ enables to guess which clocks must be grouped to form appropriate intervals (see [4] for details).

Theorem 14. *For all MITL formula Φ, $\mathcal{B}_\Phi$ has $M(\Phi)$ clocks and $O((|\Phi|)^{(m \cdot |\Phi|)})$ locations, where $m = \max_{I \in \mathcal{I}_\Phi} \left\{2 \times \left\lceil \frac{\inf(I)}{|I|} \right\rceil + 1, \left\lceil \frac{\sup(I)}{|I|} \right\rceil + 1\right\}$.*

5 Future Works: Towards Efficient MITL Model Checking

Let us close this work by several observations summarising what we believe are the *benefits* of using an OCATA based characterisation of MITL formulas, and that could yield efficient model checking algorithm for MITL. Let $\mathcal{C}$ be a timed automaton, and let Φ be an MITL formula. Obviously, one can perform *automaton-based model checking* by computing a TA $\mathcal{B}_{\neg\Phi}$ accepting $[\![\neg\Phi]\!]$ (using the technique presented in Section 4, or the technique of [2]), and explore their synchronous product $\mathcal{C} \times \mathcal{B}_{\neg\Phi}$ using classical region-based or zone-based techniques [1]. This approach could not be practical, as the number M of clocks of the TA $\mathcal{B}_{\neg\Phi}$ is usually very high, and the algorithm exploring $\mathcal{C} \times \mathcal{B}_{\neg\Phi}$ will have to maintain data structures (regions or zone) ranging over $N + M$ clocks, where N is the number of clocks of $\mathcal{C}$.

A way to avoid this blow up in the number of clocks is to perform the model-checking using the OCATA $\mathcal{A}_{\neg\Phi}$ (using its $f_{\neg\Phi}^\star$ semantics) instead of the TA $\mathcal{B}_{\neg\Phi}$. First, the size of $\mathcal{A}_{\neg\Phi}$ is *linear* in the size of Φ, and is straightforward to build. Second, a configuration of $\mathcal{C} \times \mathcal{A}_{\neg\Phi}$ stores only the clocks that correspond to *active copies* of $\mathcal{A}_{\neg\Phi}$, which, in practice, can be much smaller than the number of clocks of $\mathcal{B}_{\neg\Phi}$. Third, this approach allows to retain the structure of the OCATA in the transition system of $\mathcal{C} \times \mathcal{A}_{\neg\Phi}$, which allows to define *antichain based algorithms* [7], that rely on a *partial order on the state space* to detect redundant states and avoid exploring them. Such an approach, has been applied in the case of LTL model-checking [6]. It relies crucially on the translation of LTL formulas to *alternating automata*, and yields dramatic improvements in the practical performance of the algorithm.

To obtain such algorithms, we need a *symbolic data structure* to encode the configurations of $\mathcal{C} \times \mathcal{A}_{\neg\Phi}$. Such a data structure can be achieved by lifting, to our *interval semantics*, the technique from [12] that consists in encoding *regions* of OCATA configurations by means of *finite words*. Remark that this encoding differs from the classical regions for TA [1], in the sense that the *word* encoding allows the number of clocks to change along paths of the transition system.

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. 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), 142–170 (1992)
4. Brihaye, T., Estiévenart, M., Geeraerts, G.: On MITL and alternating timed automata Technical report arXiv.org. `http://arxiv.org/abs/1304.2814`
5. Clarke, E.M., Grumberg, O., Peled, D.: Model checking. MIT Press (2001)
6. De Wulf, M., Doyen, L., Maquet, N., Raskin, J.-F.: Antichains: Alternative algorithms for LTL satisfiability and model-checking. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 63–77. Springer, Heidelberg (2008)
7. 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)
8. Henzinger, T.A.: The temporal specification and verification of real-time systems. PhD thesis, Standford University (1991)
9. Koymans, R.: Specifying real-time properties with metric temporal logic. Real-Time Systems 2(4), 255–299 (1990)
10. Lasota, S., Walukiewicz, I.: Alternating timed automata. ACM Trans. Comput. Log. 9(2) (2008)
11. 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)
12. Ouaknine, J., Worrell, J.: On the decidability of metric temporal logic. In: LICS 2005. IEEE (2005)
13. Ouaknine, J., Worrell, J.: On the decidability and complexity of metric temporal logic over finite words. Logical Methods in Computer Science 3(1) (2007)
14. Pandya, P.K., Shah, S.S.: The Unary Fragments of Metric Interval Temporal Logic: Bounded versus Lower Bound Constraints. In: Chakraborty, S., Mukund, M. (eds.) ATVA 2012. LNCS, vol. 7561, pp. 77–91. Springer, Heidelberg (2012)
15. Raskin, J.-F., Schobbens, P.-Y.: The Logic of Event Clocks – Decidability, Complexity and Expressiveness. J. Automata, Languages and Combinatorics 4(3) (1999)
16. Vardi, M.Y., Wolper, P.: An automata-theoretic approach to automatic program verification (preliminary report). In: LICS 1986. IEEE (1986)

Predictability of Event Occurrences in Timed Systems

Franck Cassez[1] and Alban Grastien[2]

[1] NICTA* and UNSW
[2] NICTA and ANU, Australia

Abstract. We address the problem of predicting events' occurrences in partially observable timed systems modelled by timed automata. Our contribution is many-fold: 1) we give a definition of bounded predictability, namely k-predictability, that takes into account the minimum delay between the prediction and the actual event's occurrence; 2) we show that 0-predictability is equivalent to the original notion of predictability of S. Genc and S. Lafortune; 3) we provide a necessary and sufficient condition for k-predictability (which is very similar to k-diagnosability) and give a simple algorithm to check k-predictability; 4) we address the problem of predictability of events' occurrences in timed automata and show that the problem is PSPACE-complete.

1 Introduction

Monitoring and fault diagnosis aim at detecting defects that can occur at runtime. The monitored system is partially observable but a formal model of the system is available which makes it possible to build (offline) a monitor or a diagnoser. Monitoring and fault diagnosis for discrete event systems (DES) have been have been extensively investigated in the last two decades [1,2,3]. Fault diagnosis consists in detecting a fault *as soon as possible* after it occurred. It enables a system operator to stop the system in case something went wrong, or reconfigure the system to drive it to a safe state. *Predictability* is a strong version of diagnosability: instead of detecting a fault after it occurred, the aim is to *predict* the fault before its occurrence. This gives some time to the operator to choose the best way to stop the system or to reconfigure it.

In this paper, we address the problem of predicting event occurrences in partially observable timed systems modelled by timed automata.

The Predictability Problem. A timed automaton [4] (TA) generates a timed language which is a set of timed words which are sequences of pairs (event, timestamp). Only a subset of the events generated by the system is observable. The objective is to predict occurrences of a particular event (observable or not) based on the sequences of observable events. Automaton G, Fig. 1, is a timed version

* NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

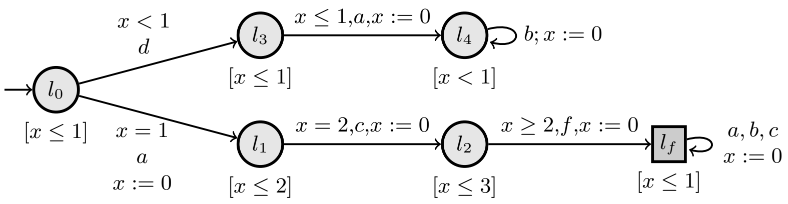


Fig. 1. Example G from [5]

of the example of automaton G_1 of [5]. The set of observable events is $\{a, b, c\}$. We would like to predict event f without observing event d. First consider the untimed version of G by ignoring the constraints on clock x. The untimed automaton can generate two types of events' sequences: $d.a.b^*$ and $a.c.f.\{a, b, c\}^*$. Because d is unobservable, after observing a we do not know whether the system is in location l_4 or l_1 and cannot predict f as, according to our knowledge, it is not bound to occur in all possible futures from locations l_4 or l_1. However, after the next observable event, b or c, we can make a decision: if we observe $a.c$, G must be in l_2 and thus f is going to happen next. After observing $a.c$ we can predict event f. Note that there is no quantitative duration between occurrences of events in discrete event systems and thus we can predict f at a *logical* time which is before f occurs. The time that separates the prediction of f from the actual occurrence of f is measured in the number of discrete steps G can make. In this sense G is 0-predictable as when we predict f, it is the next event to occur. The untimed version of G is an abstraction of a real system, and in the real system, it could be that f is going to occur 5 seconds after we observe c.

Timed automata enable us to capture quantitative aspects of real-time systems. We can use *clocks* (like x) to specify constraints between the occurrences of events. Moreover *invariants* (like $[x \leq 1]$) ensure that G changes location when the upper bound of the invariant is reached. In the timed automaton G, the (infinite) sequences with no f are of the form $(d, \delta_d)(a, \delta_a)(b, \delta_b)\cdots$ with $\delta_d < 1, \delta_a \leq 1$ and $\delta_b < 2$. The sequences with event f are of the form $(a, 1)(c, 3)(f, \delta_f)$ with $5 \leq \delta_f \leq 6$. Thus if we do *not* observe a "b" within the first two time units, we know that the system is in location l_1. This implies that f is going to occur, and we know this at time 2. But f will not occur before $1+2$ time units, the time for c to occur (from time 2) and the minimum time for f to occur after c. G is thus 3-predictable. In the sequel we formally define the previous notions and give efficient algorithms to solve the predictability problem.

Related Work. Predictability for discrete event systems was first proposed by S. Genc and S. Lafortune in [6]. Later in [5] they gave two algorithms to decide the predictability problem, one of them is a polynomial decision procedure. T. Jéron, H. Marchand, S. Genc and S. Lafortune [7] extended the previous results to occurrences of *patterns* (of events) rather than a single event. L. Brandán Briones and A. Madalinski in [8] studied *bounded* predictability without relating it to the notion defined by S. Genc and S. Lafortune.

Predictability is closely related to *fault diagnosis* [1,2,3]. The objective of fault diagnosis is to detect the occurrence of a special event, a fault, which is unobservable, as soon as possible after it occurs. Fault diagnosis for timed automata has first been studied by S. Tripakis in [9] and he proved that the diagnosis problem is PSPACE-complete. P. Bouyer, F. Chevalier and D. D'Souza [10] later studied the problem of computing a diagnoser with *fixed* resources (a deterministic TA) and proved that this problem is 2EXPTIME-complete. To the best of our knowledge the predictability problem for TA has not been investigated yet.

Our Contribution. We give a new characterization of bounded predictability and show it is equivalent to the definition of S. Genc and S. Lafortune. This new characterization is simple and dual to the one for the diagnosis problem; we can derive easily algorithms to decide predictability, bounded predictability, and to compute the largest anticipation delay to predict a fault. We also study the bounded predictability problem for TA and prove it is PSPACE-complete. We investigate implementability issues, i.e., how to build a *predictor*, and solve the *sampling predictability* problem which ensures an implementable predictor exists. We show how to compute bounded predictability with UPPAAL [11].

Omitted proofs and further details can be found in the extended version of this paper [12].

2 Preliminaries

$\mathbb{B} = \{\text{TRUE}, \text{FALSE}\}$ is the set of boolean values, $\mathbb{N}$ the set of natural numbers, $\mathbb{Z}$ the set of integers and $\mathbb{Q}$ the set of rational numbers. $\mathbb{R}$ is the set of real numbers and $\mathbb{R}_{\geq 0}$ is the set of non-negative reals.

Let X be a finite set of variables called *clocks*. A *clock valuation* is a mapping $v : X \to \mathbb{R}_{\geq 0}$. We let $\mathbb{R}_{\geq 0}^X$ be the set of clock valuations over X. We let $\mathbf{0}_X$ be the *zero* valuation where all the clocks in X are set to 0 (we use $\mathbf{0}$ when X is clear from the context). Given $\delta \in \mathbb{R}$, $v + \delta$ denotes the valuation defined by $(v + \delta)(x) = v(x) + \delta$. We let $\mathcal{C}(X)$ be the set of *convex constraints* on X which is the set of conjunctions of constraints of the form $x \bowtie c$ with $c \in \mathbb{N}$ and $\bowtie \in \{\leq, <, =, >, \geq\}$. Given a constraint $g \in \mathcal{C}(X)$ and a valuation v, we write $v \models g$ if g is satisfied by v. Given $R \subseteq X$ and a valuation v, $v[R]$ is the valuation defined by $v[R](x) = v(x)$ if $x \notin R$ and $v[R](x) = 0$ otherwise.

A finite (resp. infinite) *timed word* over Σ is a word in $\mathbb{R}_{\geq 0}.(\Sigma.\mathbb{R}_{\geq 0})^*$ (resp. $(\mathbb{R}_{\geq 0}.\Sigma)^\omega$). We write timed words as $0.4\ a\ 1.0\ b\ 2.7\ c \cdots$ where the real values are the durations elapsed between two events: thus c occurs at global time 4.1. We let $\mathit{Dur}(w)$ be the duration of a timed word w which is defined to be the sum of the durations (in $\mathbb{R}_{\geq 0}$) which appear in w; if this sum is infinite, the duration is ∞. Note that the duration of an infinite word can be finite, and such words which still contain an infinite number of events, are called *Zeno* words. An infinite timed word w is *time-divergent* if $\mathit{Dur}(w) = \infty$. We let $\mathit{Unt}(w)$ be the *untimed* version of w obtained by erasing all the durations in w, e.g., $\mathit{Unt}(0.4\ a\ 1.0\ b\ 2.7\ c\ 0) = abc$. Given w a timed word and $a \in \Sigma$, $|w|_a$ is the number of occurrences of a in w (∞ if a occurs infinitely often in w.)

$TW^*(\Sigma)$ is the set of finite timed words over Σ, $TW^\omega(\Sigma)$, the set of infinite timed words and $TW^\infty(\Sigma) = TW^*(\Sigma) \cup TW^\omega(\Sigma)$. We use Σ^* and Σ^ω for the corresponding sets of untimed words. A *timed language* is any subset of $TW^\infty(\Sigma)$. For $L \subseteq TW^\infty(\Sigma)$, we let $\mathit{Unt}(L) = \{\mathit{Unt}(w) \mid w \in L\}$.

For $w \in TW^*(\Sigma)$ and $w' \in TW^\infty(\Sigma)$, $w.w'$ is the concatenation of w and w'. A finite timed word w is a prefix of $w' \in TW^\infty(\Sigma)$ if $w' = w.w''$ for some $w'' \in TW^\infty(\Sigma)$. In the sequel we also the prefix operator and $\overline{L}$ is the set of finite words that are prefixes of words in L.

Let $\Sigma_1 \subseteq \Sigma$. $\boldsymbol{\pi}_{/\Sigma_1}$ is the projection of timed words of $TW^\infty(\Sigma)$ over timed words of $TW^\infty(\Sigma_1)$. When projecting a timed word w on a $\Sigma_1 \subseteq \Sigma$, the durations elapsed between two events are set accordingly: $\boldsymbol{\pi}_{/\{a,c\}}(0.4\ a\ 1.0\ b\ 2.7\ c) = 0.4\ a\ 3.7\ c$ (projection erases some events but preserves the time elapsed between the non-erased events). It follows that $\boldsymbol{\pi}_{/\Sigma_1}(w) = \boldsymbol{\pi}_{/\Sigma_1}(w')$ implies that $\mathit{Dur}(w) = \mathit{Dur}(w')$. For $L \subseteq TW^\infty(\Sigma)$, $\boldsymbol{\pi}_{/\Sigma_1}(L) = \{\boldsymbol{\pi}_{/\Sigma_1}(w) \mid w \in L\}$.

Timed automata (TA) are finite automata extended with real-valued clocks to specify timing constraints between occurrences of events. For a detailed presentation of the fundamental results for timed automata, the reader is referred to the seminal paper of R. Alur and D. Dill [4]. As usual we use the symbol ε to denote the silent (invisible) action in an automaton.

Definition 1 (Timed Automaton). *A* Timed Automaton *A is a tuple $(L, l_0, X, \Sigma \cup \{\varepsilon\}, E, \mathit{Inv}, F, R)$ where: L is a finite set of* locations*; l_0 is the* initial location*; X is a finite set of* clocks*; Σ is a finite set of* events*; $E \subseteq L \times \mathcal{C}(X) \times \Sigma \cup \{\varepsilon\} \times 2^X \times L$ is a finite set of* transitions*; for $(\ell, g, a, r, \ell') \in E$, g is the* guard*, a the* event*, and r the* reset *set; $\mathit{Inv} : L \to \mathcal{C}(X)$ associates with each location an* invariant*; as usual we require the invariants to be conjunctions of constraints of the form $x \preceq c$ with $\preceq \in \{<, \leq\}$. $F \subseteq L$ and $R \subseteq L$ are respectively the* final *and* repeated *sets of locations.* ■

A *state* of A is a pair $(\ell, v) \in L \times \mathbb{R}^X_{\geq 0}$. A *run* ϱ of A from (ℓ_0, v_0) is a (finite or infinite) sequence of alternating *delay* and *discrete* moves:

$$\varrho = (\ell_0, v_0) \xrightarrow{\delta_0} (\ell_0, v_0 + \delta_0) \xrightarrow{a_1} (\ell_1, v_1) \cdots \xrightarrow{a_n} (\ell_n, v_n) \xrightarrow{\delta_n} (\ell_n, v_n + \delta_n) \cdots$$

s.t. for every $i \geq 0$:

– $v_i + \delta \models \mathit{Inv}(\ell_i)$ for $0 \leq \delta \leq \delta_i$ (Def. 1 implies that $v_i + \delta_i \models \mathit{Inv}(\ell_i)$ is equivalent);
– there is a transition $(\ell_i, g_i, a_{i+1}, r_i, \ell_{i+1}) \in E$ s.t. : (i) $v_i + \delta_i \models g_i$ and (ii) $v_{i+1} = (v_i + \delta_i)[r_i]$ (by the previous condition we have $v_{i+1} \models \mathit{Inv}(\ell_{i+1})$.)

If ϱ is finite and ends in s_n, we let $\mathit{tgt}(\varrho) = s_n$. We say that event $a \in \Sigma \cup \{\varepsilon\}$ is *enabled* in $s = (\ell, v)$, written $a \in \mathit{en}(s)$, if there is a transition $(\ell, g, a, R, \ell') \in E$ s.t. $v \models g$ and $v[R] \models \mathit{Inv}(\ell')$. The set of finite (resp. infinite) runs from a state s is denoted $\mathit{Runs}^*(s, A)$ (resp. $\mathit{Runs}^\omega(s, A)$) and we define $\mathit{Runs}^*(A) = \mathit{Runs}^*((l_0, \mathbf{0}), A)$ and $\mathit{Runs}^\omega(A) = \mathit{Runs}^\omega((l_0, \mathbf{0}), A)$.

We make the following *boundedness* assumption on timed automata: time-progress in every location is bounded. This is not a restrictive assumption as every timed automaton that does not satisfy this requirement can be transformed into a language-equivalent one that is bounded [13]. This implies that every infinite run has an infinite number of events. We further assume[1] that every infinite run has an infinite number of discrete transitions with $a \neq \varepsilon$.

The *trace*, $tr(\varrho)$, of a run ϱ is the timed word $\delta_0 a_1 \delta_1 a_2 \cdots a_n \delta_n \cdots$ where ε is removed (and durations are updated accordingly). We let $Dur(\varrho) = Dur(tr(\varrho))$. For $V \subseteq Runs^*(A) \cup Runs^\omega(A)$, we let $Tr(V) = \{tr(\varrho) \mid \varrho \in V\}$.

A finite (resp. infinite) timed word w is *accepted* by A if $w = tr(\varrho)$ for some $\varrho \in Runs^*(A)$ that ends in an F-location (resp. for some $\varrho \in Runs^\omega(A)$ that reaches infinitely often an R-location). $\mathcal{L}^*(A)$ (resp. $\mathcal{L}^\omega(A)$) is the set of traces of finite (resp. infinite) timed words accepted by A. In the sequel we often omit the sets R and F in TA and this implicitly means $F = L$ and $R = L$.

Definition 2 (Product of TA). *Let $A_i = (L_i, l_0^i, X_i, \Sigma \cup \{\varepsilon\}, E_i, Inv_i, F_i, R_i)$, $i \in \{1,2\}$, be TA s.t. $X_1 \cap X_2 = \varnothing$. The* product *of A_1 and A_2 is the TA $A_1 \times A_2 = (L, l_0, X, \Sigma \cup \{\varepsilon\}, E, Inv, R, F)$ defined by: $L = L_1 \times L_2$; $l_0 = (l_0^1, l_0^2)$; $X = X_1 \cup X_2$; and $E \subseteq L \times \mathcal{C}(X) \times \Sigma \cup \{\varepsilon\} \times 2^X \times L$ and $((\ell_1, \ell_2), g_{1,2}, \sigma, r_{1,2}, (\ell'_1, \ell'_2)) \in E$ if:*

- *either $\sigma \neq \varepsilon$, and (i) $(\ell_k, g_k, \sigma, r_k, \ell'_k) \in E_k$ for $k = 1$ and $k = 2$; (ii) $g_{1,2} = g_1 \wedge g_2$ and (iii) $r_{1,2} = r_1 \cup r_2$;*
- *or $\sigma = \varepsilon$ and for $k \in \{1,2\}$, (i) $(\ell_k, g_k, \sigma, r_k, \ell'_k) \in E_k$; (ii) $g_{1,2} = g_k$, (iii) $r_{1,2} = r_k$ and (iv) $\ell'_{3-k} = \ell_{3-k}$;*

$Inv(\ell_1, \ell_2) = Inv(\ell_1) \wedge Inv(\ell_2)$, $F = F_1 \times F_2$ and R is defined[2] *such that $\mathcal{L}^\omega(A_1) \cap \mathcal{L}^\omega(A_2) = \mathcal{L}^\omega(A_1 \times A_2)$.* ■

A finite automaton (FA) is a TA with $X = \varnothing$: guards and invariants are vacuously true and time elapsing transitions do not exist.

We write $A = (L, l_0, \Sigma \cup \{\varepsilon\}, E, F, R)$ for a FA. A run of a FA A is thus a sequence of the form: $\varrho = \ell_0 \xrightarrow{a_1} \ell_1 \cdots \quad \cdots \xrightarrow{a_n} \ell_n \cdots$ where for each $i \geq 0$, $(\ell_i, a_{i+1}, \ell_{i+1}) \in E$. Definitions of traces and languages are inherited from TA but the duration of a run ϱ is the number of steps (including ε-steps) of ϱ: if ϱ is finite and ends in ℓ_n, $Dur(\varrho) = n$ and otherwise $Dur(\varrho) = \infty$. The product definition also applies to finite automata.

3 Predictability Problems

Predictability problems are defined on partially observable TA. Given a TA $A = (L, \ell_0, X, \Sigma, E, Inv, L, L)$, $\Sigma_o \subseteq \Sigma$ a set of *observable* events, and a *bound*

[1] Otherwise the trace of an infinite word can have a finite number of events in Σ but still infinite duration which cannot be defined in our setting. This is not a compulsory assumption and can be removed at the price of longer (not more complex) proofs.

[2] The product of Büchi automata requires an extra variable to keep track of the automaton that repeated its state. For the sake of simplicity we ignore this and assume the set R can be defined to ensure $\mathcal{L}^\omega(A_1) \cap \mathcal{L}^\omega(A_2) = \mathcal{L}^\omega(A_1 \times A_2)$.

$\Delta \in \mathbb{N}$, we want to predict the occurrences of event $f \in \Sigma$ at least Δ time units before they occur. Without loss of generality, we assume 1) that the target location of the f-transitions is l_f, and they all reset a dedicated clock of A, x, which is only used on f-transitions; 2) A has transitions $(l_f, \text{TRUE}, a, \{x\}, l_f)$ for every $a \in \Sigma_o$. We let $\mathit{Inv}(l_f) = x \leq 1$. In the remaining of this paper, Σ_o is fixed and we use $\boldsymbol{\pi}$ for $\boldsymbol{\pi}_{/\Sigma_o}$.

We again make the assumption that every infinite run of A contains infinitely many Σ_o events: this is not compulsory but simplifies some of the proofs.

3.1 Δ-Predictability

A run ρ of A is *non-faulty* if $\mathit{Unt}(\mathit{tr}(\rho))$ does not contain event f; otherwise it is *faulty*. We write $\mathit{NonFaulty}(s, A)$ for the non-faulty runs from s and define $\mathit{NonFaulty}(A) = \mathit{NonFaulty}((l_0, \mathbf{0}), A)$. Let $\varrho \in \mathit{NonFaulty}(A)$ be a finite non-faulty run:

$$\varrho = (l_0, v_0) \xrightarrow{\delta_0} (l_0, v_0 + \delta_0) \xrightarrow{a_1} (l_1, v_1) \cdots \xrightarrow{a_n} (l_n, v_n) \xrightarrow{\delta_n} (l_n, v_n + \delta_n).$$

ϱ is *Δ-prefaulty*, if it can be extended by a run ϱ' as follows:

$$\varrho'' = (l_0, v_0) \xrightarrow{\delta_0} \cdots \xrightarrow{\delta_n} \underbrace{tgt(\varrho) \xrightarrow{\delta'_0} s'_1 \xrightarrow{a'_1\delta'_1} \cdots \xrightarrow{a'_k\delta'_k} \cdots \xrightarrow{a'_j\delta'_j} s_j}_{\text{run } \varrho'}$$

where the extended run $\varrho'' \in \mathit{NonFaulty}(A)$ satisfies: (i) $f \in \mathit{en}(s_j)$ and (ii) $\mathit{Dur}(\rho') \leq \Delta$ (i.e., $\sum_{k=0}^{j} \delta'_k \leq \Delta$.) In words, f can occur within Δ time units from $\mathit{tgt}(\varrho)$. We let $\mathit{PreFaulty}_{\leq\Delta}(A)$ be the set of Δ-prefaulty runs of A. Note that if $\Delta \leq \Delta'$ then $\mathit{PreFaulty}_{\leq 0}(A) \subseteq \mathit{PreFaulty}_{\leq\Delta}(A) \subseteq \mathit{PreFaulty}_{\leq\Delta'}(A)$.

We want to predict the occurrence of event f at least Δ time units before it occurs and it makes sense only if $\Delta \leq \kappa(A)$ where $\kappa(A)$ is the minimum duration to reach a state where f is enabled. If f is never enabled, we let $\kappa(A) = \infty$. If $\kappa(A)$ is finite, let $0 \leq \Delta \leq \kappa(A)$ and define the following timed languages:

$$L^{\omega}_{\neg f} = \mathcal{L}^{\omega}(A) \cap \mathit{Tr}(\mathit{NonFaulty}(A)) \tag{1}$$

$$L^{-\Delta}_{f} = \mathit{Tr}(\mathit{PreFaulty}_{\leq\Delta}(A)). \tag{2}$$

If $\kappa(A) = \infty$ then we let $L^{-\Delta}_{f} = \varnothing$. $L^{\omega}_{\neg f}$ contains the infinite non-faulty traces of A. $L^{-\Delta}_{f}$ contains the finite traces w of A that can be extended into $w.x.f$ with f occurring less then Δ time units after w.

A *Δ-Predictor* is a device that predicts the occurrence of f at least Δ time units before it occurs. It should do it observing only the projection $\boldsymbol{\pi}(w)$ of the current trace w. Thus for every word $w \in L^{-\Delta}_{f}$, the predictor predicts f by issuing a 1. On the other hand, if a trace w can be extended as an infinite trace without any event f, i.e., it is in $\overline{L^{\omega}_{\neg f}}$, the predictor must not predict f and thus should issue a 0. For a trace which is in $L^{-\Delta'}_{f}$ with $\Delta' > \Delta$ and not in $\overline{L^{\omega}_{\neg f}}$, we do not require anything from the predictor: it can predict f or not and this is why we define a predictor as a partial mapping.

Definition 3 (Δ-Predictor). *A* Δ-predictor *for A is a partial mapping $P : TW^*(\Sigma_o) \longrightarrow \{0, 1\}$ such that:*

- $\forall w \in L_f^{-\Delta}, P(\boldsymbol{\pi}(w)) = 1$,
- $\forall w \in \overline{L^{\omega}_{\neg f}}, P(\boldsymbol{\pi}(w)) = 0$.

A is Δ-predictable *if there exists a Δ-predictor for A and is* predictable *if there is some Δ such that A is Δ-predictable.* ■

It follows that if f is never enabled in A, A is Δ-predictable for any Δ: a predictor is a mapping $P(\cdot) = 0$. In the sequel we assume that A contains a state where f is enabled and thus $\kappa(A)$ is finite.[3]

In the dual problem of *diagnosability* [9], it is required that the infinite words in $L^{\omega}_{\neg f}$ be *non-Zeno.* This is required by the problem statement that time must advance beyond any bound. For predictability, this is not a requirement and we could accept non time-divergent runs in $L^{\omega}_{\neg f}$. However for realistic systems we should add this requirement. This can be easily done and we discuss how to do this in section 5.2.

3.2 PSPACE-Hardness of Bounded Predictability

We are interested in the two following problems:

Problem 1 (Δ-Predictability (Bounded Predictability))
INPUT: *A TA $A = (L, \ell_0, X, \Sigma, E, Inv)$ and $\Delta \in \mathbb{N}$.*
PROBLEM: *Is A Δ-predictable?*

Problem 2 (Predictability)
INPUT: *A TA $A = (L, \ell_0, X, \Sigma, E, Inv)$.*
PROBLEM: *Is A predictable?*

Notice that predictability problems for finite automata are defined using the number of steps in the automaton A (including unobservable steps) for the duration of a run. A first result is the PSPACE-hardness of the Bounded Predictability problem. This is obtained by reducing the *reachability problem* for TA to the Bounded Predictability problem. The *location reachability problem* for TA asks, given a location l, whether (l, v) (for some valuation v) is reachable from the initial state of A. This problem is PSPACE-complete for TA [4].

Theorem 1. *The Bounded Predictability problem is PSPACE-hard for TA.*

Proof. We can reduce the location reachability problem for bounded TA to the predictability problem as follows (the reduction is similar to [9]): let A be a bounded TA and l a location of A. We can build A' by adding transitions to A: let END by a new location. We add a transition $(l, \text{TRUE}, f, \{x\}, \text{END})$, and another one $(l, \text{TRUE}, u, \{x\}, \text{END})$ with u unobservable, assuming A has at least

[3] Checking whether a state where f is enabled is reachable and the computation of $\kappa(A)$ can be done in PSPACE [14] for TA and linear time for FA.

one clock x. We then add loops on location END $(\text{END}, x = 1, a, \{x\}, \text{END})$, for each $a \in \Sigma$. Moreover $Inv(\text{END}) = x \leq 1$. It follows from our definition of predictability that l is reachable in A iff A' is not predictable, and A' has size polynomial in A.

3.3 Necessary and Sufficient Condition for Δ-Predictability

We now give a necessary and sufficient condition (NSC) for Δ-predictability which is similar in form to the condition used for Δ-*diagnosability* [9].

Lemma 1. *A is Δ-predictable iff* $\boldsymbol{\pi}(L_f^{-\Delta}) \cap \boldsymbol{\pi}(\overline{L_{\neg f}^{\omega}}) = \varnothing$.

Proof. Only If. Assume A is Δ-predictable. There exists a partial mapping P s.t. $\forall w \in L_f^{-\Delta}, P(\boldsymbol{\pi}(w)) = 1$, $\forall w \in \overline{L_{\neg f}^{\omega}}, P(\boldsymbol{\pi}(w)) = 0$. Assume $w \in \boldsymbol{\pi}(L_f^{-\Delta}) \cap \boldsymbol{\pi}(\overline{L_{\neg f}^{\omega}}) \neq \varnothing$. Then $w = \boldsymbol{\pi}(w_1) = \boldsymbol{\pi}(w_2)$ with $w_1 \in L_f^{-\Delta}$ and $w_2 \in \overline{L_{\neg f}^{\omega}}$. By definition of P we must have $P(w) = P(\boldsymbol{\pi}(w_1)) = 1$ and $P(w) = P(\boldsymbol{\pi}(w_2)) = 0$ which is a contradiction.

If. If $\boldsymbol{\pi}(L_f^{-\Delta}) \cap \boldsymbol{\pi}(\overline{L_{\neg f}^{\omega}}) = \varnothing$ define $P(w) = 1$ if $w \in \boldsymbol{\pi}(L_f^{-\Delta})$ and $P(w) = 0$ otherwise. If P does not exist, we must have $w = \boldsymbol{\pi}(w_1) = \boldsymbol{\pi}(w_2)$ with $w_1 \in L_f^{-\Delta}$ and $w_2 \in \overline{L_{\neg f}^{\omega}}$. In this case $w \in \boldsymbol{\pi}(L_f^{-\Delta}) \cap \boldsymbol{\pi}(\overline{L_{\neg f}^{\omega}})$ which is a contradiction. $\square$

From Lemma 1 we can prove the following Proposition and Theorem:

Proposition 1. *if $\Delta \leq \Delta'$ and A is Δ'-predictable, then A is Δ-predictable.*

Proof. $L_f^{-\Delta} \subseteq L_f^{-\Delta'}$ and thus $\boldsymbol{\pi}(L_f^{-\Delta}) \cap \boldsymbol{\pi}(\overline{L_{\neg f}^{\omega}}) \subseteq \boldsymbol{\pi}(L_f^{-\Delta'}) \cap \boldsymbol{\pi}(\overline{L_{\neg f}^{\omega}})$. $\square$

Theorem 2. *A is predictable iff A is 0-predictable.*

In the next section, we focus on the Δ-predictability problem for finite automata and discuss how it generalizes the previous notion introduced by S. Genc and S. Lafortune in [5]. Section 5 tackles the Δ-predictability problem for TA.

4 Predictability for Discrete Event Systems

In this section, we address the predictability problems for discrete event systems specified by FA. We first show that the definition of predictability (Def. 3) we introduced in Section 3 is equivalent to the original definition of predictability by S. Genc and S. Lafortune in [5].

4.1 Original Definition of Predictability (S. Genc and S. Lafortune)

Let $L_f = Tr(PreFaulty_{\leq 0}(A))$ be the set of non-faulty traces that can be extended with a fault in one step, and $L_{\neg f} = \overline{Tr(NonFaulty(A))}$ be the set of finite prefixes of non-faulty traces. S. Genc and S. Lafortune originally defined predictability for discrete event systems in [5] and we refer to GL-predictability for this definition. GL-predictability is defined as follows[4]:

$$\exists n \in \mathbb{N}, \forall w \in L_f, \exists t \in \overline{w} \text{ such that } \mathbf{P}(t) \tag{3}$$

with $\mathbf{P}(t)$ defined by:

$$\mathbf{P}(t) : \forall u \in L_{\neg f}, \forall v \in \mathcal{L}(A)/u, \boldsymbol{\pi}(u) = \boldsymbol{\pi}(t) \wedge |v| \geq n \implies |v|_f > 0.$$

From [5], A is GL-predictable iff Equation (3) is satisfied. GL-predictability as defined by Equation (3) is equivalent to our notion of predictability:

Theorem 3. *A is GL-predictable iff A is 0-predictable.*

4.2 Checking k-Predictability

To check whether A is k-predictable, $0 \leq k \leq \kappa(A)$, we can use the NSC we established in Lemma 1: A is k-predictable iff $\boldsymbol{\pi}(L_f^{-k}) \cap \boldsymbol{\pi}(\overline{L_{\neg f}^{\omega}}) = \varnothing$. To check this condition, it suffices to build a *twin plant* (similar to [5] and to what is defined for fault diagnosis [2]). We define two automata $A_1(k)$ and A_2 that accept $\boldsymbol{\pi}(L_f^{-k})$ and $\boldsymbol{\pi}(\overline{L_{\neg f}^{\omega}})$ and synchronize them to check whether the intersection is empty. The first automaton $A_1(k)$ accepts finite words which are in $\boldsymbol{\pi}(L_f^{-k})$ and is defined as follows:

1. in A, we compute the set of states F_k that can reach a state where f is enabled within k steps (this can be done in linear time using a backward breadth-first search from states where f is enabled.)
2. $A_1(k)$ is a copy of A where the set of final states is F_k, and every $a \notin \Sigma_o$ is replaced by ε.

It follows that $A_1(k)$ accepts $\boldsymbol{\pi}(L_f^{-k})$.

The second automaton A_2 accepts $\boldsymbol{\pi}(\overline{L_{\neg f}^{\omega}})$. To compute it, we merely need to compute the states from which there is an infinite path without any state where f is enabled. This can be done in linear time again (e.g., computing the states that satisfy the CTL formula $\mathsf{EG}\neg en(f)$.) A_2 is defined as follows:

1. let $F_{\neg f}$ be the set of states in A from which there exists an infinite path with no states where f is enabled.
2. A_2 is a copy of A restricted to the set of states $F_{\neg f}$, and every $a \notin \Sigma_o$ is replaced by ε (this implies that the target state of the f transitions cannot be in A_2).

From the previous construction with sets of accepting states F_k for $A_1(k)$ and $F_{\neg f}$ for A_2 (every state in A_2 is accepting), $\mathcal{L}^*(A_1(k) \times A_2) = \boldsymbol{\pi}(L_f^{-k}) \cap \boldsymbol{\pi}(\overline{L_{\neg f}^{\omega}})$ and we can check k-predictability in quadratic time in the size of A.

Example 1. For the untimed version of Automaton G (Fig. 1, page 63), we obtain $G_1(0)$ and G_2 as depicted on Fig. 2. Recall that d is unobservable.

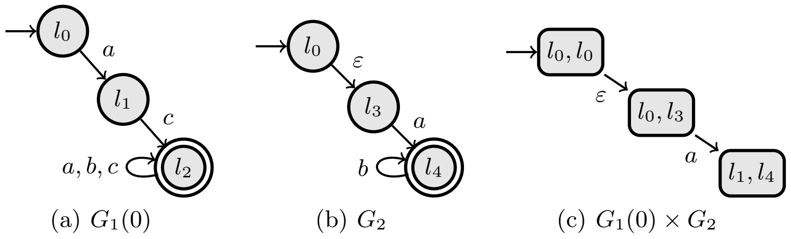


Fig. 2. Construction of G_1 and G_2 for automaton G ((Fig. 1)

Computing the largest k such that A is k-predictable can also be done in quadratic time. In A, we can compute, in linear time, the shortest distance $d_f(q)$ (going backwards) from q to a state where f is enabled (it is ∞ if q is unreachable going backwards in A). In the product $A_1(k) \times A_2$, if there is a run from the initial state to (s_1, s_2) and $d(s_1) = k', k' \leq k$, this implies that A is not k'-predictable. To determine the largest k such that A is k-predictable, it suffices to perform the following steps:

1. compute the shortest distance $d_f(q)$ to an f-enabled state for each $q \in Q$;
2. build the product $A_1(0) \times A_2$;
3. let S be the set of reachable states in $A_1(0) \times A_2$ and $M = \min_{(s_1,s_2) \in S} d_f(s_1)$.

The largest k such that A is k-predictable is $M - 1$.

Example 2. On automaton G of Fig. 1: $d(l_2) = 0$, $d(l_1) = 1$, $d(l_0) = 2$, $d(l_3) = d(l_4) = \infty$. The minimum value reachable in $G_1(0) \times G_2$ is obtained for l_1 and is $d(l_1) = 1$. Thus G is 0-predictable.

5 Predictability for Timed Automata

In this section we address the predictability problems for TA. We first rewrite the NSC of Lemma 1 using infinite languages. This enables us 1) to deal with time-divergent runs and 2) to design an algorithm to solve the predictability problems for TA.

5.1 Checking Δ-Predictability

We can reformulate Lemma 1 without the prefix operator by extending $L_f^{-\Delta}$ into an equivalent language of infinite words: let $L_f^{\omega,-\Delta} = L_f^{-\Delta}.(\Sigma_o \times \mathbb{R}_{\geq 0})^\omega$.

Lemma 2. $\boldsymbol{\pi}(L_f^{-\Delta}) \cap \boldsymbol{\pi}(\overline{L_{\neg f}^{\omega}}) = \varnothing \iff \boldsymbol{\pi}(L_f^{\omega,-\Delta}) \cap \boldsymbol{\pi}(L_{\neg f}^{\omega}) = \varnothing$.

[4] Technically S. Genc and S. Lafortune let w range over $L_f.f$ and impose that $|t|_f = 0$; the definition we give in Equation (3) is equivalent to Definition 1 of [5].

To check Δ-predictability we build a product of timed automata $A_1(\Delta) \times A_2$, and reduce the problem to Büchi emptiness on this product. This construction is along the lines of the *twin plant* introduced in [2,9]. The difference in the predictability problem lies in the construction of $A_1(\Delta)$ which is detailed later. The twin plant idea is the following:

– $A_1(\Delta)$ accepts $\boldsymbol{\pi}(L_f^{\omega,-\Delta})$ i.e., (projections of) infinite timed words of the form $w.(\mathbb{R}_{\geq 0} \times \Sigma_o)^\omega$ with $w \in L_f^{-\Delta}$;
– A_2 accepts $\boldsymbol{\pi}(L_{\neg f}^\omega)$ i.e., (projections of) infinite non-faulty timed words in $L_{\neg f}^\omega$;
– the product $A_1(\Delta) \times A_2$ accepts the language $\boldsymbol{\pi}(L_f^{\omega,-\Delta}) \cap \boldsymbol{\pi}(L_{\neg f}^\omega)$;
– thus checking Δ-predictability of A reduces to Büchi emptiness checking on the product $A_1(\Delta) \times A_2$.

$A_1(\Delta)$ itself is made of two copies of A: the original A and a twin copy (see Fig. 3). A_1 starts in the initial location of A, ℓ_0, and at some point in time switches to the twin copy (grey area on Fig. 3). The purpose of the twin copy is to extend the previously formed timed word with a timed word of duration less than Δ time units that reaches a state where f is enabled. The actions performed in the copy do not matter as we only have to check that f is reachable within Δ time units since we switched to the copy. In this case the timed word built in the original A is in $L_f^{-\Delta}$.

$A_1(\Delta) = (L \cup \tilde{L} \cup \{\text{END}\}, l_0^1, X \cup \{y\}, \Sigma \cup \{\varepsilon\}, E_1, Inv_1, \varnothing, \{\text{END}\})$ is formally defined as follows[5] (see Fig. 3):

– $\tilde{L} = \{\tilde{\ell}, \ell \in L\}$ is the set of twin locations;
– $l_0^1 = l_0$; $A_1(\Delta)$ starts in the same initial state as A.
– $Inv_1(\ell) = Inv_1(\tilde{\ell}) = Inv(\ell)$; invariants are the same as in the original automaton A including the twin locations;
– the transition relation is defined as follows:
  • original transitions of A: $(\ell, g, a', R, \ell') \in E_1$ iff $(\ell, g, a, R, \ell') \in E$ and $a \in \Sigma_o \setminus \{f\}$; $a' = a$ if $a \in \Sigma_o$ and $a' = \varepsilon$ otherwise; this renaming hides the unobservable events by renaming them in ε.
  • transitions to the twin locations: $(\ell, \text{TRUE}, \varepsilon, \{y\}, \tilde{\ell}) \in E_1$ for each $\ell \in L$; A_1 can switch to the twin copy at any time and doing so preserves the values for the clocks in X but resets y;
  • equivalent unobservable transitions inside the twin copy: $(\tilde{\ell}, g, \varepsilon, R, \tilde{\ell}_1) \in E_1$ iff $(\ell, g, a, R, \ell_1) \in E$ for some $a \neq f$;
  • equivalent of f-transitions in the twin copy: $(\tilde{\ell}', g \wedge y \leq \Delta, \varepsilon, R, \text{END}) \in E_1$ iff $(\ell', g, f, R, l_f) \in E$.
  • loop transitions on observable events in the twin copy: $(\tilde{\ell}, \text{TRUE}, a, \varnothing, \tilde{\ell}) \in E_1$ for each $a \in \Sigma_o$. This enables A_2 (defined below) to synchronize with A_1 on Σ_o after A_1 has chosen to switch to the twin copy of A.

[5] For now ignore the NZ location in the Figure and the invariants $[y \leq k]$. Their sole purposes is to ensure time-divergence.

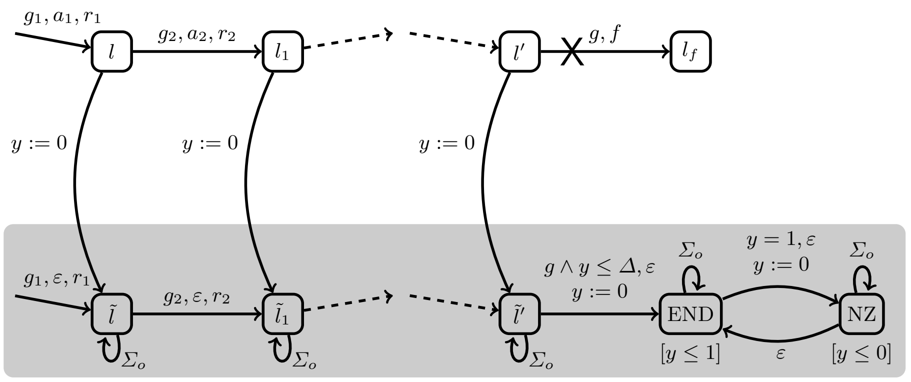


Fig. 3. Construction of Automaton A_1

Finally, A_2 is simply of copy of A without the f-transitions and the clocks are renamed to be local to A_2. Every location in A_2 is a *repeated* location. Notice that the only repeated location in $A_1(\Delta)$ is END. By definition of the synchronized product, $\mathcal{L}^\omega(A_1(\Delta) \times A_2) = \mathcal{L}^\omega(A_1(\Delta)) \cap \mathcal{L}^\omega(A_2)$.

Lemma 3. $\boldsymbol{\pi}(L_f^{\omega,-\Delta}) \cap \boldsymbol{\pi}(L_{\neg f}^\omega) = \mathcal{L}^\omega(A_1(\Delta) \times A_2)$.

Theorem 4. *Problems 1 and 2 are PSPACE-complete.*

Proof. PSPACE-easiness of Problem 1 is established as follows: checking Büchi emptiness for timed automata is in PSPACE [4]. The product $A_1(\Delta) \times A_2$ has size polynomial in the size of A and thus checking Büchi emptiness of the product is in PSPACE as well. Problem 1 is thus in PSPACE. By Theorem 2, Problem 2 is in PSPACE as well. Theorem 1 establishes PSPACE-hardness for Problem 1, and as it is equivalent to Problem 2, it is PSPACE-hard as well. $\square$

5.2 Restriction to Time-Divergent Runs of $L_{\neg f}^{\omega}$

To deal with time-divergence and enforce the runs in $L_{\neg f}^\omega$ to have infinite duration, we can add another automaton in the product with a Büchi condition that enforces time-divergence (this is how this kind of requirements is usually addressed). In our setting, we can re-use the fresh clock y of $A_1(\Delta)$ after location END is visited: it is not useful anymore to check whether a timed word is in $L_f^{-\Delta}$. The modifications to $A_1(\Delta)$ required to ensure time-divergence in A_2 are the following:

– add a new location NZ, which is now the repeated location of $A_1(\Delta)$;
– add two transitions as depicted on Fig. 3 between END and NZ.

This way infinite timed words accepted by $A_1(\Delta)$ must be time-divergent and with the synchronization with A_2 this forces the runs of A_2 to be time-divergent.

Finally, once we know how to solve Problem 1, we can compute the optimal (maximum) anticipation delay by performing a binary search on the possible values of $0 \leq \Delta \leq \kappa(A)$.

5.3 Implementability of the Δ-Predictor

In the previous sections, we defined a predictor as a mapping from timed words to $\{0,1\}$. To build an implementation of this mapping (an actual predictor) we still have some key problems to address: 1) we have to recognize when a timed word is in $L_f^{-\Delta}$; and 2) we have to detect that a timed word is in $L_f^{-\Delta}$ as soon as possible. S. Tripakis addressed similar problems in [9] in the context of fault diagnosis where a *diagnoser* is given as an algorithm that computes a state estimate of the system after reading a timed word w. The diagnoser updates its status after the occurrence of an observable event or after a *timeout* (TO) has occurred, which means some timed elapsed since the last update and no observable event occurred. The value of the timeout period (TO) is required to be less than the minimum delay between two observable events to ensure that the diagnoser works as expected. However, point 2) above still poses problem in our context, as demonstrated by the TA $\mathcal{B}$ of Fig. 4.

The set of observable events is $\{a\}$ and $\mathcal{B}$ is 4-predictable. To see this, define the predictor P as follows: for a timed word $w = \delta.w'$ with $\delta \geq 2$, $P(w) = 1$ and otherwise $P(w) = 0$. Indeed if 2 time units elapse and we see no observable events, for sure the system is still is l_0 and thus a fault f is bound to happen, but not before 4 time units. An implementation of a 4-predictor has to observe the state of the system *exactly* at time 2 otherwise it cannot predict the fault 4 time units in advance.

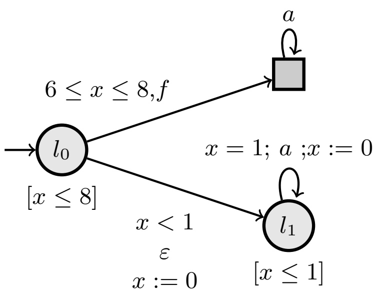


Fig. 4. The Timed Automaton $\mathcal{B}$

Now assume the platform on which we implement the predictor can make an observation every $\frac{3}{5}$ time units. The first observation of the predictor occurs at time $\frac{3}{5}$; the third at $\frac{9}{5}$ and we cannot predict the fault as we still don't know whether the system is in l_0 or has made a silent move to l_1. The next observation is at $\frac{12}{5}$: if we have seen no a so far, for sure the system is in l_0 and we can predict the fault. However the fault may now occur in $\frac{18}{5}$ time units i.e., less than 4 time units from the current time. Such a platform cannot implement the 4-predictor. The maximal anticipation delay we computed in the previous section is thus an *ideal* maximum that can be achieved by an *ideal* predictor that could monitor the system *continuously*. In a realistic system, there is a *sampling rate*, or at least a minimum amount of time between two observations [15]. In the sequel we address the *sampling predictability problem* that takes into account the speed of the platform.

5.4 Sampling Predictability

Let $\alpha \in \mathbb{Q}$ and L be a timed language. We let $L \bmod \alpha$ be the set of timed words in L with a duration multiple of α: $L \mod \alpha = \{w \in L, \exists k \in \mathbb{N}, \mathit{Dur}(w) = k \cdot \alpha\}$.

Given a *sampling rate* $\alpha \in \mathbb{Q}$, the *sampling* predictability problem is defined by refining the definition of a Δ-predictor: an (α, Δ)*-predictor* for A is a partial mapping $P : TW^*(\Sigma_o) \bmod \alpha \longrightarrow \{0, 1\}$ such that:

- $\forall w \in L_f^{-\Delta} \bmod \alpha, P(\boldsymbol{\pi}(w)) = 1$,
- $\forall w \in \overline{L_{\neg f}^{\omega}} \bmod \alpha, P(\boldsymbol{\pi}(w)) = 0$.

A timed automaton A is (α, Δ)-predictable if there exists a (α, Δ)-predictor for A and is α-predictable is there is some Δ such that A is (α, Δ)-predictable.

Remark 1. The problem of deciding whether there exists a sampling rate α such that A is α-predictable is also interesting but very likely to be undecidable as the existence of a *sampling rate* s.t. a location is reachable in a TA is undecidable [16].

The solution to the sampling predictability problem is a simple adaptation of the solution we presented in Section 5: in the construction of automaton $A_1(\Delta)$ (Fig. 3, page 73), it suffices to restrict the transitions from the original A to the twin copy (those resetting y) to happen at time points multiple of α. This can be achieved by adding a *sampler* timed automaton, and a common fresh clock, s, that *sampler* resets every α time units. The transitions resetting y in A_1 are now guarded by $s = 0$.

We can now safely define an implementation for an (α, Δ)-predictor along the lines of the *diagnoser* defined in [9]. The implementation performs an observation every α time units. It computes a state estimate of the system. If one of the states in the state estimate can reach a state where f is enabled within Δ time units, the predictor predicts f and issue a 1. Otherwise it issues 0. Computing a representation of the state estimate as a set of polyedra is a standard operation and can be done given an observed timed word w, and the timed automaton model A. Checking that one of the states in the estimate can reach an f-enabled state within Δ time units can also be done using standard reachability algorithm. It can be performed on-line or off-line by computing a polyedral representation of this set of states.

6 Conclusion and Future Work

In this paper we have proved some new results for predictability of events' occurrences for timed automata. We also contributed a new and simpler definition of bounded predictability for finite automata. In [10], P. Bouyer, F. Chevalier and D. D'Souza proposed an algorithm to decide the existence of a diagnoser with fixed resources (number of clocks and constants). The very same question arises for the existence of a predictor in timed systems. *Dynamic* observers [17] have been proposed in the context of fault diagnosis and *opacity* [18]; in [19] it is shown how to compute a most permissive observer that ensures diagnosability (or opacity [20]) and also how to compute an optimal observer [21] (w.r.t. to a given criterion). We can define the same problems for predictability. Given the similarities between the fault diagnosis and predictability problems, it would be interesting to state these two problems in a similar and unified way and design an algorithm that can solve the unified version.

References

1. Sampath, M., Sengupta, R., Lafortune, S., Sinnamohideen, K., Teneketzis, D.: Diagnosability of discrete event systems. IEEE Trans. on Auto. Cont. 40(9) (1995)
2. Yoo, T.S., Lafortune, S.: Polynomial-time verification of diagnosability of partially-observed discrete-event systems. IEEE Trans. on Auto. Cont. 47(9), 1491–1495 (2002)
3. Jiang, S., Huang, Z., Chandra, V., Kumar, R.: A polynomial algorithm for testing diagnosability of discrete event systems. IEEE Trans. on Auto. Cont. 46(8) (2001)
4. Alur, R., Dill, D.: A theory of timed automata. Theoretical Computer Science 126, 183–235 (1994)
5. Genc, S., Lafortune, S.: Predictability of event occurrences in partially-observed discrete-event systems. Automatica 45(2), 301–311 (2009)
6. Genc, S., Lafortune, S.: Predictability in discrete-event systems under partial observation. In: IFAC Symp. on Fault Detection, Supervision and Safety of Technical Processes, Beijing, China. IEEE (2006)
7. Jéron, T., Marchand, H., Genc, S., Lafortune, S.: Predictability of sequence patterns in discrete event systems. In: IFAC WC, Seoul, Korea, pp. 537–543 (2008)
8. Brandán Briones, L., Madalinski, A.: Bounded predictability for faulty discrete event systems. In: 30th Int. Conf. of the Chilean Computer Science Society (2011)
9. Tripakis, S.: Fault diagnosis for timed automata. In: Damm, W., Olderog, E.-R. (eds.) FTRTFT 2002. LNCS, vol. 2469, pp. 205–224. Springer, Heidelberg (2002)
10. Bouyer, P., Chevalier, F., D'Souza, D.: Fault diagnosis using timed automata. In: Sassone, V. (ed.) FOSSACS 2005. LNCS, vol. 3441, pp. 219–233. Springer, Heidelberg (2005)
11. Larsen, K.G., Pettersson, P., Yi, W.: Uppaal in a nutshell. STTT 1(1-2), 134–152 (1997)
12. Cassez, F., Grastien, A.: Predictability of Event Occurrences in Timed Systems. CoRR/abs arXiv:1306.0662 [cs.SY] (2013), http://arxiv.org/abs/1306.0662
13. Behrmann, G., Fehnker, A., Hune, T., Larsen, K.G., Pettersson, P., Romijn, J., Vaandrager, F.: Minimum-cost reachability for priced timed automata. In: Di Benedetto, M.D., Sangiovanni-Vincentelli, A.L. (eds.) HSCC 2001. LNCS, vol. 2034, pp. 147–161. Springer, Heidelberg (2001)
14. Courcoubetis, C., Yannakakis, M.: Minimum and maximum delay problems in real-time systems. Formal Methods in System Design 1(4), 385–415 (1992)
15. De Wulf, M., Doyen, L., Raskin, J.-F.: Almost ASAP semantics: From timed models to timed implementations. In: Alur, R., Pappas, G.J. (eds.) HSCC 2004. LNCS, vol. 2993, pp. 296–310. Springer, Heidelberg (2004)
16. Cassez, F., Henzinger, T.A., Raskin, J.-F.: A Comparison of Control Problems for Timed and Hybrid Systems. In: Tomlin, C.J., Greenstreet, M.R. (eds.) HSCC 2002. LNCS, vol. 2289, pp. 134–148. Springer, Heidelberg (2002)
17. Cassez, F., Tripakis, S.: Fault diagnosis with static and dynamic diagnosers. Fundamenta Informaticae 88(4), 497–540 (2008)
18. Cassez, F., Dubreil, J., Marchand, H.: Synthesis of opaque systems with static and dynamic masks. Formal Methods in System Design 40(1), 88–115 (2012)
19. Cassez, F., Tripakis, S., Altisen, K.: Sensor minimization problems with static or dynamic observers for fault diagnosis. In: ACSD 2007, pp. 90–99. IEEE Comp. Soc. (2007)
20. Cassez, F., Dubreil, J., Marchand, H.: Dynamic observers for the synthesis of opaque systems. In: Liu, Z., Ravn, A.P. (eds.) ATVA 2009. LNCS, vol. 5799, pp. 352–367. Springer, Heidelberg (2009)
21. Cassez, F., Tripakis, S., Altisen, K.: Synthesis of optimal-cost dynamic observers for fault diagnosis of discrete-event systems. In: TASE 2007, pp. 316–325. IEEE Comp. Soc. (2007)

Transience Bounds for Distributed Algorithms

Bernadette Charron-Bost[1], Matthias Függer[2], and Thomas Nowak[1]

[1] CNRS, LIX, École polytechnique, France
{charron,nowak}@lix.polytechnique.fr
[2] ECS Group, TU Wien, Austria
fuegger@ecs.tuwien.ac.at

Abstract. A large variety of distributed systems, like some classical synchronizers, routers, or schedulers, have been shown to have a periodic behavior after an initial transient phase (Malka and Rajsbaum, WDAG 1991). In fact, each of these systems satisfies recurrence relations that turn out to be linear as soon as we consider max-plus or min-plus algebra. In this paper, we give a new proof that such systems are eventually periodic and a new upper bound on the length of the initial transient phase. Interestingly, this is the first asymptotically tight bound that is linear in the system size for various classes of systems. Another significant benefit of our approach lies in the straightforwardness of arguments: The proof is based on an easy convolution lemma borrowed from Nachtigall (Math. Method. Oper. Res. 46) instead of purely graph-theoretic arguments and involved path reductions found in all previous proofs.

1 Introduction

A large variety of distributed systems, like the network synchronizers in [13], or the distributed link reversal algorithms [22] which can be used for routing [14], scheduling [4], distributed queuing [21,2], or resource allocation [7] have been shown to have a periodic behavior after an initial transient phase. Each of these systems satisfies a recurrence relation [16] that turns out to be linear as soon as one considers max-plus or min-plus algebra. Indeed, the fundamental theorem in these algebras—an analog of the Perron-Frobenius theorem—shows that any linear system with irreducible system matrix is periodic from some index, called the *transient* of the system. For all the above mentioned systems, the study of the transient plays a key role in characterizing performances: For example, in the case of link reversal routing, the system transient is equal to the time complexity of the routing algorithm.

Hartmann and Arguelles [17] have shown that the transients of linear systems are computable in polynomial time. However, their algorithms provide no analysis of the transient phase, and do not hint at the parameters that influence system transients. Conversely, upper bounds involving these parameters help to predict the duration of the transient phase, and to define strategies to reduce transients during system design. From both numerical and methodological viewpoints, it is therefore important to determine accurate transience bounds.

The problem of bounding the transients has already been studied [1,5,8,13,17,20]. A polynomial bound has been established by Even and Rajsbaum [13] for a large class of synchronizers in the specific case of integer delays, and Hartmann and Arguelles [17] have established an upper bound on the transients of general linear systems. More recently, Charron-Bost et al. [8] have given two new transience bounds, called the *repetitive* and the *explorative* bound, which both improve on the Even and Rajsbaum's synchronizer bound, and are incomparable with Hartmann and Arguelles' bound. In each of the above mentioned works [13,17,8], the approach is graph-theoretic in nature: The problem of bounding from above the transient is reduced to the study of long walks in a specific weighted graph, and the key technical point is then to design sophisticated walk transformers that do not modify the weights of some walks. On the contrary, our approach here is more algebraic in the sense that instead of walk transformers, we use a convolution lemma borrowed from [18]. Based on simpler and more direct arguments than the classical one, we thus obtain an easy proof that linear systems are eventually periodic, and a new upper bound on the length of the initial transient phase which is in $O(N^4)$ for linear systems of dimension N with integer matrices.

Interestingly, the proof simplification does not damage the resulting bound: Our new transience bound is of the same order as the repetitive and explorative bounds, and so is incomparable with Hartmann and Arguelles' bound, and better than Even and Rajsbaum's bound. Moreover, like the repetitive and explorative bounds by Charron-Bost et al., it is linear in the size of the system in various classes of linear systems whereas both Even and Rajsbaum's bound and Hartmann and Arguelles' bounds are both intrinsically at least quadratic.

Finally, we demonstrate how our general transience bound enables the performance analysis of a large variety of distributed systems. First, we immediately derive a general transience bound for a large class of synchronizers, and we quantify how our synchronizer bound is better than that of Even and Rajsbaum [13] in the specific case of integer delays. From this synchronizer example, we show that our transience bound is asymptotically tight. Our result also applies to the analysis of the performance of distributed routers and schedulers based on the link-reversal algorithms: We obtain an $O(N^3)$ transience bound, improving the $O(N^4)$ bound established by Malka and Rajsbaum [16], and an $O(N)$ bound for such routers and schedulers when running in trees. For link-reversal routers, eventual periodicity actually corresponds to termination, and an $O(N^2)$ bound on time complexity [6] directly follows from our transience bound.

2 Preliminaries

This section introduces max-plus linear systems, their graph interpretation, and discusses general properties of eventually periodic sequences.

As observed in [16], the behavior of distributed systems like network synchronizers and distributed link reversal algorithms can be described by a sequence

$(x(n))_{n\geqslant 0}$ of N-dimensional vectors that satisfy a recurrence relation of the following form:

$$\forall i \in [N],\ \forall n \geqslant 0, \quad x_i(n+1) = \max_{j\in \mathbf{N}_i} \big(x_j(n) + a_{ij}\big) \tag{1}$$

where the a_{ij}'s are reals, and $\mathbf{N}_i$ is a subset of $[N] = \{1, \dots, N\}$. Trivially, a system of this form corresponds to the recurrence relation

$$\forall i \in [N],\ \forall n \geqslant 0, \quad x_i(n+1) = \max_{j\in [N]} \big(x_j(n) + A_{ij}\big) \tag{2}$$

with $A_{ij} = a_{ij}$ if j in $\mathbf{N}_i$ and $-\infty$ otherwise. Recurrence (2) turns out to be linear system in the max-plus algebra, i.e., when replacing '+' by 'max' and '$\times$' by '+', over the set $\overline{\mathbb{R}} = \mathbb{R} \cup \{-\infty\}$. Let us recall that the *max-plus product* of two matrices A and B is defined by $(AB)_{ij} = \max_k(A_{ik} + B_{kj})$. With this definition, (2) is equivalent to

$$\forall n \geqslant 0, \quad x(n+1) = A\,x(n)\ . \tag{3}$$

Given an initial vector $v \in \mathbb{R}^N$, recurrence (3) admits a unique solution given by

$$\forall n \geqslant 0, \quad x(n) = A^n v\ . \tag{4}$$

Thus the analysis of distributed systems whose behaviors are controlled by a recurrence relation of the form (1) leads to study the systems (4).

To every matrix A naturally corresponds a weighted digraph which we denote by $G(A)$: Its node set is $[N]$ and there exists an edge from node i to node j with weight A_{ij} if and only if A_{ij} is finite. Matrix A is *irreducible* if $G(A)$ is strongly connected. A *walk* consists of a sequence of successive edges, as well as a start and an end node. We denote by $\ell(W)$ the length of walk W. A *path* is a walk without node repetition. A *closed* walk is a walk whose start node is equal to its end node. A *cycle* is a closed walk in which the only node repetition is the start and end node. The *circumference* $\Gamma(G)$ of graph G is the maximum cycle length in G.[1] We write $\mathcal{W}(i,j)$ for the set of walks from node i to node j, and $\mathcal{W}(i\!\rightarrow)$ for the set of walks starting from i. Further, $\mathcal{W}^n(i,j)$ and $\mathcal{W}^n(i\!\rightarrow)$ denote the set of walks in $\mathcal{W}(i,j)$ and $\mathcal{W}(i\!\rightarrow)$ of length n, respectively.

We write $p(W)$ for the edge weight of walk W, i.e., the sum of the weights of its edges. Given a vector v, the edge-node weight $p_v(W)$ is equal to $p(W) + v_j$, where j is W's end node. These definitions yield a correspondence between the matrix power A^n as well as the vector $A^n v$, and the maximum weight of walks of length n in $G(A)$, namely:

$$\begin{aligned} (A^n)_{ij} &= \max\{p(W) \colon W \in \mathcal{W}^n(i,j)\} \\ (A^n v)_i &= \max\{p_v(W) \colon W \in \mathcal{W}^n(i\!\rightarrow)\} \end{aligned} \tag{5}$$

[1] The computation of the circumference of a graph is NP-hard in the number of nodes. However, the circumference is always upper bounded by the number of nodes.

In the following, we write A^n_{ij} instead of $(A^n)_{ij}$, as no confusion can arise. A closed walk is *critical* if its average edge weight is maximum, i.e., if it is equal to the *rate of* A defined by

$$\lambda(A) = \max\{p(C)/\ell(C) \colon C \text{ is a closed walk in } G(A)\} \quad , \tag{6}$$

and simply denoted λ when no confusion can arise. The set of critical closed walks induces the *critical subgraph* G_c whose nodes are called *critical nodes*. It is well-known (e.g., see [15]) that every closed walk in G_c is necessarily critical.

Let p be a strictly positive integer and ϱ a real number. A sequence $f : \mathbb{N} \to \overline{\mathbb{R}}$ is *eventually periodic with period* p *and ratio* ϱ if there exists an integer T such that:

$$\forall n \geqslant T \ : \ f(n+p) = f(n) + p \cdot \varrho \tag{7}$$

Obviously if q is any multiple of p, then f is eventually periodic with period q and ratio ϱ. Hence, there always exists a common period of two eventually periodic sequences.

For every period p, there exists a unique minimal transient T_p that satisfies Equation (7). The next lemma shows that these minimal transients do, in fact, not depend on p. We will henceforth call this common value *the* transient of f. Due to limitation of space, its proof is postponed to the appendix.

Lemma 1. *Let* $f : \mathbb{N} \to \overline{\mathbb{R}}$ *be eventually periodic. Let* p *and* q *be two periods of* f *with respective minimal transients* T_p *and* T_q*. Then* $T_p = T_q$*.*

The next two elementary lemmas both play an important role in our approach: They state transience bounds for a sequence obtained by the element-wise composition of two eventually periodic sequences f and g with common ratio in terms of the transients of f and g.

Lemma 2. *Let* $f, g : \mathbb{N} \to \overline{\mathbb{R}}$ *be eventually periodic with common ratio* ϱ *and respective transients* T_f *and* T_g*. Then the sequence* $\max\{f, g\}$ *is eventually periodic with ratio* ϱ *and transient at most* $\max\{T_f, T_g\}$*.*

In analogy to classical convolution, one defines the *max-plus convolution* $f \otimes g$ of two sequences f and g as

$$(f \otimes g)(n) = \max_{n_1 + n_2 = n} \big(f(n_1) + g(n_2)\big) \ . \tag{8}$$

Nachtigall [18] then proved a bound on the transient of the max-plus convolution of two eventually periodic sequences f and g with the same ratio.

Lemma 3 ([18, Lemma 6.1]). *Let* $f, g : \mathbb{N} \to \overline{\mathbb{R}}$ *be eventually periodic with common ratio* ϱ *and respective transients* T_f *and* T_g*. Then the convolution* $f \otimes g$ *is eventually periodic with ratio* ϱ *and transient at most equal to* $T_f + T_g + p - 1$ *if* p *denotes a common period to* f *and* g*.*

The notion of eventual periodicity naturally extends to vectors: a sequence of vectors $(x(n))_{n \geqslant 0}$ is *eventually periodic with period* p *and ratio* ϱ if each sequence $(x_i(n))_{n \geqslant 0}$ is eventually periodic with period p and ratio ϱ. Its *transient* is the maximum transient of the $(x_i(n))_{n \geqslant 0}$'s.

3 Transience Bound

The fundamental theorem in max-plus algebra [11] shows that any linear system as defined in (4) is eventually periodic with ratio equal to the rate λ of A if A is irreducible. In this section, we establish an effective upper bound on the transient of these systems. In this way, we also give an alternative proof of the fundamental theorem that is simpler and more direct than the classical one.

3.1 Proof Strategy

By definition of the max-plus matrix product, the i-th component of $x(n)$ is equal to

$$x_i(n) = \max_{j \in [N]} \left(A^n_{ij} + v_j \right) \ .$$

In view of Lemma 2, our strategy will thus consist in showing that each sequence $(A^n_{ij} + v_j)_{n \geqslant 0}$ is eventually periodic. In the case either i or j is a critical node, the question is solved by Nachtigall [18, Lemma 3.2] and Even and Rajsbaum [13, Theorem 6]: they independently showed that in this case, the sequence $(A^n_{ij})_{n \geqslant 0}$ is eventually periodic, and gave effective upper bounds on the transients. For simplicity, every sequence $(A^n_{ij})_{n \geqslant 0}$ will be denoted by A_{ij} in the following.

Lemma 4. *Let A be an irreducible $N \times N$ matrix, and λ the rate of A. If k is a node of a critical cycle C of length ℓ, then both sequences A_{ik} and A_{ki} are eventually periodic with period ℓ, ratio λ, and transient at most $\ell \cdot (N-1)$.*

In the appendix, we give a proof of this lemma that essentially follows the one by Nachtigall [18].

In the general case where neither i nor j is critical, we observe that for any pair of nonnegative integers (n_1, n_2) such that $n = n_1 + n_2$, we have

$$A^n_{ij} = \max_{k \in [N]} \left(A^{n_1}_{ik} + A^{n_2}_{kj} \right) \ .$$

It trivially follows that

$$A^n_{ij} = \max_{n_1+n_2=n} \left(\max_{k \in [N]} \left(A^{n_1}_{ik} + A^{n_2}_{kj} \right) \right) = \max_{k \in [N]} \left(\max_{n_1+n_2=n} \left(A^{n_1}_{ik} + A^{n_2}_{kj} \right) \right) \ ,$$

and so

$$A^n_{ij} = \max_{k \in [N]} \left((A_{ik} \otimes A_{kj})(n) \right) \ .$$

So we can write

$$x_i(n) = \max_{j \in [N]} \max_{k \in [N]} \left((A_{ik} \otimes A_{kj})(n) + v_j \right) \ . \tag{9}$$

If k is a critical node which lies on a critical cycle of length ℓ_k, then Lemmas 3 and 4 imply that each sequence $A_{ik} \otimes A_{kj}$ is eventually periodic, with period ℓ_k, ratio λ, and a transient that is bounded from above by $2\ell_k \cdot (N-1) + \ell_k - 1$.

Thus, if we could show that the range of the second maximum in Equation (9) can be restricted to the critical nodes, i.e.,

$$x_i(n) = \max_{j \in [N]} \max_{k \in G_c} \big((A_{ik} \otimes A_{kj})(n) + v_j\big) \quad , \tag{10}$$

we would readily obtain a bound on the transience of the sequence x_i from Lemma 2. We will, indeed, show in Section 3.2 that Equation (10) holds for n at least equal to some integer n_c. An effective upper bound on n_c finally allows us to conclude with a bound on the transience of x_i.

The following lemma provides a condition ensuring that Equation (10) holds.

Lemma 5. *If there exists a walk of maximum p_v-weight in $\mathcal{W}^n(i \to)$ that contains a critical node, then*

$$x_i(n) = \max_{j \in [N]} \max_{k \in G_c} \big((A_{ik} \otimes A_{kj})(n) + v_j\big) \quad .$$

Proof. From (9), we trivially derive that

$$x_i(n) \geqslant \max_{j \in [N]} \max_{k \in G_c} \big((A_{ik} \otimes A_{kj})(n) + v_j\big) \quad .$$

Conversely, let W be a walk of maximum p_v-weight in $\mathcal{W}^n(i \to)$ that contains a critical node k_0, and let j_0 be the end node of W. By (5), we have $x_i(n) = p(W) + v_{j_0}$. Let us decompose W into $W = W_1 \cdot W_2$ such that W_1 ends at k_0. Setting $\ell_1 = \ell(W_2)$ and $\ell_2 = \ell(W_2)$, we get

$$x_i(n) = p(W_1) + p(W_2) + v_{j_0} \leqslant A^{\ell_1}_{ik_0} + A^{\ell_2}_{k_0 j_0} + v_{j_0} \quad .$$

Therefore,

$$x_i(n) \leqslant \max_{n_1 + n_2 = n} \big(A^{n_1}_{ik_0} + A^{n_2}_{k_0 j_0} + v_{j_0}\big) = (A_{ik_0} \otimes A_{k_0 j_0})(n) + v_{j_0} \quad .$$

It follows that

$$x_i(n) \leqslant \max_{j \in [N]} \max_{k \in G_c} \big((A_{ik} \otimes A_{kj})(n) + v_j\big) \quad ,$$

which concludes the proof.

3.2 Critical Bound

We next show that there always exists an integer n_c such that the condition in Lemma 5 holds for all $n \geqslant n_c$, and we give an effective upper bound B_c on n_c which we call the *critical bound.* For that, we compare a maximum p_v-weight walk $\hat{W}$ in $\mathcal{W}^n(i \to)$ that does not visit a critical node with a walk W constructed from $\hat{W}$ such that it "pumps" its weight in a critical cycle as often as possible, but still has the same length n as $\hat{W}$. Since $\hat{W}$ may not use critical cycles, it will, when compared to W, on average lose weight in each of its cycles. We arrive at a bound B_c, such that, if $\hat{W}$ has length $n \geqslant B_c$ then $\hat{W}$ has weight less than W and thus cannot be a maximum p_v-weight walk in $\mathcal{W}^n(i \!\rightarrow)$.

Let $\|v\| = \max_{i \in [N]}(v_i) - \min_{i \in [N]}(v_i)$. Denote by δ the minimal finite entry of A and by $\varDelta_{nc}$ the maximum A_{ij} where both i and j are non-critical nodes. Further let

$$\lambda_{nc} = \max\left\{p(C)/\ell(C)\colon C \text{ is a closed walk with no critical node in } G(A)\right\} \ .$$

Lemma 6 (Critical Bound). *For all $i \in [N]$ and $n \geqslant 0$, each walk with maximum p_v-weight in $\mathcal{W}^n(i \!\rightarrow)$ contains a critical node if*

$$n \geqslant B_c = \max\left\{N \ , \ \frac{\|v\| + (\varDelta_{nc} - \delta)\,(N-1)}{\lambda - \lambda_{nc}}\right\} \ .$$

Proof. We first reduce the case of a graph G with arbitrary λ to the case where $\lambda = 0$: Let G_0 be the graph constructed from G by subtracting λ from all of G's edge weights. Clearly then G_0 has $\lambda = 0$ and the (unweighted) critical subgraphs of G and G_0 are the same. Further W is a walk with maximum p_v-weight in G if and only if it is a walk with maximum p_v-weight in G_0. The graph parameters δ, $\varDelta_{nc}$, and λ_{nc} of G_0 are obtained by subtracting G's λ from the respective graph parameters of G. The lemma's bound B_c thus is the same for graphs G and G_0, and we may safely assume that $\lambda = 0$.

If $\lambda_{nc} = -\infty$, then every nonempty cycle contains a critical node. Because every walk of length greater or equal to N necessarily contains a cycle as a subwalk and because $B_c \geqslant N$, in particular every walk with maximum p_v-weight in $\mathcal{W}^n(i \!\rightarrow)$ contains a critical node if $n \geqslant B_c$ and $\lambda_{nc} = -\infty$.

We now consider the case $\lambda_{nc} \neq -\infty$. We proceed by contradiction: Suppose that there exists an integer n such that $n \geqslant B_c$, a node i and a walk of maximum p_v-weight in $\mathcal{W}^n(i \!\rightarrow)$ with non-critical nodes only; let $\hat{W}$ be such a walk. Let W_0 be the *acyclic part* of $\hat{W}$, defined in the following manner: Starting at $\hat{W}$, we repeatedly remove nonempty subcycles from the walk until we arrive at a path. In general there are several possible choices of which subcycles to remove, but we fix some global choice function to make the construction of W_0 deterministic.

Next choose a critical node k, and then a prefix W_c of W_0, such that the distance between k and the end node of W_c is minimal. Let W_2 be a path of minimal length from the end node of W_c to k. Let W_3 be the walk such that $W_0 = W_c \cdot W_3$. Further let C be a critical cycle starting at k.

We distinguish two cases for n, namely (a) $n \geqslant \ell(W_c) + \ell(W_2)$, and (b) $n < \ell(W_c) + \ell(W_2)$.

Case A: Let $m \in \mathbb{N}$ be the quotient in the Euclidean division of $n - \ell(W_c) - \ell(W_2)$ by $\ell(C)$, and choose W_1 to be a prefix of C of length $n - \big(\ell(W_c) + \ell(W_2) + m \cdot \ell(C)\big)$ (see Figure 1). Clearly W_1 starts at k. If we set $W = W_c \cdot W_2 \cdot C^m \cdot W_1$, we get $\ell(W) = n$ and

$$p_v(W) \geqslant \min_{1 \leqslant j \leqslant N}(v_j) + p(W_c) + p(W_2) + p(W_1) \tag{11}$$

since we assume $\lambda = 0$.

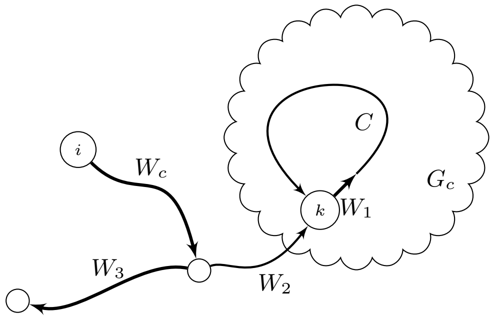


Fig. 1. Walk W in proof of Lemma 6

For the p_v-weight of $\hat{W}$, we have

$$\begin{aligned} p_v(\hat{W}) &\leqslant p_v(W_0) + \lambda_{nc}\cdot\big(\ell(\hat{W}) - \ell(W_0)\big) \\ &\leqslant \max_{1\leqslant j\leqslant N}(v_j) + p(W_0) + \lambda_{nc}\cdot\big(\ell(\hat{W}) - \ell(W_0)\big) \end{aligned} \tag{12}$$

By assumption $p_v(\hat{W}) \geqslant p_v(W)$, and from (11), (12), and $\lambda_{nc} < 0$ we therefore obtain

$$\begin{aligned} \ell(\hat{W}) &\leqslant \frac{\|v\| + p(W_3) - p(W_1) - p(W_2)}{-\lambda_{nc}} + \ell(W_0) \\ &\leqslant \frac{\|v\| + \varDelta_{nc}\,\ell(W_3) - \delta\left(\ell(W_1) + \ell(W_2)\right)}{-\lambda_{nc}} + \ell(W_0) \end{aligned} \tag{13}$$

Denote by N_{nc} the number of non-critical nodes. The following three inequalities trivially hold: $\ell(W_3) \leqslant N_{nc} - 1$, $\lambda_{nc} \geqslant \delta$, and $\ell(W_1) < N - N_{nc}$. Since there is at least one critical node, we have $\ell(W_3) < N - 1$. Moreover from the minimality constraint for the length of W_2 follows that $\ell(W_2) + \ell(W_0) \leqslant N_{nc}$. Thereby

$$\ell(\hat{W}) < \frac{\|v\| + (\varDelta_{nc} - \delta)\,(N-1)}{-\lambda_{nc}} \ , \tag{14}$$

a contradiction to $n \geqslant B_c$. The lemma follows for case A.

Case B: In this case $\ell(W_c) \leqslant n < \ell(W_c) + \ell(W_2)$, and we set $W = W_c \cdot W_2'$, where W_2' is a prefix of W_2, such that $\ell(W) = n$. Hence,

$$p_v(W) \geqslant \min_{1\leqslant j\leqslant N}(v_j) + p(W_c) + p(W_2') \ . \tag{15}$$

We again obtain (12). By assumption $p_v(\hat{W}) \geqslant p_v(W)$, and by similar arguments as in case A we derive

$$\ell(\hat{W}) \leqslant \frac{\|v\| + p(W_3) - p(W_2')}{-\lambda_{nc}} + \ell(W_0)$$

and since W_2' is a prefix of W_2 with $\ell(W_2') < \ell(W_2)$,

$$\ell(\hat{W}) < \frac{\|v\| + \mathit{\Delta}_{nc}\,\ell(W_3) - \delta\,\ell(W_2)}{-\lambda_{nc}} + \ell(W_0) \ ,$$

which is less or equal to the bound obtained in (13) of case A. By similar arguments as in case A, the lemma follows in case B.

3.3 Transience Bound

By combination of the above lemmas, we finally obtain the following transience bound.

Theorem 1. *Let A be an irreducible $N \times N$ matrix and let v be a vector in $\mathbb{R}^N$. Then the sequence of vectors $x(n) = A^n v$ is eventually periodic with ratio λ, and its transient is at most*

$$\max\left\{\frac{\|v\| + (\mathit{\Delta}_{nc} - \delta)\,(N-1)}{\lambda - \lambda_{nc}} \ , \ \mathit{\Gamma}_c \cdot (2N-1) - 1\right\} \ ,$$

where $\mathit{\Gamma}_c$ is the circumference of the critical graph G_c.

Proof. From Lemmas 5 and 6, we know that each i-th component of $x(n)$ equals

$$x_i(n) = \max_{j \in [N]} \max_{k \in G_c} \big((A_{ik} \otimes A_{kj})(n) + v_j\big)$$

when $n \geqslant B_c$. For each critical node k, let ℓ_k denote the length of a critical cycle containing k. By Lemmas 1 and 4, we obtain that all sequences A_{ik} and A_{kj} are eventually periodic, with period ℓ_k, ratio λ, and a transient less or equal to $\mathit{\Gamma}_c \cdot (N-1)$ because $\ell_k \leqslant \mathit{\Gamma}_c$. Lemma 3 shows that the sequence $\big((A_{ik} \otimes A_{kj})(n) + v_j\big)_{n \geqslant 0}$ is eventually periodic, with ratio λ, and a transient less or equal to $2\mathit{\Gamma}_c \cdot (N-1) + \mathit{\Gamma}_c - 1$. By Lemma 2, the same property holds for the sequence $\big(\max_{j \in [N]} \max_{k \in G_c}((A_{ik} \otimes A_{kj})(n) + v_j\big)_{n \geqslant 0}$. This proves that each sequence x_i is eventually periodic, with ratio λ, and a transient at most equal to

$$\max\big\{B_c \, , \, \mathit{\Gamma}_c \cdot (2N-1) - 1\big\} \ . \tag{16}$$

This concludes the proof.

In the case each finite A's entry is an integer, the term $\lambda - \lambda_{nc}$ cannot become arbitrarily small: If N_{nc} denotes the number of non-critical nodes, then

$$\frac{1}{\lambda - \lambda_{nc}} \leqslant (N - N_{nc}) \cdot N_{nc} \leqslant \frac{N^2}{4} \ . \tag{17}$$

From that, we immediately derive that our critical bound, and so our transient bound for linear systems with integer matrices is in $O(N^3)$.

3.4 Comparison with Previous Bounds

Hartmann and Arguelles [17] stated the following bound on the transient of the linear system:

$$B_{\mathrm{HA}} = \max\left\{ \frac{\|v\| + N \cdot (\varDelta - \delta)}{\lambda - \lambda^0} \ , \ 2N^2 \right\} \tag{18}$$

Here, λ^0 is a parameter of the max-balanced reweighted graph [19] of G and $\varDelta$ is the maximum edge weight in G.

The first term in (18) corresponds to our critical bound, but is incomparable with it in general. The second term in (18), namely $2N^2$, is always greater than the second term in the maximum in our bound, $\varGamma_c \cdot (2N-1) - 1$, because of the trivial estimate $\varGamma_c \leqslant N$. As demonstrated in the next section, a major difference to our bound is that (18) is inherently quadratic in N and hence prohibits a subquadratic analysis of the transient.

Charron-Bost et al. [8] gave two transience bounds for linear systems—the repetitive and the explorative bound—which both allow for a subquadratic analysis of the transient. The repetitive bound is equal to $\max\{B_c\, , \, \hat{g} \cdot (2N-1) - 1\}$ where $\hat{g}$ is the maximum girth of strongly connected components of the critical graph G_c. It is always smaller than our bound in Theorem 1 because $\hat{g} \leqslant \varGamma_c$. The explorative bound is equal to $\max\{B_c\, , \, \hat{\gamma} \cdot (2N-1) - 1 + \hat{ep}\}$ where $\hat{\gamma}$ is the maximum cyclicity and $\hat{ep}$ the maximum *exploration penalty* of strongly connected components of the critical graph G_c. The exploration penalty of a graph is the transient of the sequence of matrix powers of its unweighted adjacency matrix. The explorative bound is, in general, incomparable with our bound in Theorem 1.

4 Applications

As stated in the Introduction, various distributed algorithms correspond to linear max-plus systems. In this section, we explain how our transience bound immediately applies to such distributed algorithms, and allows us to analyze their performances.

4.1 Synchronizers

Even and Rajsbaum [13] presented a transience bound for a type of network synchronizers in a system with constant integer communication delays. They considered a variant of the α-synchronizer [3] in a centrally clocked distributed system of N processes that communicate by message passing over a strongly connected network graph G. Each link has constant transmission delay, specified in terms of central clock ticks. Processes execute the α-synchronizer after an initial boot-up phase: After receiving round n messages from all its neighbors, a process proceeds to round $n+1$ and broadcasts its round $n+1$ message. Denote

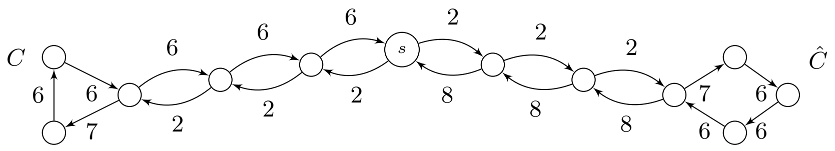


Fig. 2. Graph $H_{3,2}$

by $t(n)$ the vector such that $t_i(n)$ is the clock tick at which process i broadcasts its round n message. Even and Rajsbaum showed, by a subtle graph-theoretic approach, that the synchronizer becomes periodic by time $B_{\mathrm{ER}} = l_0 + N + 2N^2$, where l_0 is always greater or equal to our critical bound B_c.

One can easily establish that t is in fact a linear max-plus system: $t(n+1) = At(n)$ where A is the adjacency matrix of the network delay graph. Our bounds hence directly apply and we obtain a bound on the transient of $(t(n))_{n \geqslant 0}$ strictly better than B_{ER}.

As an example, let us consider the graph family $H_{\ell,c}$, $\ell \geqslant 2$ and $c \geqslant 1$, introduced by Even and Rajsbaum [13] to study the asymptotic behavior of B_{ER}: Let $\hat{C}$ and C be cycles of length ℓ and $\ell+1$ respectively, with edge weights $3c$, except for one link per cycle with weight $3c+1$. There exists for both $\hat{C}$ and C a path of length ℓ to a distinct node s, and an antiparallel path back. Hereby the edges in the path from s to C and from s to $\hat{C}$ have weight c, the edges in the path from C to s have weight $3c$, and from $\hat{C}$ to s, $4c$. As an example, $H_{3,2}$ is depicted in Figure 2.

Observing that the nodes of $\hat{C}$ are the critical nodes, $\Delta = 4c$, $\delta = c$, $N = 4\ell$, $\lambda = 3c + 1/\ell$, and $l_0 = 112c\ell^3 - 16\ell^3 - 12c\ell^2 + 4\ell - 1$, Even and Rajsbaum's bound is

$$(112c - 16)\ell^3 + (32 - 12c)\ell^2 + 8\ell - 1 \ .$$

For $\Delta_{nc} = \Delta$ and $\lambda_{nc} = 3c + 1/(\ell+1)$, we obtain for the critical bound $B_c = 3c\ell(\ell+1)(N-1)+1 = 12c\ell^3+9c\ell^2-3c\ell+1$. Since the circumference of the critical subgraph is $\Gamma_c = \ell$, we may bound the transient of $(t(n))_{n\geqslant 0}$ with Theorem 1 by

$$\max\{B_c, 2\ell N - \ell - 1\} = \max\{B_c, 8\ell^2 - \ell - 1\} = 12c\ell^3 + 9c\ell^2 - 3c\ell + 1 \ .$$

Since Even and Rajsbaum have shown that the transient for the graph family $H_{\ell,c}$, $\ell \geqslant 2$, is in $\Omega(\ell^3) = \Omega(N^3)$, this proves that our transience bound in $O(N^3)$ for integer matrices is asymptotically tight.

4.2 Full Reversal Routing and Scheduling

Link reversal is a versatile algorithm design paradigm, which was, in particular, successfully applied to routing [14] and scheduling [4]. Charron-Bost et al. [10] showed that the analysis of a general class of link reversal algorithms can be reduced to the analysis of Full Reversal, a particularly simple algorithm on directed graphs.

The Full Reversal algorithm comprises a single rule: Each sink reverses all its (incoming) edges. Given a weakly connected initial graph G_0 without antiparallel edges, we consider a *greedy* execution of Full Reversal as a sequence $(G_t)_{t \geqslant 0}$ of graphs, where G_{t+1} is obtained from G_t by reversing the edges of *all* sinks in G_t. As no two sinks in G_t can be adjacent, G_{t+1} is well-defined. For each $t \geqslant 0$ we define the *work vector* $W(t)$ by setting $W_i(t)$ to the number of reversals of node i until iteration t, i.e., the number of times node i is a sink in the execution prefix $G_0, \dots, G_{t-1}$.

Charron-Bost et al. [9, Corollary 2] have shown that the sequence of work vectors can be described as a max-plus linear system. More precisely, we have $W(0) = 0$ and $-W(t+1) = (-A) \cdot (-W(t))$, where $A_{i,j} = 1$ and $A_{j,i} = 0$ if (i,j) is an edge of G_0; otherwise $A_{i,j} = +\infty$. Observe that $-A$ is a matrix with integer weights, $\Delta_{nc} \in \{0, -1\}$ and $\delta = -1$. Our results, in particular, imply the eventual periodicity of the Full Reversal algorithm.

Full Reversal Routing. In the routing case, the initial graph G_0 contains a nonempty set of *destination nodes*, which are characterized by having a self-loop. The initial graph without these self-loops is required to be weakly connected and acyclic [9,14]. It was shown that for such initial graphs, the execution terminates (eventually all G_t are equal), and after termination, the graph is destination-oriented, i.e., every node has a walk to some destination node. We now show how the previously known results that the termination time of Full Reversal routing is quadratic in general [6] and linear in trees [9] directly follows from Theorem 1.

The set of critical nodes is equal to the set of destination nodes and each strongly critical component of G_c consists of a single node. Hence $\lambda = 0$ and $\lambda_{nc} \leqslant -1/N_{nc} \leqslant -1/(N-1)$, i.e., $(N-1)^2$ is an upper bound on the critical bound. Since $\Gamma_c = 1$, we obtain from Theorem 1 that the termination time is at most $(N-1)^2$, which improves the asymptotic quadratic bound given by Busch and Tirthapura [6].

If the undirected support of initial graph G_0 without the self-loops at the destination nodes is a *tree*, we can use our transience bound to give a new proof that the termination time of Full Reversal routing is linear in N [9, Corollary 5]. Indeed, in that particular case either $\lambda_{nc} = -1/2$ or $\lambda_{nc} = -\infty$, which both give a critical bound at most equal to $2(N-1)$. From Theorem 1 we obtain the linear transience bound of $2(N-1)$, whereas Hartmann and Arguelles' bound is $2N^2$.

Full Reversal Scheduling. When using the Full Reversal algorithm for scheduling, the undirected support of the weakly connected initial graph G_0 is interpreted as a conflict graph: nodes model processes and an edge between two processes signifies the existence of a shared resource whose access is mutually exclusive. The direction of an edge signifies which process is allowed to use the resource next. A process waits until it is allowed to use all its resources—that is, it waits until it is a sink—and then performs a step, that is, reverses all edges to release its resources. To guarantee liveness, the initial graph G_0 is required to be acyclic.

Contrary to the routing case, critical components have at least two nodes, because there are no self-loops. By using (17), our critical bound is upper-bounded by $N^2(N-1)/4+1$, which shows that the transient for Full Reversal scheduling is at most cubic in the number N of processes. Malka and Rajsbaum [16, Theorem 6.4] proved by reduction to Timed Marked Graphs that the transient is at most in the order of $O(N^4)$. Thus, our bound allows to improve this asymptotic result by an order of N.

In the case of Full Reversal scheduling on *trees* we even obtain a linear bound in N: In this case it holds that $\lambda = -1/2$, $\lambda_{nc} = -\infty$, and so our critical bound is 1. Further, $G_c = G$ and $\mathit{\Gamma}_c = 2$. Theorem 1 thus implies a linear upper bound on the transient of Full Reversal scheduling on trees of $4N-3$, which establishes a new result for these distributed schedulers. By contrast, Hartmann and Arguelles again can only obtain the quadratic bound of $2N^2$.

References

1. Akian, M., Gaubert, S., Walsh, C.: Discrete Max-Plus Spectral Theory. In: Litvinov, G.L., Maslov, V.P. (eds.) Idempotent Mathematics and Mathematical Physics, pp. 53–78. AMS, Providence (2005)
2. Attiya, H., Gramoli, V., Milani, A.: A Provably Starvation-Free Distributed Directory Protocol. In: Dolev, S., Cobb, J., Fischer, M., Yung, M. (eds.) SSS 2010. LNCS, vol. 6366, pp. 405–419. Springer, Heidelberg (2010)
3. Awerbuch, B.: Complexity of Network Synchronization. J. ACM 32, 804–823 (1985)
4. Barbosa, V.C., Gafni, E.: Concurrency in Heavily Loaded Neighborhood-Constrained Systems. ACM T. Progr. Lang. Sys. 11, 562–584 (1989)
5. Bouillard, A., Gaujal, B.: Coupling Time of a (max,plus) Matrix. In: Workshop on Max-Plus Algebra at the 1st IFAC Symposium on System Structure and Control. Elsevier, Amsterdam (2001)
6. Busch, C., Tirthapura, S.: Analysis of Link Reversal Routing Algorithms. SIAM J. Comput. 35, 305–326 (2005)
7. Chandy, K.M., Misra, J.: The Drinking Philosophers Problem. ACM T. Progr. Lang. Sys. 6, 632–646 (1984)
8. Charron-Bost, B., Függer, M., Nowak, T.: New Transience Bounds for Long Walks (2012) `arXiv:1209.3342 [cs.DM]`
9. Charron-Bost, B., Függer, M., Welch, J.L., Widder, J.: Full Reversal Routing as a Linear Dynamical System. In: Kosowski, A., Yamashita, M. (eds.) SIROCCO 2011. LNCS, vol. 6796, pp. 101–112. Springer, Heidelberg (2011)
10. Charron-Bost, B., Függer, M., Welch, J.L., Widder, J.: Partial is Full. In: Kosowski, A., Yamashita, M. (eds.) SIROCCO 2011. LNCS, vol. 6796, pp. 113–124. Springer, Heidelberg (2011)
11. Cohen, G., Dubois, D., Quadrat, J.-P., Viot, M.: Analyse du comportement périodique de systèmes de production par la théorie des dioïdes. INRIA Research Report 191 (1983)
12. Dubois, D., Stecke, K.E.: Dynamic Analysis of Repetitive Decision-Free Discrete Event Processes: The Algebra of Timed Marked Graphs and Algorithmic Issues. Ann. Oper. Res. 26, 151–193 (1990)
13. Even, S., Rajsbaum, S.: The Use of a Synchronizer Yields Maximum Computation Rate in Distributed Systems. Theor. Comput. Syst. 30, 447–474 (1997)

14. Gafni, E.M., Bertsekas, D.P.: Asymptotic Optimality of Shortest Path Routing Algorithms. IEEE T. Inform. Theory. 33, 83–90 (1987)
15. Heidergott, B., Olsder, G.J., van der Woude, J.: Max Plus at Work. Princeton University Press, Princeton (2006)
16. Malka, Y., Rajsbaum, S.: Analysis of Distributed Algorithms Based on Recurrence Relations. In: Toueg, S., Spirakis, P.G., Kirousis, L. (eds.) WDAG 1991. LNCS, vol. 579, pp. 242–253. Springer, Heidelberg (1992)
17. Hartmann, M., Arguelles, C.: Transience Bounds for Long Walks. Math. Oper. Res. 24, 414–439 (1999)
18. Nachtigall, K.: Powers of Matrices over an Extremal Algebra with Applications to Periodic Graphs. Math. Method. Oper. Res. 46, 87–102 (1997)
19. Schneider, H., Schneider, M.H.: Max-Balancing Weighted Directed Graphs and Matrix Scaling. Math. Oper. Res. 16, 208–222 (1991)
20. Soto y Koelemeijer, G.: On the Behaviour of Classes of Min-Max-Plus Systems. PhD Thesis, TU Delft (2003)
21. Tirthapura, S., Herlihy, M.: Self-Stabilizing Distributed Queuing. IEEE T. Parall. Distr. 17, 646–655 (2006)
22. Welch, J.L., Walter, J.E.: Link Reversal Algorithms. Morgan & Claypool, San Rafael (2012)

Back in Time Petri Nets*

Thomas Chatain[1] and Claude Jard[2]

[1] LSV, ENS Cachan, INRIA, CNRS, France
[2] LINA, Université de Nantes, France

Abstract. The time progress assumption is at the core of the semantics of real-time formalisms. It is also the major obstacle to the development of partial-order techniques for real-time distributed systems since the events are ordered both by causality and by their occurrence in time. Anyway, extended free choice safe time Petri nets (TPNs) were already identified as a class where partial order semantics behaves well. We show that, for this class, the time progress assumption can even be dropped (time may go back in case of concurrency), which establishes a nice relation between partial-order semantics and time progress assumption.

1 Introduction

Time Petri nets [10] are now a well established model to describe dynamic systems present in many areas. They are restricted to control aspects, but have the following interesting features: – it is a hybrid model mixing discrete state changes with the inclusion of continuous timed constraints; – it makes explicit the causal dependency relationships between state transitions; – it defines exclusive choices (conflicts) between transitions and by complementation, transitions that can be executed independently (in concurrency).

The standard semantics of time Petri nets is sequential and defines timed words formed by a succession of discrete transitions interspersed with timed transitions representing the passage of time. The consequence is that time progresses throughout the execution of the sequence.

However, in a sequence of transitions, it is possible that transitions are in concurrency (we say they are "concurrent"). The result in general (modulo the satisfaction of time constraints) is that the ordering of these transitions is arbitrary and that the reverse order is also possible. This observation led to the definition of a concurrent semantics for Petri nets in which the behaviors are defined by partial orders (processes in the jargon of Petri nets). This new semantics is mapped with the sequential semantics by considering the total orders compatible with the partial order (the "interleavings").

In this concurrent semantics, however, the time continues to evolve sequentially in a monotonic way. But regarding concurrent transitions, this can be disputable. From the point of view of modeling, concurrent transitions can be executed in parallel, even on remote computers distributed over a communication network. In such an execution environment, it is not necessary to consider

* This work is supported by the French ANR project ImpRo (ANR-2010-BLAN-0317).

that an exact global clock is shared by the entire system. It is even realistic to consider that there is no shared clock in such a system and that time constraints must only be considered for transitions that are causally related. The original idea developed in this paper is to question the existence of a global clock. We keep of course the property that the execution of causally related transitions is totally ordered in time, but this constraint is released for execution of concurrent transitions. One way to do this is to ask **what would happen if we allowed the time to go back in the firing of a concurrent transition?**

We want that such "back in time" semantics preserves nevertheless the concurrent semantics. We show in this paper that it is possible in the case of extended free choice time Petri nets.

To do this, we begin by expressing the standard semantics by equipping tokens with their date of birth. In this new formulation, the constraint of time progression is made explicit. The elimination of this constraint defines a back in time semantics. We first show that it produces executions that are correct with respect to the standard semantics (Theorem 1). But the semantics does not exploit all the possibilities offered by the non-temporal sequencing of the execution of concurrent transitions. We thus develop the notion of completeness to suggest that the semantics must produce all linear extensions of timed processes. We propose a new back in time semantics, which is both correct and complete (Theorem 3).

Related Work. There are several variants of semantics for time Petri Nets, according the fact that the time constraints are placed on transitions or places (see the survey [5]). They are not always comparable. An important point, however, is to take into account or not the notion of urgency. The absence of urgency (the "weak"-semantics [12]) substantially alters the theoretical power of the model as well as its ability to model temporal phenomena. All these variants did not have the audacity to question the linear flow of time, which could go back in time in some cases. The closest idea to be compared is that expressed by P. Niebert [11] in a model-checking method for timed automata. In his approach, time is seen as a ordinary symbolic variable that may regress in some transitions at the Time Transition System level associated with the model. The goal is to have a more compact representation of the state space.

Time and causality does not necessarily blend well in a non deterministic model such as Petri nets. The reason is that the mixture of conflict and urgency breaks the locality of the standard firing rule of transitions. This is the main difficulty that had to be overcome in recent years to finally define the notion of unfolding of time Petri nets [7]. It is therefore understandable why the subclass of extended free-choice Petri nets is often used when applied to timed and stochastic model [2] net. In this class, in fact, we just have to look at the considered transition and transitions in conflict with it to decide its firing: decision appears to be local, while in the general case one must consider all firable transitions (those in conflict, but also those concurrent). It is therefore natural to use it when authorizing time to go back for concurrent transitions.

The rest of the paper is organized as follows. We first recall the standard semantics of time Petri nets with a special emphasis on the extended free-choice

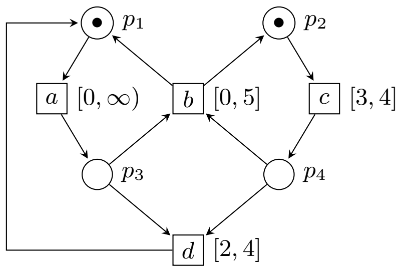


Fig. 1. An extended free choice safe time Petri net

subclass. Then we propose an alternative semantics based on tokens, which we prove to be equivalent. In section 4, we present the partial order semantics (definition of processes). Section 5 presents two new (back in time) semantics. The first is proved to be correct (consistent with the processes). The second is proved to be correct and complete (defines all the linearizations of the processes). We end with a discussion and perspectives for the general case.

2 Time Petri Nets

2.1 Definition

Definition 1 (Time Petri Net (TPN) [10]). *A* time Petri net *is a tuple* $(P, T, pre, post, efd, lfd, M_0)$ *where* P *and* T *are finite sets of* places *and* transitions *respectively, pre and post map each transition* $t \in T$ *to its (nonempty)* preset *denoted* ${}^\bullet t \overset{\text{def}}{=} pre(t) \subseteq P$ *and its (possibly empty)* postset *denoted* $t^\bullet \overset{\text{def}}{=} post(t) \subseteq P$; $efd : T \longrightarrow Q$ *and* $lfd : T \longrightarrow Q \cup \{\infty\}$ *associate the* earliest firing delay $efd(t)$ *and* latest firing delay $lfd(t)$ *with each transition* t*;* $M_0 \subseteq P$ *is the* initial marking.

A time Petri net is represented as a graph with two types of nodes: places (circles) and transitions (rectangles). The interval $[efd(t), lfd(t)]$ is written near each transition (see Figure 1).

2.2 Standard Clocks-on-Transitions Semantics

State. A *state* of a time Petri net is given by a triple (M, ν, θ), where

- $M \subseteq P$ is a *marking* (ususally reprensented graphically by tokens in the places), from which we define the set of *enabled* transitions $En(M) \overset{\text{def}}{=} \{t \in T \mid {}^\bullet t \in M\}$;
- $\nu : En(M) \to \mathbb{R}$ assigns a clock value to every enabled transition;
- $\theta \in \mathbb{R}$ is the current time. It is not required for defining the semantics, but we use it here for convenience and to ease the comparison with the alternative semantics that we give later.

A time Petri net starts in an *initial state* $(M_0, \nu_0, 0)$, which is given by the *initial marking* M_0. Initially, all the clocks are reset: for every $t \in \mathit{En}(M_0)$, $\nu_0(t) \overset{\text{def}}{=} 0$. From state (M, ν, θ), two types of actions are possible:

Time Delay. The TPN can wait until time θ' and reach state (M, ν', θ') with $\nu'(t) \overset{\text{def}}{=} \nu(t) + (\theta' - \theta)$ for every $t \in \mathit{En}(M)$, provided

- time progresses: $\theta' \geq \theta$; and
- no enabled transition overtakes its maximum delay: $\forall t \in \mathit{En}(M) \quad \nu'(t) \leq \mathit{lfd}(t)$.

The time delay is written $(M, \nu, \theta) \xrightarrow{\theta'} (M, \nu', \theta')$.

Discrete Action. Transition t can fire from state (M, ν, θ), if:

- t is enabled: $t \in \mathit{En}(M)$;
- t has reached its minimum firing delay: $\nu(t) \geq \mathit{efd}(t)$;

Firing transition t from state (M, ν, θ) leads to state (M', ν', θ), with $M' \overset{\text{def}}{=} (M \setminus {}^\bullet t) \cup t^\bullet$ and $\nu'(t) \overset{\text{def}}{=} \nu(t)$ if $t \in \mathit{En}(M \setminus {}^\bullet t)$ and $\nu'(t) \overset{\text{def}}{=} 0$ if $t \in \mathit{En}(M') \setminus \mathit{En}(M \setminus {}^\bullet t)$ (t is called "newly enabled"). We write $(M, \nu, \theta) \xrightarrow{t} (M', \nu', \theta)$.

Timed Words. When representing an execution, we often forget the information about the intermediate states and delays, and remember only the sequence $((t_1, \theta_1), \ldots, (t_n, \theta_n))$ of transitions with their firing dates. This representation is called a *timed word*. The empty timed word is denoted ϵ.

2.3 Assumptions

Boundedness. All the TPNs we consider in this article are *1-bounded*, or *safe*, i.e. in every reachable state (M, ν, θ), if a transition t can fire from (M, ν, θ), then $t^\bullet \cap (M \setminus {}^\bullet t) = \emptyset$. Hence there is never more than one token in a place. This assumption is generally done in TPNs. Considering several tokens in places requires to define complex service policies [4].

Moreover, for technical simplicity, we require that even the untimed support is safe, i.e. the TPN remains safe if one replaces all the earliest firing delays by 0 and all the latest firing delays by ∞.

Time Divergence. Finally we assume that time *diverges*: in every infinite timed word accepted by the TPN, time diverges to infinity. This assumption is very usual and comes from the intuition that no physical system can execute infinitely many action in finite time.

2.4 Extended Free Choice Time Petri Nets

In this article we will focus on a common class of TPNs which have many interesting properties, among which some concerning partial-order semantics and time.

Two transitions t and t' are in *conflict* if they share an input place. An extended free choice Petri net is a Petri net where every two transitions in conflict, have exactly the same input places.

Definition 2 (extended free choice [3,8]). *An* extended free choice *(time) Petri net is a (time) Petri net such that:*

$$\forall t, t' \in T \quad {}^{\bullet}t \cap {}^{\bullet}t' \neq \emptyset \implies {}^{\bullet}t = {}^{\bullet}t' \, .$$

It is technically convenient to focus on normalized extended free choice TPNs, where every two transitions in conflict have the same latest firing delay.

Definition 3 (Normalization of an Extended Free Choice TPN). *Given an extended free choice TPN $N = (P, T, pre, post, efd, lfd, M_0)$, for every transition t we define $lfd'(t) \stackrel{\text{def}}{=} \min(\{lfd(t') \mid t' \in T \wedge {}^{\bullet}t \cap {}^{\bullet}t' \neq \emptyset\})$. Then we define the set $T' \subseteq T$ of transitions satisfying $efd(t) \leq lfd'(t)$. The normalization of N is the TPN $N' \stackrel{\text{def}}{=} (P, T', pre_{|T'}, post_{|T'}, efd_{|T'}, lfd'_{|T'}, M_0)$.*

For the TPN of Figure 1, normalization lowers $lfd(b)$ from 5 to 4 because transition d which is in conflict with b, has a smaller latest firing delay than b.

Lemma 1. *The semantics of extended free choice safe TPNs is unchanged by normalization: the original TPN and its normalized have the same set of states, and the possible delays and discrete actions are the same (they induce exactly the same timed transition system).*

Proof. Replacing $lfd(t)$ by $lfd'(t)$ does not change the behaviour: in an extended free choice TPN, the transitions in conflict with t are enabled at the same time as t. So one of them has to fire before the minimum of their latest firing delays expires. This firing disables the others.

Finally it is obvious that transitions having $efd(t) > lfd'(t)$ can never fire. □

3 Clocks-on-Tokens Semantics

In the previous section we have defined the semantics of TPNs in the usual way. But our work is based on another way to express it, where the clocks are assigned to the tokens instead of the enabled transitions. This semantics is equivalent to the previous one (at least in the case of safe Petri nets) in the sense that both semantics are timed bisimilar.

State. A *state* of a time Petri net is now given by a triple (M, dob, θ), where $M \subseteq P$ is the *marking*, $\theta \in \mathbb{R}$ is the current time and $dob : M \longrightarrow \mathbb{R}$ associates a *date of birth* $dob(p) \in \mathbb{R}$ with each token (marked place) $p \in M$.

The marking and the current time play exactly the same role as in the clocks-on-transitions semantics, and the set of enabled transitions is defined the same way. But for every transition t enabled in a state (M, dob, θ), we define its *date of enabling* $doe(t)$ as the date of birth of the youngest token in its input places:

$$doe(t) \stackrel{\text{def}}{=} \max_{p \in {}^{\bullet}t} dob(p) \, .$$

The *initial state* is now $(M_0, dob_0, 0)$ and initially, all the tokens carry the date 0 as date of birth: for all $p \in M_0$, $dob_0(p) \overset{\text{def}}{=} 0$.

Time Delay. The TPN can wait until time θ' provided

- time progresses: $\theta' \geq \theta$;
- no enabled transition overtakes its maximum delay:
  $\forall t \in En(M) \quad \theta' \leq doe(t) + lfd(t)$.

The reached state is simply (M, dob, θ'), and we write $(M, dob, \theta) \xrightarrow{\theta'} (M, dob, \theta')$.

Discrete Action. Transition t can fire from state (M, dob, θ) if:

- t is enabled: $t \in En(M)$;
- t has reached its minimum firing delay: $\theta \geq doe(t) + efd(t)$;

Firing transition t from state (M, dob, θ) leads to state (M', dob', θ), with $M' \overset{\text{def}}{=} (M \setminus {}^\bullet t) \cup t^\bullet$ and $dob'(p) \overset{\text{def}}{=} dob(p)$ if $p \in M \setminus {}^\bullet t$ and $dob'(p) \overset{\text{def}}{=} \theta'$ if $p \in t^\bullet$ (by assumption the two cases are exclusive). We write $(M, dob, \theta) \xrightarrow{t} (M', dob', \theta)$.

3.1 Timed Bisimulation between the Two Semantics

Definition 4 (Timed Transition System). *A* timed transition system *(TTS) is a tuple $(S, s_0, \Sigma, \rightarrow)$ where S is a set of states, $s_0 \in S$ is the initial state, Σ is a finite set of actions disjoint from $\mathbb{R}_{\geq 0}$ (for the case of the semantics of a TPN, Σ is the set T of transitions), and $\rightarrow \subseteq S \times (\Sigma \cup \mathbb{R}_{\geq 0}) \times S$ is a set of edges. We write $s \xrightarrow{e} s'$ for $(s, e, s') \in \rightarrow$.*

Definition 5 (Timed bisimulation). *Let $T_1 = (S_1, s_1^0, \Sigma, \rightarrow_1)$ and $T_2 = (S_2, s_2^0, \Sigma, \rightarrow_2)$ be two TTS. We say that T_1 and T_2 are* timed bisimilar *if there exists a binary relation $\approx$ between S_1 and S_2, called* timed bisimulation relation, *such that:*

- *$s_1^0 \approx s_2^0$,*
- *if $s_1 \xrightarrow{a}_1 s_1'$ with $a \in \Sigma \cup \mathbb{R}_{\geq 0}$ and $s_1 \approx s_2$, then $\exists s_2'$ such that $s_2 \xrightarrow{a}_2 s_2'$ and $s_1' \approx s_2'$; conversely if $s_2 \xrightarrow{a}_2 s_2'$ with $a \in \Sigma \cup \mathbb{R}_{\geq 0}$ and $s_1 \approx s_2$, then $\exists s_1'$ such that $s_1 \xrightarrow{a}_1 s_1'$ and $s_1' \approx s_2'$.*

Lemma 2. *For every safe TPN, the TTS induced by the clocks-on-tokens semantics is timed bisimilar to the TTS induced by the clocks-on-transitions semantics.*

Proof. Every state $S = (M, dob, \theta)$ of the clocks-on-tokens semantics maps to a state $f(S) \overset{\text{def}}{=} (M, \nu, \theta)$ with $\nu(t) \overset{\text{def}}{=} \theta - doe(t)$. Note that f may not be injective. Then we define the relation $\approx$ as:

$$(M, \nu, \theta) \approx (M, dob, \theta) \iff (M, \nu, \theta) = f((M, dob, \theta))$$

and we show that $\approx$ is a timed bisimulation relation. The relation between initial states is immediate. Consider now related states $(M, \nu, \theta) \approx (M, dob, \theta)$.

- $(M,\nu,\theta) \xrightarrow{\theta'} (M,\nu',\theta')$
  iff $\forall t \in En(M) \quad \nu(t) + (\theta' - \theta) \leq lfd(t)$
  iff $\forall t \in En(M) \quad \theta' - doe(t) \leq lfd(t)$
  iff $(M, dob, \theta) \xrightarrow{\theta'} (M, dob, \theta')$
- If $(M, dob, \theta) \xrightarrow{t} (M', dob', \theta)$, then $(M,\nu,\theta) \xrightarrow{t} f((M', dob', \theta))$ (this is immediate by the definition of f).
- If $(M,\nu,\theta) \xrightarrow{t} (M',\nu',\theta)$, then $(M, dob, \theta) \xrightarrow{t} (M', dob', \theta)$ with $dob'(p) = dob(p)$ for every $p \in M \setminus {}^\bullet t$ and $dob'(p) = \theta$ for every $p \in t^\bullet$. We only need to check that $f((M', dob', \theta)) = (M', \nu', \theta)$. □

4 Partial Order Representation of Runs: Processes

We will define the mapping Π from the timed words of a safe time Petri net to their partial order representation as processes. These processes are those described in [1]. We use a canonical coding like in [9].

Coding of Events and Conditions. Each process will be a set $\mathcal{E}$ of pairs $(e, \mathcal{E}(e))$, where e is an *event* and $\mathcal{E}(e) \in \mathbb{R}$ is its firing date. We denote $E_{\mathcal{E}}$ (or simply E) the set of events in $\mathcal{E}$. Each event e is itself a pair $({}^\bullet e, \tau(e))$ that codes an occurrence of the transition $\tau(e)$ in the process. ${}^\bullet e$ is a set of pairs $b \overset{\text{def}}{=} ({}^\bullet b, place(b))$. Such a pair is called a *condition* and refers to the token that has been created by the event ${}^\bullet b$ in the place $place(b)$. We say that the event $e \overset{\text{def}}{=} ({}^\bullet e, \tau(e))$ *consumes* the conditions in ${}^\bullet e$. Symmetrically the set $\{(e,p) \mid p \in \tau(e)^\bullet\}$ of conditions that are *created* by e is denoted $e^\bullet$. A virtual initial event $\perp$ is used as ${}^\bullet b$ for initial conditions. By convention $\perp^\bullet \overset{\text{def}}{=} \{\perp\} \times M_0$ and $\mathcal{E}(\perp) \overset{\text{def}}{=} 0$. We write $E_\perp$ for $E \cup \{\perp\}$.

For all set B of conditions, we denote $Place(B) \overset{\text{def}}{=} \{place(b) \mid b \in B\}$, and when the restriction of $place$ to B is injective, we denote $place^{-1}_{|B}$ its inverse, and for all $P \subseteq Place(B)$, $Place^{-1}_{|B}(P) \overset{\text{def}}{=} \{place^{-1}_{|B}(p) \mid p \in P\}$. The set of conditions that remain at the end of the process $\mathcal{E}$ (meaning that they have been created by an event of E, and no event of E has consumed them) is $\uparrow(E) \overset{\text{def}}{=} \bigcup_{e \in E_\perp} e^\bullet \setminus \bigcup_{e \in E} {}^\bullet e$.

To summarize the coding of the processes, it is convenient to define a set D_N, such that all the events that appear in the processes of a Petri net N, are elements of D_N.

Definition 6 (D_N). *We define D_N as the smallest set such that*

$$\forall B \subseteq \bigcup_{e \in D_{N\perp}} e^\bullet \quad \forall t \in T \quad Place(B) = {}^\bullet t \implies (B,t) \in D_N \ .$$

Notice that this inductive definition is initialized by the fact that $\perp \in D_{N\perp}$.

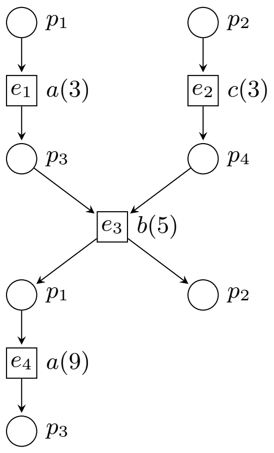


Fig. 2. A representation of a process of the time Petri net of Figure 1. The dates of the events are in brackets.

Causality. We define the relation $\rightarrow$ on the events as: $e \rightarrow e' \iff e^\bullet \cap {}^\bullet e' \neq \emptyset$. The reflexive transitive closure $\rightarrow^*$ of $\rightarrow$ is called the *causality* relation. Two events of a process that are not causally related are called *concurrent*. For all event e, we denote $\lceil e \rceil \overset{\text{def}}{=} \{f \in E \mid f \rightarrow^* e\}$, and for all set E of events, $\lceil E \rceil \overset{\text{def}}{=} \bigcup_{e \in E} \lceil e \rceil$.

Dates. We define the date of birth of the token which stands in a place $p \in Place(\uparrow(E))$ after a process $\mathcal{E}$ as $dob_{\mathcal{E}}(p) \overset{\text{def}}{=} \mathcal{E}({}^\bullet(place^{-1}_{|\uparrow(E)}(p)))$.

This allows us to define the state that is reached after a process $\mathcal{E}$ of N as: $RS(\mathcal{E}) \overset{\text{def}}{=} (Place(\uparrow(E)), dob_{\mathcal{E}}, \max_{e \in E_\perp} \mathcal{E}(e))$. Actually we will use this definition even for sets $\mathcal{E}$ of dated events which are not processes.

Definition 7. *The function Π that maps each timed word $((t_1, \theta_1), \ldots, (t_n, \theta_n))$ to a process is defined as follows:*

- $\Pi(\epsilon) \overset{\text{def}}{=} \emptyset$
- $\Pi(((t_1, \theta_1), \ldots, (t_{n+1}, \theta_{n+1}))) \overset{\text{def}}{=} \mathcal{E} \cup \{(e, \theta_{n+1})\}$, *where*
  - $\mathcal{E} \overset{\text{def}}{=} \Pi(((t_1, \theta_1), \ldots, (t_n, \theta_n)))$ *and*
  - *the event* $e \overset{\text{def}}{=} (Place^{-1}_{|\uparrow(E)}({}^\bullet t_{n+1}), t_{n+1})$ *represents the last firing of the sequence.*

In the representation of a process, the rectangles represent the events, and the circles represent the conditions. An arrow from a condition b to an event e means that $b \in {}^\bullet e$. An arrow from an event e to a condition b means that $e = {}^\bullet b$.

Figure 2 shows a process of the TPN of Figure 1, which is the image by Π of the two timed words $((a, 3), (c, 3), (b, 5), (a, 9))$ and $((c, 3), (a, 3), (b, 5), (a, 9))$: permuting the concurrent events corresponding to the first occurrences of a and c yields two different timed words for the same process. Yet these permutations are very limited because the sequential semantics forces time to progress. We will obtain much more permutations in Section 5 with our relaxed semantics.

4.1 Characterization of Processes

Since timed processes are defined as sets of dated events (the events being taken from the set D_N defined above), a natural problem is to decide whether a subset of dated events is a process. In the general case, the answer is nontrivial and was treated in [1]. But the case of extended free choice time Petri nets behaves much better and we recall here how to characterize processes in this case. We assume that the net is normalized, which simplifies even more the formulation.

Lemma 3 (Characterization of processes). *Let N be a normalized safe extended free choice TPN and let $\mathcal{E} \subseteq D_N \times \mathbb{R}$ be a finite set of dated events (we denote E the set of events in $\mathcal{E}$). $\mathcal{E}$ is a process of N iff:*

- *E is causally closed: $\lceil E \rceil = E$;*
- *E is conflict free: $\nexists e, e' \in E \quad e \neq e' \ \wedge \ {}^\bullet e \cap {}^\bullet e' \neq \emptyset$;*
- *firing delays are met: $\forall e \in E \quad efd(\tau(e)) \leq \mathcal{E}(e) - doe(e) \leq lfd(\tau(e))$;*
- *$\mathcal{E}$ is* temporally complete*: every transition t enabled in the state $RS(\mathcal{E})$ reached after $\mathcal{E}$, satisfies $doe(t) + lfd(t) \leq \max_{e \in E_\perp} \mathcal{E}(e)$.*

Example. Consider the process of Figure 2 and replace the firing dates by parameters $\mathcal{E}(e_1)$, $\mathcal{E}(e_2)$, $\mathcal{E}(e_3)$, $\mathcal{E}(e_4)$. The values of the parameters that correspond to processes accepted by the TPN of Figure 1 are those satisfying the following constraints, given by earliest and latest firing delays of the events

$$\begin{cases} 0 \leq \mathcal{E}(e_1) \leq \infty \\ 3 \leq \mathcal{E}(e_2) \leq 4 \\ 0 \leq \mathcal{E}(e_3) - \max\{\mathcal{E}(e_1), \mathcal{E}(e_2)\} \leq 4 & \qquad \text{(after normalization } lfd(b) \text{ is set to 4)} \\ 0 \leq \mathcal{E}(e_4) - \mathcal{E}(e_3) \leq \infty \end{cases}$$

plus the constraint that the process is temporally complete:

$$\max\{\mathcal{E}(e_1), \mathcal{E}(e_2), \mathcal{E}(e_3), \mathcal{E}(e_4)\} - \mathcal{E}(e_3) \leq 4$$

i.e., here, that transition c, which is enabled in the final state, has not fired yet at the time of the end of the process.

It is worth noticing that every set $\mathcal{E}$ satisfying all the conditions but the last, is a prefix (i.e. a causally closed subset) of a process. The following lemma is a corollary of Theorem 3.2 of [6] and a strong property of extended free choice TPNs. The fact that it does not hold for general safe TPNs is at the origin of the difficulties to define their unfoldings.

Lemma 4 (Characterization of prefixes of processes). *Let N be a normalized safe extended free choice TPN satisfying the diverging time assumption and let $\mathcal{E} \subseteq D_N \times \mathbb{R}$ be a finite set of dated events (we denote E the set of events in $\mathcal{E}$). $\mathcal{E}$ is a prefix of a process of N iff:*

- *E is causally closed: $\lceil E \rceil = E$;*
- *E is conflict free: $\nexists e, e' \in E \quad e \neq e' \ \wedge \ {}^\bullet e \cap {}^\bullet e' \neq \emptyset$;*
- *firing delays are respected: $\forall e \in E \quad efd(\tau(e)) \leq \mathcal{E}(e) - doe(e) \leq lfd(\tau(e))$.*

Proof. By construction (and by Lemma 3), all the prefixes of the processes satisfy the constraints. We show the converse by induction on the number of events in $\mathcal{E}$. The case of the empty process is immediate.

Consider now a nonempty process $\mathcal{E}$ satisfying the constraints. Choose an event e of $\mathcal{E}$ with maximal date; if there are several candidates, choose one which is also maximal w.r.t. causality (causality is acyclic by construction of D_N). The selected event e is maximal w.r.t. causality among all the events of $\mathcal{E}$: the inequality about the earliest firing delays, with the fact that earliest firing delays are non negative, prevent events with strictly smaller date than e from being causal successors of e.

Process $\mathcal{E}$ without event e is causally closed, conflict free and satisfies the inequalities about the firing delays. Assume, by the induction hypothesis, that it is a prefix of a process $\mathcal{F}$. This process can be truncated if necessary so that all the events that are in $\mathcal{F}$ and not in $\mathcal{E}$ have a date strictly smaller than the date $\mathcal{E}(e)$ of e. This means that the current date in the state $RS(\mathcal{F})$ reached after $\mathcal{F}$, is smaller or equal to $\mathcal{E}(e)$. Therefore, if $\mathcal{F}$ contains an event f in conflict with e, then f can be dropped (together with its causal successors, if any): this still yields an acceptable $\mathcal{F}$ because, N being extended free choice and normalized, e and f have exactly the same input conditions, the same date of enabling and the same latest firing delay and this delay is not expired at time $\mathcal{E}(e)$.

Hence we can assume that $\mathcal{F}$ has no event in conflict with e. The idea is to add e to $\mathcal{F}$. But, because of their latest firing delay, some transitions enabled in $RS(\mathcal{F})$ may not be allowed to wait until time $\mathcal{E}(e)$. We can simply fire them and complete process $\mathcal{F}$ with the corresponding events, as long as such transitions remain. The diverging time assumption ensures that this terminates; and, for the same reason as before, no event in conflict with e needs to be added (they can wait until $\mathcal{E}(e)$). Finally e can be added, and $\mathcal{E}$ is a prefix of the process that we obtain. □

5 Back in Time Semantics

We now explore the semantics of time Petri nets when the time progress assumption is dropped.

Of course dropping the time progress assumption will generate new timed words. But we will show that for free choice TPNs, the relaxed clocks-on-tokens semantics remains related to the classical semantics in the sense that it produces the same partial-order executions.

5.1 Relaxed Clocks-on-Tokens Semantics

Definition 8 (relaxed clocks-on-tokens semantics). *The relaxed semantics is obtained simply by dropping the constraint of time progress $\theta' \geq \theta$ in the time delays. The constraint that no enabled transition overtakes its maximum delay, remains: $\forall t \in En(M) \quad \theta' \leq doe(t) + lfd(t)$.*

Hence time may go back in the past without any restriction. On the other hand, the constraint about maximum delays prevents time from progressing too much.

Timed Words and Processes. We define timed words like for the classical semantics, as sequences $((t_1, \theta_1), \ldots, (t_n, \theta_n))$ of transitions with their firing dates. Simply, now, the ordering $\theta_1 \leq \cdots \leq \theta_n$ may not hold. This does not prevent us from defining processes from these timed words, using the same mapping Π as for the classical semantics. We show that the processes that we obtain are prefixes of processes of the classical semantics.

Example. As an example, consider the TPN of Figure 1. The timed word $(a, 4), (c, 3)$ is accepted by the relaxed clocks-on-tokens semantics: after firing a at time 4, p_2 and p_3 are marked, with $dob(p_2) = 0$ and $dob(p_3) = 4$, and the current time is 4. After that, time may go back to 3, and c can fire.

Theorem 1 (Correctness). *Let N be an extended free choice TPN. Every process of N under the relaxed clocks-on-tokens semantics, is a prefix of a process of N under the classical semantics.*

Proof. We show, by induction on the length, that for every timed word w accepted by N under the relaxed clocks-on-tokens semantics, all the events of $\Pi(w)$ satisfy the conditions of Lemma 4 and that the state $RS(w)$ reached after $\Pi(w)$ is the state reached after w. The case of the empty timed word is immediate.

Now, assume that transition t fires at time θ from the state reached after a timed word w that satisfies the induction hypothesis. Let e be the event corresponding this firing of $t = \tau(e)$ at time $\theta = \mathcal{E}(e)$. The constraint that no enabled transition overtakes its maximum delay during the delay to time θ, is satisfied in particular by θ, and guarantees that $\mathcal{E}(e) - doe(e) \leq lfd(t)$. And the constraint about the earliest firing delay of t is checked when t fires, which implies that $\mathcal{E}(e) - doe(e) \geq efd(t)$. □

Definition 9 (Linearization). *Let $\mathcal{E}$ be a process or a prefix of a process and let L be a bijection from E to $\{1, \ldots, |E|\}$ such that for every i, j, $L(i) \to L(j) \implies i < j$ (i.e. L respects the causal ordering). Then the timed word $((\tau(L(1)), \mathcal{E}(L(1))), \ldots, (\tau(L(n)), \mathcal{E}(L(n))))$ is called a* linearization *of $\mathcal{E}$.*

Corollary 1. *Every timed word accepted by N under the relaxed clocks-on-tokens semantics, is a prefix of a permutation of a timed word accepted by N under the classical clocks-on-tokens semantics.*

Proof. By construction of the processes, every timed word w accepted by N under the relaxed clocks-on-tokens semantics, is a linearization of the process $\Pi(w)$, which is also a process of N under the classical semantics. □

Example. As an example, consider again the TPN N of Figure 1. The timed word $((a, 4), (c, 3))$, which is accepted by the relaxed clocks-on-tokens semantics, is a linearization of $\Pi((c, 3), (a, 4))$, represented on Figure 3, which is a process of N under the classical semantics.

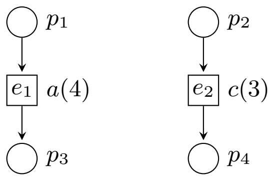


Fig. 3. A process of the time Petri net of Figure 1. The dates of the events are in brackets. This process has two linearizations: $((c,3),(a,4))$ and $((a,4),(c,3))$. Only the first one is accepted by the classical semantics, but both are accepted by the relaxed clocks-on-tokens semantics.

5.2 More Relaxed Clocks-on-Tokens Semantics

Our previous subsection was based on a minimalist modification of the classical semantics: we have simply dropped the time progress assumption. We have shown that this preserves the partial order behaviour and that the timed words accepted under the relaxed semantics are linearizations of processes under the classical semantics.

Yet not all linearizations of processes are accepted by the relaxed clocks-on-tokens semantics. For example, for the TPN N of Figure 1, $(a,5),(c,3)$ is a linearization of a process, but it is not accepted by the relaxed semantics. The reason is that $\mathit{lfd}(c) < 5$, i.e. the latest firing delay of c prevents time from progressing up to 5 while c is enabled. Still a and c are concurrent, which suggests that, if one makes abstraction of the temporal ordering, they can fire independently in any order: in particular one could fire a and then – after possibly going back in time – fire c.

For this we need to relax a bit more the constraints about time progress, and let time go back, but also go forth without any restriction as long as discrete actions are not concerned. Hence, only when firing a transition t, one checks that the earliest and latest firing delays are respected, and this check can be done only locally (on the transitions in conflict with t) since concurrent transitions are not concerned.

Definition 10 (more relaxed clocks-on-tokens semantics). *The more relaxed clocks-on-tokens semantics is defined from the clocks-on-tokens semantics, with the following changes.*

Time delay. *All the constraints on θ' for the time delay are dropped: the TPN can now reach any time θ' from any state.*

Discrete action. *But transition t can fire from state $(M, \mathit{dob}, \theta)$ if:*
- *t is enabled: $t \in \mathit{En}(M)$;*
- *t has reached its minimum firing delay: $\theta \geq \mathit{doe}(t) + \mathit{efd}(t)$;*
- *no transition in conflict with t overtakes its maximum firing delay:* $\forall t' \in T \quad {}^\bullet t' \cap {}^\bullet t \neq \emptyset \implies \theta \leq \mathit{doe}(t') + \mathit{lfd}(t')$.

Notice that after normalization of an extended free choice TPN, the constraint about latest firing delays can simply be checked on the transition t itself: $\theta \leq \mathit{doe}(t) + \mathit{lfd}(t)$.

Example. Consider again the TPN of Figure 1. It can now wait until time 5 without firing any transition, fire a, and then go back to time 3 and fire c.

Again correctness of the more relaxed clocks-on-tokens semantics is expressed in terms of preservation of processes.

Theorem 2 (Correctness). *Let N be an extended free choice TPN. Every process of N under the more relaxed clocks-on-tokens semantics, is a prefix of a process of N under the classical clocks-on-tokens semantics.*

Proof. Like for Theorem 1, we show, by induction on the length, that for every timed word w accepted by N under the more relaxed clocks-on-tokens semantics, all the events of $\Pi(w)$ satisfy the conditions of Lemma 4 and that the state $RS(w)$ reached after $\Pi(w)$ is the state reached after w. The difference is that, with the more relaxed semantics, the condition about the latest firing delay is not checked during the delay to θ, but when firing t. □

Theorem 3 (Completeness). *The timed words accepted by the more relaxed clocks-on-tokens semantics of an extended free choice TPN coincide precisely with the linearizations of the prefixes of its processes under the classical semantics.*

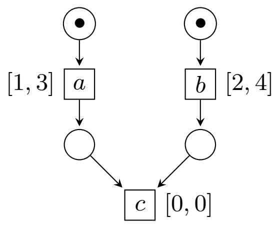
Proof. That the more relaxed clocks-on-tokens semantics generates only linearizations of prefixes of its processes under the classical semantics, is a direct corollary of Theorem 2.

It remains to check that it generates all of them. We proceed by induction on the number of events in the prefix. The case of the empty prefix is immediate. Let $\mathcal{E}$ be a nonempty process under the classical semantics, and $((t_1, \theta_1), \ldots, (t_n, \theta_n))$ be a linearization of $\mathcal{E}$. By definition of the linearization, $((t_1, \theta_1), \ldots, (t_{n-1}, \theta_{n-1}))$ is also a linearization of a prefix. Assume, by the induction hypothesis, that it is accepted under the more relaxed semantics. It remains to show that the more relaxed semantics lets transition t_n fire at time θ_n from the state reached after $((t_1, \theta_1), \ldots, (t_{n-1}, \theta_{n-1}))$. The constraint of Lemma 4 applied to $\mathcal{E}$ gives precisely this constraint for the last event of the linearization (the one corresponding to the firing of t_n at time θ_n). □

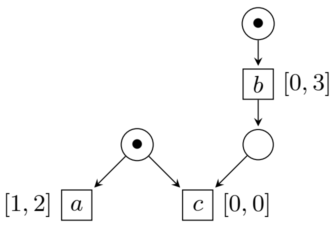
6 Discussion and Perspectives

Relaxed Clocks-on-Transitions Semantics. As an alternative to our relaxed clocks-on-transitions semantics, one could envisage starting from the classical clocks-on-transitions semantics and dropping the constraint $\theta \leq \theta'$.

But even for free choice TPNs, timed words accepted by the relaxed clocks-on-transitions semantics are not necessarily permutations of timed words accepted by the classical clocks-on-transitions semantics. For the TPN of Figure 4(a), the relaxed clocks-on-transitions semantics accepts the timed word $((a, 3), (b, 2), (c, 2))$: it allows one to fire a at time 3, followed by b at time 2. In the



(a) Counter-example for the clocks-on-transitions semantics



(b) Counter-example for non free choice TPNs

Fig. 4. Counter-examples

reached state, c is enabled and its clock has value $\nu(c) = 0$ since it was enabled by the firing of b. As a result, c fires at time 2. In the classical clocks-on-transitions semantics, firing b at time 2 and a at time 3 implies firing c at time 3.

This shows that the clocks-on-tokens semantics, although equivalent to the clocks-on-transitions semantics in the classical setting, is more robust to variations of the semantics. Another argument in this sense is given in [4] for the case of unbounded time Petri nets.

Counter-Example for Non Free Choice TPNs. The TPN of Figure 4(b) is not free choice since transitions a and b are in conflict, but do not have the same presets. With any of the relaxed semantics that we defined, a can fire at time 1 from the initial state, and then b can fire at time 0. This is not a linearization of any process because in the classical semantics, firing b at time 0 implies firing c at time 0 and thus prevents a from firing.

Perspectives. We believe that our result can be adapted quite easily to other formalisms for real-time concurrent systems (arc-timed Petri nets, networks of timed automata...) provided a restriction on local choices is done (corresponding to the extended free choice assumption for TPNs). Without this assumption, the previous counter-example already shows how transitions that are apparently concurrent can influence one the other. Then a back in time semantics for the general case should be constrained by these dependencies. Some of these dependencies played a role in the definition of unfoldings for TPNs, but identifying them precisely remains a challenge.

Finally, the algorithms for the analysis of timed models construct and solve systems of linear constraints on temporal parameters (like occurrence time, value of clock...) By relaxing the constraints about time progress, our back in time semantics would generate fewer inequalities and fewer symbolic states. And it preserves properties like fireability of a transition or reachability of a place. We plan to experiment the construction of the symbolic state graph generated by our semantics and evaluate how much it improves the analysis algorithms.

References

1. Aura, T., Lilius, J.: A causal semantics for time Petri nets. Theoretical Computer Science 243(1-2), 409–447 (2000)
2. Benveniste, A., Fabre, E., Haar, S.: Markov nets: probabilistic models for distributed and concurrent systems. IEEE Trans. Automat. Contr. 48(11), 1936–1950 (2003)
3. Best, E.: Structure theory of Petri nets: the free choice hiatus. In: Brauer, W., Reisig, W., Rozenberg, G. (eds.) APN 1986, Part 1. LNCS, vol. 254, pp. 168–205. Springer, Heidelberg (1987)
4. Boucheneb, H., Lime, D., Roux, O.H.: On multi-enabledness in time Petri nets. In: Colom, J.-M., Desel, J. (eds.) PETRI NETS 2013. LNCS, vol. 7927, pp. 130–149. Springer, Heidelberg (2013)
5. Boyer, M., Roux, O.H.: Comparison of the expressiveness of arc, place and transition time petri nets. In: Kleijn, J., Yakovlev, A. (eds.) ICATPN 2007. LNCS, vol. 4546, pp. 63–82. Springer, Heidelberg (2007)
6. Chatain, T.: Dépliages symboliques de réseaux de Petri de haut niveau et application à la supervision des systèmes répartis. Thèse de doctorat, Université Rennes 1, Rennes, France (November 2006)
7. 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)
8. Desel, J., Esparza, J.: Free Choice Petri nets. Cambridge University Press (1995)
9. Engelfriet, J.: Branching processes of Petri nets. Acta Informatica 28(6), 575–591 (1991)
10. Merlin, P.M., Farber, D.J.: Recoverability of communication protocols – implications of a theorical study. IEEE Transactions on Communications 24 (1976)
11. Niebert, P., Qu, H.: Adding invariants to event zone automata. In: Asarin, E., Bouyer, P. (eds.) FORMATS 2006. LNCS, vol. 4202, pp. 290–305. Springer, Heidelberg (2006)
12. Reynier, P.-A., Sangnier, A.: Weak time Petri nets strike back! In: Bravetti, M., Zavattaro, G. (eds.) CONCUR 2009. LNCS, vol. 5710, pp. 557–571. Springer, Heidelberg (2009)

A Mechanized Semantic Framework for Real-Time Systems*

Manuel Garnacho, Jean-Paul Bodeveix, and Mamoun Filali-Amine

IRIT - CNRS - Université de Toulouse, France

Abstract. Concurrent systems consist of many components which may execute in parallel and are complex to design, to analyze, to verify, and to implement. The complexity increases if the systems have real-time constraints, which are very useful in avionic, spatial and other kind of embedded applications. In this paper we present a logical framework for defining and validating real-time formalisms as well as reasoning methods over them. For this purpose, we have implemented in the Coq proof assistant well known semantic domains for real-time systems based on labelled transitions systems and timed runs. We experiment our framework by considering the real-time CSP-based language FIACRE, which has been defined as a pivot formalism for modeling languages (AADL, SDL, ...) used in the TOPCASED project. Thus, we define an extension to the formal semantic models mentioned above that facilitates the modeling of fine-grained time constraints of FIACRE. Finally, we implement this extension in our framework and provide a proof method environment to deal with real-time system in order to achieve their formal certification.

1 Introduction

Real-time concurrent systems consist of many components which may execute in parallel and communicate data or synchronize at a specified time. Therefore, these systems are complicated to design, to analyze, to verify, and finally to implement. The complexity arises from the nondeterminism of behaviors, time constraints and the combinations of ways in which the components can interact. In order to be able to prove or verify properties on such systems one needs to formalize their semantics.

Formal semantics of specification and programming languages are *mathematical descriptions* of the meaning of programs (and their behaviors) written in these languages. They play a very important role in many areas of computer science such as *verification* of critical systems because they enable us to formally prove that these systems meet their specifications. Formal semantics of component-based languages, which are useful to design distributed or concurrent systems, are commonly defined in terms of *transition systems*. If the language has also real-time features, then its semantics can be expressed as *timed transition systems* [4] (TTS for short, see Definition 3). For this paper, we address

* This work has been partly funded by the FNRAE project QUARTEFT.

the challenge of providing a framework that allows to build *mechanized reasonings* over timed systems within a *proof assistant*. Doing so requires to *mechanize the semantics* of timed systems in adapted mathematical terms, which are here theories of transition systems and execution runs (or traces) which include time aspects.

The present work proposes also an extension to the semantic model of transition systems *with time constraints* proposed by T.Henzinger *et al* [16] that we call *time constrained transition systems* (TCTS for short, see Definition 7) since the term TTS is overloaded and already used here. The purpose of this extension is to *model more directly and easily* some real-time constructs of specification languages and then be able to prove *timed temporal properties* over critical embedded systems. We define *semantic interpretations* of systems modeled as TCTS in terms of TTS and *timed runs* (see Section 2.2) that enable to reason over these models and their time properties. Those semantic interpretations correspond to arrows 2 and 3 of Figure 1 which describes the global architecture of our framework. Arrow 1, which addresses the semantics interpretation of the syntactical constructs of our component-based language (namely FIACRE), goes beyond our current purpose and is not presented here (this is our long term goal).

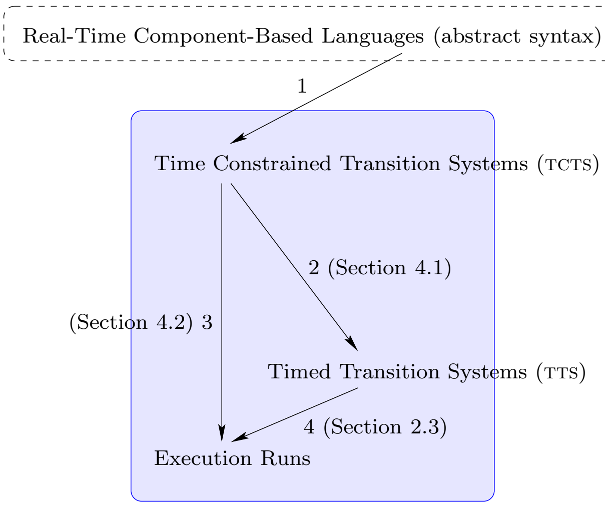

Real-Time Component-Based Languages (abstract syntax)
1
Time Constrained Transition Systems (TCTS)
2 (Section 4.1)
(Section 4.2) 3
Timed Transition Systems (TTS)
4 (Section 2.3)
Execution Runs

Fig. 1. Schematic description of the architecture of our semantic framework

There are several reasons for wanting to mechanize such theories based on transition systems and timed runs in an interactive theorem-prover rather than building a dedicated automatic tool [13]. The most important reason is certainly that proof assistants like COQ [7], HOL [18] or PVS [22], provide a generic and very expressive proof environment for dealing with problems which cannot be automatically decided. We claim also that the results which are encoded and checked with such tools reach probably the currently highest level of confidence in formal software verification. In other words, proofs built and then checked with, for example Coq, avoid to a very high extent fallacious and incomplete arguments that are so easy and so common to find in standard mathematical proofs. Then, all the proofs that we provide in this paper have been encoded in the calculus of inductive constructions (cic for short) using the Coq proof assistant, and then automatically checked to guarantee the mathematical soundness of the proofs [1]. However, although all of this formalization work (shaded part of Figure 1) has been carried out in Coq, we refrained from displaying Coq syntax here. We adopt standard mathematical notations as much as possible for the sake of clarity and because a large part of this work is about semantics of real-time systems that is widely independent of Coq.

Overview. The remainder of this paper is structured as follows: Section 1.1 introduces the general principles of component-based languages. We describe their timed semantics through a toy example. Section 2 recalls the basis of transition systems and execution runs. Section 3 defines an extension of transition systems with time constraints. Section 4 is dedicated to the semantic interpretations of transition systems in two distinct models that address the dynamic time aspects of real-time systems. Then, in Section 5 we explain why our work has been implemented in the interactive theorem prover Coq and how it can be used for formal verification. We draw a conclusion and present some directions for future works in Section 6.

1.1 Timed Semantics of Component-Based Languages

Existing specification languages inspired by [17] are useful to represent both behavioral and timing aspects of concurrent (or distributed) real-time systems. For example, fiacre [6] is defined over two main notions : (1) *processes* that describe the behavior of sequential components. A process is defined by a set of control states (or positions), each associated with a piece of program built from deterministic constructs available in classical programming languages, nondeterministic constructs (nondeterministic choice of the transition to take, for example), communication events on ports, and jumps to next position; (2) *components* that describe the composition of processes. A component is defined as a parallel composition of components and/or processes communicating through *ports* and shared variables.

In this paper we focus only on nondeterministic transitions and timing constraints of synchronizations (composition and priorities are adressed in [12]). fiacre allows to model real-time behaviors by using timed ports, *i.e.* ports with their own timer (or clock). A timed transition connects two control states using such a port. Moreover, fiacre allows to code such transitions in two ways : the first one preserves the clocks of other *enabled transitions* whereas the second one resets the clock of other *enabled transitions.*

[1] We invite the interested reader to browse the sources of our development at the web address : `http://www.irit.fr/~Manuel.Garnacho/Coq/FORMATS13.`

The example presented in Figure 2 describes a process which declares three local ports, `p`, `q` and `r`, and two variables of type `nat`, namely `x` and `y`. In this specification language, local ports are only used to temporize transitions and processes which use those ports do not get "synchronized" with others through those ports. The port `p` is time constrained by the interval $[1,3[$, `q` by $]3,\infty[$ and `r` by $[1,1]$. Considering `p`, it means that when a transition which synchronizes on it is enabled, it must fire after waiting *at least* one (included) and *at most* three (not included) time units. In the same way, `q` delays the process `P` for at least three (not included) time units[2] and `r` exactly one unit of time.

Then, we see that in the process `P`, there are two control states, namely `s0` and `s1`. Only `s0` is an initial one, which means that the process has to start its execution from it. But let us first take a look at `s1` : *from* this control state, a *nondeterministic choice* is defined with the construction `select`. This choice includes two concurrent transitions : the first one that synchronizes on `p` then adds to `y` the value of `x`; the second one that synchronizes on `q` then assigns to `x` the value of `y` plus 2. Afterward, each of those transitions *jumps* to a next control state with the construction `to`, *i.e.* it jumps back to `s1` in the case of an action on `p` and goes to `s0` in the other case. The timed-semantics of the construction `to`, consists in *resetting* the timer of all transitions included in the same nondeterministic choice (we provide in comments the set of ports that *have to be reset* for each transition). This notion will be formalized in Section 3, thanks to an extension of the semantic model introduced in [16]. Consequently, from the control state `s1` the process `P` is never synchronized on `q`, so the variable `x` will never be assigned and it never goes back to `s0`. Indeed, after every single unit of time the process will be synchronized on `p` before strictly two more time units, after which it resets the timer of `q` before having waited enough time to synchronize on it. Thus, the second transition of the nondeterministic choice from `s1` of `P` is *dead code* due to the time constraints of `P`.

Now, through the same example we present another construction that *loops* to the current control state but with a different timed-semantics. This construction is still related to the nondeterministic choice of transitions and is defined with the keyword `loop`. It consists in going back to the current control state while *preserving the time already waited by other transitions* (which were not taken) included in the same nondeterministic choice as the one that has been taken. The clock of the taken transition is reset however. For example, in the nondeterministic choice from the control state `s0` of `P`, the first transition loops after synchronizing on `r` and then increments `x`. This transition occurs after exactly every single time unit and then only resets its own timer. On the other hand, the nondeterministic choice allows the process to synchronize on `q` at any time after being continuously enabled for at least three units of time. Note that the transition through `q` from `s0` may never occur because of its unconstrained maximal delay. Proving such semantics properties on the value of `x` and `y` or that the transition through `q` from `s1` is *dead code*, can be as much crucial as difficult.

[2] Time bounds are either integral or infinite. Elapsing of time is continuous.

```
process P is

  ports p in [1,3[, q in ]3,∞[, r in [1,1]
  var x, y : nat := 0
  states s0 s1
  init to s0

  from s0 select
 /*1*/      r ; x := x+1; loop     /* reset = {r} */
 /*2*/ []  q ; to s1              /* reset = {r,q} */
      end

  from s1 select
 /*3*/      p ; y := x+y; to s1   /* reset = {p,q} */
 /*4*/ []  q ; x := y+2; to s0   /* reset = {p,q} */
      end
```

Fig. 2. An example real-time process specified in FIACRE

Related Work. Starting with the work of T. Henzinger *et al.* [16, 15], we extend their mathematical semantic model for specifying real-time processes such as the one presented in Figure 2. All definitions and theorems presented in the following are directly derived from a mechanization developed in COQ. This part of our contributions connects to the previous work of R. Cardell-Oliver *et al.* [14] about the embedding in HOL of [16] and C. Paulin-Mohring about the formalization of timed automata in COQ [20]. With respect to the Petri Net community, our work is in the spirit of B. Berard *et al.* [4]. However, we are concerned by a component-based language with states and ports and we go further since we adress its mechanization. Morever, our work is not in the same scope as [13] because our goal is not to develop an automatic verification tool certified in COQ. Our purpose is to provide a real-time semantic framework which can be used to formalize real-time patterns and more generally real-time specification languages. As a matter of fact, we present in the following the formalization of the FIACRE language where we are especially interested in the semantic differences between the contructs `loop` and `to` (see Section 1.1). Also, the proof environment of our framework allows to *certify transformations* between real-time formalisms. For instance in [8], we are concerned by the transformation from AADL to FIACRE, which requires formal certification for avionic or spatial applications.

2 Semantics Kernel

This section is dedicated to the basis of classical semantic notions we use in our work. These notions are of two kinds: *timed transition systems* and *timed execution runs* [4]. Moreover, *both include time* and since *time is dense* in most real-time specification languages (including FIACRE), we formalize time in this paper with non-negative real numbers : $\mathbb{R}^+$. However, in our COQ mechanization we have developed an axiomatic *theory of time*[3] which can be instantiated by constructive reals [13] as well as by `nat` to encode discrete time (if necessary).

[3] `http://www.irit.fr/~Manuel.Garnacho/Coq/FORMATS13/time.html`.

These semantic models enable us to describe formally the dynamics of real-time systems, mixing the evolution of their states and the progress of time, to reason over those with rigorous logical methods.

2.1 Transition Systems

First of all, we remind the basic model of *labelled transition systems* [2] that is usually used to give a mathematical representation of computer programs.

Definition 1 (Labelled Transition Systems). *A labelled transition system (*LTS *for short) is a* 4*-tuple* $lts \overset{def}{=} \langle L, S, \boldsymbol{init}, \boldsymbol{next}\rangle$*, where :* L *is the set of* labels *of* lts*;* S *is a set of* states *(or stores);* $\boldsymbol{init}$ *is a non-empty subset of* S *that defines the initial states of* lts*; and* $\boldsymbol{next}$ *defines the set of transitions of* lts *that are triplets of the form* $(\mathsf{sto}, \ell, \mathsf{sto}') \in S \times L \times S$*, also denoted as* $\mathsf{sto} \xrightarrow{\ell} \mathsf{sto}'$*, where* $\mathsf{sto} \in S$ *is the source state,* $\ell \in L$ *is the label and* $\mathsf{sto}' \in S$ *the target state of the transition.*

Definition 2. *Assuming a* LTS*,* $lts \overset{def}{=} \langle L, S, \boldsymbol{init}, \boldsymbol{next}\rangle$*, a label* $\ell \in L$ *is* enabled *from a state* $\mathsf{sto} \in S$*, if there exists a state* $\mathsf{sto}' \in S$ *such that the triplet* $(\mathsf{sto}, \ell, \mathsf{sto}')$ *belongs to* $\boldsymbol{next}$*. Formally, we define the predicate* $\boldsymbol{enb}$ *over* $S \times L$ *as* $\boldsymbol{enb}_{\mathsf{sto}}(\ell) \overset{def}{=} \exists \mathsf{sto}' \in S, \boldsymbol{next}(\mathsf{sto}, \ell, \mathsf{sto}')$

Definition 3 (Timed Transition Systems). *A timed transition system (*TTS *for short) over a set of labels* L*, is a* LTS *over* $L \cup \mathbb{R}^+$*,* $\langle L \cup \mathbb{R}^+, S, \boldsymbol{init}, \boldsymbol{t\_next}\rangle$*, that satisfies the following properties :*

- *zero delay :* $\forall \mathsf{sto} \in S, \mathsf{sto} \xrightarrow{0} \mathsf{sto}$
- *time-determinism :* $\forall \mathsf{sto}, \mathsf{sto}', \mathsf{sto}'' \in S, \forall \delta \in \mathbb{R}^+,$
$$\mathsf{sto} \xrightarrow{\delta} \mathsf{sto}' \Rightarrow \mathsf{sto} \xrightarrow{\delta} \mathsf{sto}'' \Rightarrow \mathsf{sto}' = \mathsf{sto}''$$
- *additivity :* $\forall \mathsf{sto}, \mathsf{sto}', \mathsf{sto}'' \in S, \forall \delta, \delta' \in \mathbb{R}^+,$
$$\mathsf{sto} \xrightarrow{\delta} \mathsf{sto}' \Rightarrow \mathsf{sto}' \xrightarrow{\delta'} \mathsf{sto}'' \Rightarrow \mathsf{sto} \xrightarrow{\delta+\delta'} \mathsf{sto}''$$
- *continuity :* $\forall \mathsf{sto}, \mathsf{sto}'' \in S, \forall \delta', \delta'' \in \mathbb{R}^+,$
$$\mathsf{sto} \xrightarrow{\delta'+\delta''} \mathsf{sto}'' \Rightarrow \exists \mathsf{sto}', \mathsf{sto} \xrightarrow{\delta'} \mathsf{sto}' \wedge \mathsf{sto}' \xrightarrow{\delta''} \mathsf{sto}''$$

2.2 Executions

We remind now the classical representation of runs in a timed context, that is useful to represent *the behaviors* of systems and then reason with temporal linear logic [21] about those.

Definition 4 (Timed execution states). *With respect to a set of states* S *and a set of events* L*, a* timed execution state *(*TES *for short),* s*, is a triplet of the form* $\langle \mathsf{sto}, \mathsf{evt}, \mathsf{now}\rangle$ *where :*

- $\mathsf{sto}_s \in S$*, is the state (or store) reached in* s
- $\mathsf{evt}_s \in L \cup \{start\} \cup \{div\}$*, is the event that has reached* s*.*
- $\mathsf{now}_s \in \mathbb{R}^+$ *gives the current global time (or date) in* s*.*

Remarks. The two special events *start* and *div* are seen, respectively, at the *beginning* of a run and whenever *time is diverging* (in case of deadlock, for instance) in a run.

Definition 5 (Runs). *A* run *(or execution sequence) is a function σ, from $\mathbb{N}$ to* TES*, that represents a (possibly finite) sequence of events all of which reach a state at a given date. For a* TES *s reached at a position i of σ, we have $\sigma_i = s$.*

Remarks. For the sake of conciseness, we allow ourselves to substitute s (σ_i) by i in the following whenever it might simplify the notations (as below). Also, according to our representation of runs, finite ones are encoded using a repetition of the special event *div* after the last state.

Definition 6 (Well formed runs). *According to the previous definition, a run σ is* well formed *if it satisfies the following properties :* **start** *: $\mathsf{evt}_0 =$ start $\wedge$ $\mathsf{now}_0 = \delta^0_\sigma$; and* **monotonicity** *: $\forall i \in \mathbb{N}$, $\mathsf{now}_i \leq \mathsf{now}_{i+1}$, where δ^0_σ is the initial delay associated to the beginning (the event start) of the run σ.*

Remark. This definition of timed runs does not satisfy the *progress* property ($\forall r \in \mathbb{R}^+$, $\exists i \in \mathbb{N}$, $r \leq \mathsf{now}_i$) [16] because of *Zeno behaviors* (that include an infinite number of discrete steps in a finite amount of time) that may be generated from the component-based source language, as for instance in FIACRE, that we want to capture.

2.3 Semantics of Timed Transition Systems in Terms of Runs

We define now the semantics of TTS in terms of runs (arrow 4 of Figure 1).

Predicates. In order to describe the semantics of TTS*s* in terms of runs, we have to define two predicates that express when a given label may be taken and when a label is taken. Assume a given run σ at some position i, σ-$\boldsymbol{enb}_i(\ell)$ means that the label ℓ may be taken at i and $\boldsymbol{taken}_i(\ell)$ means that ℓ is taken at i. Formally :

- $\sigma\text{-}\boldsymbol{enb}_i(\ell) \overset{def}{=} \exists \mathsf{sto}, \mathsf{sto}' \in S, \exists \delta \in \mathbb{R}^+, \boldsymbol{next}(\mathsf{sto}_i, \ell, \mathsf{sto}) \wedge \boldsymbol{next}(\mathsf{sto}, \delta, \mathsf{sto}')$
- $\boldsymbol{taken}_i(\ell) \overset{def}{=} \mathsf{evt}_{i+1} = \ell$ (sto_{i+1} corresponds to one of the sto' above)

Remark. The intermediate state, sto, used in the definition of σ-$\boldsymbol{enb}$ is needed because the runs that we consider do not include explicit *duration.* There are other formalisms of execution runs where time passing is explicit (mixing durations and events) but here, to avoid *Zeno phenomena* (coming from the TTS*s* and not from the systems theirselves) in runs, we only consider events.

Semantic Interpretation in Terms of Runs. The run-semantics of a TTS, $tts \overset{def}{=} \langle L \cup \mathbb{R}^+, S, \boldsymbol{init}, \boldsymbol{t\_next} \rangle$, is a set of well formed runs $\Sigma(tts)$, that expresses all possible behaviors of *tts* at execution, such as each of those has to fulfill that $\boldsymbol{init}(\mathsf{sto}_0)$ and the following step condition :

$\forall i$, either

$\exists \ell$ such that $\exists \mathsf{sto} \in S, \exists \delta \in \mathbb{R}^+, \boldsymbol{t\_next}(\mathsf{sto}_i, \ell, \mathsf{sto}) \wedge \boldsymbol{t\_next}(\mathsf{sto}, \delta, \mathsf{sto}_{i+1})$

and $\mathsf{now}_{i+1} = \mathsf{now}_i + \delta$ and $\boldsymbol{taken}_i(\ell)$ (a label is taken)
or $\neg\exists\ell$ such that $\sigma\text{-}\boldsymbol{enb}_i(\ell)$ and $\forall j > i, \mathsf{sto}_j = \mathsf{sto}_i \ \wedge\ \mathsf{evt}_j = div$ (time is diverging).

Graphically, when a label ℓ is taken at some position i, it can be represented as $\sigma_i \xrightarrow{\ell} s \xrightarrow{\delta} \sigma_{i+1}$, where s is an implicit intermediate TES which is not visible in the run, defined as $\langle \mathsf{sto}_{i+1}, \mathsf{evt}_{i+1}, \mathsf{now}_i \rangle$.

3 A Time Constrained Model for Real-Time Systems

We give in this section an *extended definition* of the semantics model proposed by T. Henzinger *et al.* in [16] that deals with advanced (*i.e.* fine-grained) timed constructs such as the ones presented in Section 1.1. As in [16], we incorporate real-time constraints to LTS in order to ensure that a labelled transition is fired neither too early nor too late. An extension to their initial model is the so-called *reset relation* that specifies which clocks (or timers) are reset after the firing of a given transition. We call this model *Time Constrained Transition Systems* (TCTS for short) since time features are only expressed as syntactic timed constraints on transitions. We have introduced this semantics model in order to verify real-time processes such as the one presented in Figure 2 more easily. Moreover, the exhibited semantic model is interesting by itself since it can be reused to define the semantics of real-time component-based langages such as FIACRE [11] or BIP[3] or even high-level specification models such as Timed Petri Nets with *read-arcs* [4]. We also present two timed-semantic interpretations, corresponding to arrows 2 and 3 of Figure 1.

3.1 Time Constrained Transition Systems

In order to reason about timing of transitions, we identify each transition by its name. For instance, according to the process `P` of Figure 2, its corresponding TCTS include four names, one for each of the four possible transitions from its two nondeterministic choices, even if the number 2 and 4 synchronize both on `q`. Moreover, we suppose that there is an implicit clock (while time is explicit in TTS) associated to every transition. Then, this model allows to specify static time constraints over the transitions of a given system.

Definition 7. *A Time Constrained Transition System, namely tcts, is a* 8*-tuple* $\langle L, T, S, \mathsf{lbl}, \boldsymbol{init}, \boldsymbol{next}, \mathcal{R}, I\rangle$*, where* $\langle L, S, \boldsymbol{init}, \boldsymbol{next}\rangle$ *is a* LTS*, and :*

- T *is the set of* (the names of) transitions *of tcts.*
- lbl *is a function from* T *to* L*, that associates to every transition a label.*
- $\mathcal{R}$ *is the reset transition relation.* $(tr, tr') \in \mathcal{R}$ *states that at* execution, *the firing of tr resets the* implicit clock *of* tr'*. Otherwise, the implicit clock associated to* tr' *keeps running. For all* $tr \in T$, $(tr, tr) \in \mathcal{R}$ *(*$\mathcal{R}$ *is refelexive).*
- I *is a function that assigns to every label* $\ell \in L$ *a non-empty interval of* $\mathbb{R}^+$*.* I_ℓ *(or* $I(\ell)$*) specifies both minimal (lower) and maximal delay (upper bound) to elapse once* ℓ *has been enabled (see Definition 2).*

We introduce the downward and the upward closure of an interval I_ℓ defined respectively by $\overleftarrow{I_\ell} \overset{def}{=} \{t \mid \exists r \in I_\ell, t \leq r\}$ and $\overrightarrow{I_\ell} \overset{def}{=} \{t \mid \exists r \in I_\ell, r \leq t\}$. For example, if I_ℓ is defined by the interval $[min, max]$ then $\overleftarrow{I_\ell}$ is $[0, max]$ and $\overrightarrow{I_\ell}$ is $[min, \infty[$.

Example 1. Consider the process given in Figure 2, we define its corresponding TCTS as $\langle L, T, S, \mathsf{lbl}, \boldsymbol{init}, \boldsymbol{next}, \mathcal{R}, I\rangle$, where :

- $L \overset{def}{=} \{\ell_p, \ell_q, \ell_r\}$
- $S \overset{def}{=} \{(x, y, loc) \mid x, y \in \mathbb{N} \wedge loc \in \{l_0, l_1\}\}$
- $\forall \mathsf{sto} \in S, \boldsymbol{init}(\mathsf{sto}) \overset{def}{=} \mathsf{sto}(x) = 0 \wedge \mathsf{sto}(y) = 0 \wedge \mathsf{sto}(loc) = l_0$
- $\boldsymbol{next}(\mathsf{sto}, \ell, \mathsf{sto}') \overset{def}{=}$

$$\bigwedge \left(\begin{array}{l} \mathsf{sto}(loc) = l_0 \Rightarrow \bigvee \begin{pmatrix} \ell = \ell_r \wedge \mathsf{sto}' = \mathsf{sto}[\mathsf{sto}(x) + 1/x] \\ \ell = \ell_q \wedge \mathsf{sto}' = \mathsf{sto}[l_1/loc] \end{pmatrix} \\ \mathsf{sto}(loc) = l_1 \Rightarrow \bigvee \begin{pmatrix} \ell = \ell_p \wedge \mathsf{sto}' = \mathsf{sto}[\mathsf{sto}(x) + \mathsf{sto}(y)/y] \\ \ell = \ell_q \wedge \mathsf{sto}' = \mathsf{sto}[\mathsf{sto}(y) + 2/x][l_0/loc] \end{pmatrix} \end{array} \right)$$

- $T \overset{def}{=} \{tr_1, tr_2, tr_3, tr_4\}$
- $\mathsf{lbl}(tr_1) \overset{def}{=} \ell_r \mid \mathsf{lbl}(tr_2) \overset{def}{=} \ell_q \mid \mathsf{lbl}(tr_3) \overset{def}{=} \ell_p \mid \mathsf{lbl}(tr_4) \overset{def}{=} \ell_q$
- $\mathcal{R} \overset{def}{=} \{(tr_i, tr_i) \mid 1 \leq i \leq 4\} \cup \{(tr_2, tr_1), (tr_3, tr_4), (tr_4, tr_3)\}$
- $I_{\ell_p} \overset{def}{=} [1, 3[, I_{\ell_q} \overset{def}{=}]3, \infty[$ and $I_{\ell_r} \overset{def}{=} [1, 1]$.

Remarks. Because of the functional relation from T to L, we consider (by abuse of notation) in the remaining that, for any TCTS, $\boldsymbol{next}$ is also defined over $S \times T \times S$. Furthermore, assuming a TCTS, $tcts \overset{def}{=} \langle L, T, S, \mathsf{lbl}, \boldsymbol{init}, \boldsymbol{next}, \mathcal{R}, I\rangle$, a transition $tr \in T$ is *enabled* from a state $\mathsf{sto} \in S$, if there is a state $\mathsf{sto}' \in S$ such that the triplet $(\mathsf{sto}, tr, \mathsf{sto}')$ belongs to $\boldsymbol{next}$. Formally, we extend the predicate $\boldsymbol{enb}$ over $S \times T$ as $\boldsymbol{enb}_{\mathsf{sto}}(tr) \overset{def}{=} \exists \mathsf{sto}' \in S, \boldsymbol{next}(\mathsf{sto}, tr, \mathsf{sto}')$. Also, considering a transition $tr \in T$, we write I_{tr} instead of $I_{\mathsf{lbl}(tr)}$ in order to simplify notations.

As we have seen, TCTS is a mathematical model that allows to specify *time aspects* of real-time systems. We have also defined the composition of TCTS to model concurrent or distributed aspects of systems in the same formalism (see [12]). The semantics of this composition is based on the composition of LTS. However, due to lack of space, we choose to focus on the real-time aspects here.

4 Semantic Interpretations

We consider in this section two models to interpret TCTS w.r.t. time semantic aspects. The first one is TTS with explicit time steps together with time assumptions (see Definition 3). This model allows to reason over behaviors of a given TCTS using a branching time logic as [10] or (bi-)simulation relations [4].

Secondly, TCTS timing aspects are also interpreted as sets of *runs*, that describe the executions of a TCTS as sequences of timed events. This model allows to define and to prove the satisfaction of temporal properties expressed in a linear temporal logic [21] by a given TCTS.

4.1 Semantics of TCTSs in Terms of TTSs

Now, we want to give the semantics of TCTS in terms of TTS, interpreting time constraints by timed transitions. The purpose, among others, is to be able to reason on TCTS*s* at the TTS level.

So, given a TCTS, $tcts \overset{def}{=} \langle L, T, S, \mathsf{lbl}, \boldsymbol{init}, \boldsymbol{next}, \mathcal{R}, I\rangle$, we define a corresponding TTS through the semantics function $[\![\ ]\!]$: TCTS $\rightarrow$ TTS, such that $[\![tcts]\!] = \langle L \cup \mathbb{R}^+, \mathcal{S}, \boldsymbol{init}, \boldsymbol{t\_next}\rangle \in$ TTS, where :

- a state of $s \in \mathcal{S}$ is a pair over $S \times (T \rightarrow \mathbb{R}^+)$, such that $s \overset{def}{=} \langle \mathsf{sto}, \mathsf{clk}\rangle$ with :
  - ◇ $\mathsf{sto}_s \in S$ is a store of the underlying TCTS.
  - ◇ $\mathsf{clk}_s : T \rightarrow \mathbb{R}^+$ associates to every transition $tr \in T$ an *explicit clock* that is needed to count elapsed time since tr is enabled.
  - ◇ and for every $tr \in T$, if $\boldsymbol{enb}_{\mathsf{sto}_s}(tr)$ then $\mathsf{clk}_s(tr) \in \overleftarrow{I_{tr}}$
- $\boldsymbol{t\_next}$ is a predicate over $\mathcal{S} \times (L \cup \mathbb{R}^+) \times \mathcal{S}$ defined as :

$$\forall s \in \mathcal{S}, \forall s' \in \mathcal{S}, \forall \ell \in L \cup \mathbb{R}^+, \ \bigwedge \begin{pmatrix} \ell \in L \Rightarrow \boldsymbol{discrete}(s, \ell, s') \\ \ell \in \mathbb{R}^+ \Rightarrow \boldsymbol{delay}(s, \ell, s') \end{pmatrix}, \text{ with :}$$

◇ $\forall s \in \mathcal{S}, \forall s' \in \mathcal{S}, \forall \ell \in L,\ \boldsymbol{discrete}(s, \ell, s') \overset{def}{=} \exists tr \in T, \mathsf{lbl}(tr) = \ell$ and

$$\boldsymbol{next}(\mathsf{sto}_s, tr, \mathsf{sto}_{s'}) \text{ and } \mathsf{clk}_s(tr) \in \overrightarrow{I_{tr}}$$
$$\text{and } \forall tr', \mathsf{clk}_{s'}(tr') = \begin{cases} 0 \text{ if } \neg \boldsymbol{enb}_{\mathsf{sto}_s}(tr') \vee (tr, tr') \in \mathcal{R} \\ \mathsf{clk}_s(tr') \text{ else} \end{cases}$$

In other words, considering two timed states s and s' of $[\![tcts]\!]$, the label of a discrete transition tr links them if (1) tr is a transition between sto_s and $\mathsf{sto}_{s'}$ in the source TCTS; (2) since enabled, tr is delayed enough time units (according to the specification I_{tr}); and (3) for every other enabled transition tr', its clock is preserved if and only if tr does not reset it (clocks are set to 0 for all other transitions, tr included since $(tr, tr) \in \mathcal{R}$).

◇ $\forall s \in \mathcal{S}, \forall s' \in \mathcal{S}, \forall \delta \in \mathbb{R}^+,\ \boldsymbol{delay}(s, \delta, s') \overset{def}{=} \forall tr \in T,$

$$\mathsf{clk}_{s'}(tr) = \mathsf{clk}_s(tr) + \delta$$
$$\text{and if } \boldsymbol{enb}_{\mathsf{sto}_s}(tr) \text{ then we must have } \mathsf{clk}_s(tr) + \delta \in \overleftarrow{I_{tr}}$$
$$\text{and } \mathsf{sto}_{s'} = \mathsf{sto}_s$$

In other words, considering two timed states s and s' of $[\![tcts]\!]$, time passes (δ units) between them if for all transitions tr of $tcts$, (1) its clock goes δ time units from s to s' (2) after being enabled δ more units time, the maximal delay to take tr is not reached; and (3) stores are the same.

Theorem 1. $\forall tcts$: TCTS, $[\![tcts]\!]$: TTS.

The proof consists in proving for any $tcts \in$ TCTS, that its semantic interpretation, $[\![tcts]\!]$, satisfies the four properties of TTS*s* (see Definition 3).

4.2 Semantics of TCTSs in Terms of Timed Runs

The behavior of a TCTS can be expressed by a set of *runs* (see Definition 5) in order to reason about real-time systems using temporal logics. Here, according to the definition of TCTS with the set $\mathcal{R}$, we need to extend TES*s* of runs in order to deal with resetting of clocks. Runs of a TCTS are now sequences of TES*s* as in Definition 8, but with an additional predicate on the names of transitions.

Definition 8. *Assuming a* TCTS*, $tcts \overset{def}{=} \langle L, T, S, \mathsf{lbl}, \boldsymbol{init}, \boldsymbol{next}, \mathcal{R}, I\rangle$, a* timed execution state *of tcts, s, is now a 4-tuple $\langle \mathsf{sto}, \mathsf{evt}, \mathsf{now}, \sigma\text{-}\boldsymbol{reset}\rangle$ where :*
- $\mathsf{sto}_s \in S$*, is the state (or store) reached in s*
- $\mathsf{evt}_s \in L \cup \{start\} \cup \{div\}$*, is the event by which s is attained.*
- $\mathsf{now}_s \in \mathbb{R}^+$ *gives the global time in s.*
- $\sigma\text{-}\boldsymbol{reset}_s$ *is a function from T to $\mathbb{B}$ (or a predicate over T) that indicates which transitions have had their* implicit clock *reset when s has been reached.*

Predicates. Assume a given run σ at some position i, $\sigma\text{-}\boldsymbol{enb}_i(tr)$ means that the transition $tr \in T$ may be taken at i and $\boldsymbol{taken}_i(tr)$ means that tr is taken at i. Formally :
- $\sigma\text{-}\boldsymbol{enb}_i(tr) \overset{def}{=} \exists \mathsf{sto} \in S, \boldsymbol{next}(\mathsf{sto}_i, tr, \mathsf{sto})$
- $\boldsymbol{taken}_i(tr) \overset{def}{=} \bigwedge \begin{pmatrix} \mathsf{evt}_{i+1} = \mathsf{lbl}(tr) \\ \boldsymbol{next}(\mathsf{sto}_i, tr, \mathsf{sto}_{i+1}) \\ \forall tr', \sigma\text{-}\boldsymbol{reset}_{i+1}(tr') \Leftrightarrow (tr, tr') \in \mathcal{R} \end{pmatrix}$

Remark. Here, the predicate $\boldsymbol{taken}()$ is more sophisticated than in Section 2.3 because it deals with T and no more with L. Indeed, the first condition of the conjunction above is not enough because two transitions of T might satisfy it.

Run-Semantics of a TCTS. Now, we are able to express the semantic interpretation of a TCTS in terms of runs, as we did in Section 2.3 with TTS. Consider a TCTS, $tcts \overset{def}{=} \langle L, T, S, \mathsf{lbl}, \boldsymbol{init}, \boldsymbol{next}, \mathcal{R}, I\rangle$, we define below the set of well formed runs that we call $\Sigma(tcts)$. Every run σ of $\Sigma(tcts)$, which expresses a behavior of $tcts$, has to fulfill the four following conditions :

Initial timed execution state condition : $\boldsymbol{init}(\mathsf{sto}_0) \wedge \mathsf{now}_0 \in \overrightarrow{I_{\mathsf{evt}_1}}$

In other words, any run must start from an initial state of its underlying TCTS and the taken transition at position 0 has been delayed enough.

Next timed execution state condition :

$\forall i \in \mathbb{N}$, either $\exists tr \in T$ such that $\boldsymbol{taken}_i(tr)$ (a discrete transition is taken)
or $\neg\exists tr \in T$ such that $\sigma\text{-}\boldsymbol{enb}_i(tr)$ and $\mathsf{sto}_i = \mathsf{sto}_{i+1} = \mathsf{sto}_{i+2} = ... \wedge$
$\sigma_i \xrightarrow{div} \sigma_{i+1} \xrightarrow{div} ...$ (time diverging)

In other words, a next timed execution state can be reached from a state at position i of σ if either a discrete transition can be taken or a deadlock is met and time goes to infinity (the tag event *div* was introduced especially for this purpose in order to distinguish deadlock from the identity transition).

Lower bound condition : $\forall tr \in T$, and $\forall j \in \mathbb{N}$, if $\boldsymbol{taken}_j(tr)$ then

$$\begin{aligned}&\mathsf{now}_j \in \overrightarrow{I_{tr}} \text{ and}\\ &\forall i \in \mathbb{N} \text{ such that } i < j,\\ &\quad \text{if } \mathsf{now}_j - \mathsf{now}_i \notin \overrightarrow{I_{tr}} \text{ then } \sigma\text{-}\boldsymbol{enb}_i(tr) \wedge \neg\sigma\text{-}\boldsymbol{reset}_{i+1}(tr)\end{aligned}$$

In other words, a transition *tr* can be taken only after being continuously enabled for enough (means that it is delayed for at least the lower bound of I_{tr}) time units without being reset by the execution of another transition. The condition $\mathsf{now}_j \in \overrightarrow{I_{tr}}$ is required in the case where *tr* is enabled since the start of the execution. Indeed, in this case the premise $\mathsf{now}_j - \mathsf{now}_i \notin \overrightarrow{I_{tr}}$ is useless and we need to know that *tr* has been continuously enabled even before the position σ_0. It seems that this case has been omitted in [16].

Upper bound condition : $\forall tr \in T$ and $\forall i \in \mathbb{N}$,

$$\begin{aligned}&\exists j \in \mathbb{N} \text{ such that } i \leq j \text{ and } \mathsf{now}_j - \mathsf{now}_i \in \overleftarrow{I_{tr}}\\ &\text{and if } \sigma\text{-}\boldsymbol{enb}_j(tr) \text{ then } \boldsymbol{taken}_j(tr) \vee \sigma\text{-}\boldsymbol{reset}_j(tr)\end{aligned}$$

In other words, once enabled, a transition *tr* is not delayed for too long or it has been reset meanwhile. A transition cannot be continuously enabled for more than the upper bound of I_{tr} without being taken or reset.

4.3 Soundness of the Semantic Interpretations

Now we want to establish the soundness of our semantic interpretations by proving that the runs built straight from a given TCTS are the same that those built from its semantic interpretation in terms of a TTS. To do so, we must first define a semantic interpretation from timed execution state associated to TCTS*s* to the ones associated to their corresponding TTS*s*. So, we introduce the function γ from $\Sigma(tcts)$ to $\Sigma([\![tcts]\!])$, where states are in $\mathcal{S} \overset{def}{=} S \times (T \to \mathbb{R}^+)$ (see Section 4.1), *i.e.* every state associates a clock to every transition.

Definition 9. $\forall tcts : \text{TCTS}, \forall \sigma \in \Sigma(tcts), \forall i \in \mathbb{N},\ \gamma(\sigma_i) \overset{def}{=}$

$$\begin{cases}\langle\langle \mathit{sto}_0, (\lambda tr \in T.\ \delta^0_\sigma)\rangle, start, \mathit{now}_0\rangle \ \text{ if } i = 0,\\[4pt] \left\langle\left\langle \begin{array}{l}\mathit{sto}_i,\\ \lambda tr \in T.\ \begin{pmatrix}\delta \text{ if } \neg\sigma\text{-}\boldsymbol{enb}_{i-1}(tr) \text{ or } \sigma\text{-}\boldsymbol{reset}_i(tr)\\ \mathit{clk}_{\gamma(\sigma_{i-1})}(tr) + \delta \text{ otherwise}\end{pmatrix}\end{array}\right\rangle, \mathit{evt}_i, \mathit{now}_i\right\rangle \text{ else}\end{cases}$$

where $\delta \overset{def}{=} (\mathit{now}_i - \mathit{now}_{i-1})$ *and* δ^0_σ *is the initial delay of* σ*, which is* now_0*.*

Then, we define the semantics interpretation of runs of TCTS*s in terms of runs of corresponding* TTS*s as* $\forall tcts \in \text{TCTS}, \forall \sigma \in \Sigma(tcts),\ \gamma(\sigma) \overset{def}{=} \forall i \in \mathbb{N}, \gamma(\sigma_i)$.

Theorem 2. $\forall tcts : \text{TCTS}, \forall \sigma : \mathbb{N} \to \text{TES},\ \sigma \in \Sigma(tcts) \Leftrightarrow \gamma(\sigma) \in \Sigma([\![tcts]\!])$.

5 Mechanizing and Reasoning with a Proof Assistant

We explain in this section *why* we have implemented during the last two years the formalization work presented in this paper in an interactive theorem prover based on type-theory. First of all, the purpose is to validate with a very high level of confidence all the theorems about semantic interpretations of TCTS that we have presented previously. Moreover, using formal fundational proofs can also be useful to go beyond model-checking techniques and capabilities [13], when it is needed for example to check parameterized systems. To make that practical, we have *deeply embedded* in Coq a *timed linear temporal logic*[4] (called timed-SELTL) which is an extension of the State/Event linear temporal logic (SE-LTL) [9] since it embeds time intervals within temporal operators, together with a *proof methodology* dedicated to the *formal certification* of real-time systems.

Definition 10. *A timed-SELTL formula f is defined inductively as follows (where φ_S is a predicate over S, e is label of L and I is a time interval) :*

$$f ::= \varphi_S \mid e \mid \dot\neg\, f \mid f \,\dot\wedge\, f \mid f\, \mathcal{U}_I f$$

Remarks. As usual, we define the three temporal operators $\Diamond$ (eventually), $\Box$ (always) and $\mathcal{W}$ (weak until) as $\Diamond_I f \overset{def}{=} \top\, \mathcal{U}_I f$, $\Box_I f \overset{def}{=} \neg(\top\, \mathcal{U}_I \neg f)$ and $f_1\, \mathcal{W}_I f_2 \overset{def}{=} (f_1\, \mathcal{U}_I f_2)\, \dot\vee\, (\Box_{\overleftarrow{I}} f_1)$, where $\mathcal{U}_I$ is the *until operator* constrained with a time interval. We introduce as well the connectors $\dot\Rightarrow$ and $\dot\vee$. The *semantics* of this logic is defined over our formalization of runs and its implementation is available in our Coq framework.

Then, in order to reason and prove the correctness of real-time specifications modelled as TCTS*s*, we have *embedded* in our Coq framework a proof methodology very close to the one proposed in [15], based on the *same class of real-time properties* they consider (bounded response, bounded unless and bounded invariance properties) expressed as timed-SELTL formulæ. To do so, we have established *several* Coq *theorems* such as the two given below, where p, q, r and Φ are predicates over S. These theorems provide efficient proof rules since they enable us to certify in Coq real-time specifications expressed as timed-SELTL formulæ *over runs* (their conclusion), but *reasoning locally* (thanks to their premises) on TCTS*s* with Hoare triple and logical formulae. We have also proved *transitivity* and *disjunction* lemmas which are essential to make the method usable.

$$\boxed{\begin{array}{l}
(1)\ \forall s, p(s) \Rightarrow \neg \boldsymbol{enb}_s(tr) \\
(2)\ \forall s, p(s) \Rightarrow \Phi(s) \\
(3)\ \forall tr' \neq tr, \{\Phi\}\ tr'\ \{\Phi\} \\
(4)\ \forall s, \Phi(s) \Rightarrow q(s) \\
(5)\ \forall s, \Phi(s) \wedge \boldsymbol{enb}_s(tr) \Rightarrow r(s) \\
\hline
\qquad \forall\sigma, \forall i,\ \sigma \models_i p \,\dot\Rightarrow\, q\ \mathcal{W}_{\overrightarrow{I_{tr}}}\ r
\end{array}}$$

$$\boxed{\begin{array}{l}
(1)\ \forall s, p(s) \Rightarrow \Phi(s) \vee q(s) \\
(2)\ \forall s, \Phi(s) \Rightarrow \boldsymbol{enb}_s(tr) \\
(3)\ \forall s, \forall tr' \neq tr, \Phi(s) \wedge \boldsymbol{enb}_s(tr') \\
\qquad \Rightarrow \min(I_{tr'}) > 0 \wedge (tr', tr) \notin \mathcal{R} \\
(4)\ \forall tr' \neq tr, \{\Phi\}\ tr'\ \{\Phi \vee q\} \\
(5)\ \{\Phi\}\ tr\ \{q\} \wedge \max(I_{tr}) < \infty \\
\hline
\qquad \forall\sigma, \forall i,\ \sigma \models_i p \,\dot\Rightarrow\, \Diamond_{\overleftarrow{I_{tr}}}\, q
\end{array}}$$

6 Conclusions

We have presented a mathematical model for specifying and reasoning over real-time systems. This model enables us to specify some very subtle timed-semantic

[4] Carlos D.Luna has defined in Coq the TCTL tree logic : `http://coq.inria.fr/pylons/contribs/view/CTLTCTL/v8.4`

differences of fine-grained constructs (see Section 1.1) allowed by modern real-time component-based languages such as FIACRE. Also, we provide logical and functional definitions to give semantic interpretations of real-time systems specified as TCTSs in terms of well known formalisms that allow to reason over their timing and temporal aspects. All definitions and theorems presented in this paper have been fully formalized and established in the COQ proof assistant[5].

We stress also in this conclusion that we do not address the whole FIACRE language [6] semantics in this paper. However, we consider the introduced model of TCTS (together with [12]) as the *semantic kernel* of FIACRE and the cornerstone of future research and developments. Transition systems with *real-time constraints and priorities* are very complicated and reasoning about them is tedious. But on the other hand such systems are widely used in real-life embedded systems, especially in spatial and avionics domains, which is a good motivator for developing a fully mechanized framework. This makes the presented work a very useful starting point for further theoretical or applied developments.

Future work will consist in certifying through the use of COQ some *existing patterns* used in the design of avionic embedded applications, such as the *periodic controller* presented in [8]. Also, an application of this work is to certify within COQ the translation of timed temporal formulæ into FIACRE observers [1]. Another envisioned application of our semantic framework concerns the verification of transformations between timed formalisms such as the one presented in [5], or model simplification, as for example, the flattening operator of component-based languages. At last, we are convinced that this work together with [12] could be a good basis for the certified compilation [19] of the FIACRE language.

References

1. Abid, N., Zilio, S.D., Botlan, D.L.: A Verified Approach for Checking Real-Time Specification Patterns. In: Proceedings of VeCos (2012)
2. Arnold, A.: Finite Transition Systems - Semantics of Communicating Systems. Prentice Hall international series in computer science. Prentice Hall (1994)
3. Basu, A., Bozga, M., Sifakis, J.: Modeling Heterogeneous Real-time Components in BIP. In: SEFM, pp. 3–12 (2006)
4. Bérard, B., Cassez, F., Haddad, S., Lime, D., Roux, O.H.: Comparison of Different Semantics for Time Petri Nets. In: Peled, D.A., Tsay, Y.-K. (eds.) ATVA 2005. LNCS, vol. 3707, pp. 293–307. Springer, Heidelberg (2005)
5. Bérard, B., Cassez, F., Haddad, S., Lime, D., Roux, O.H.: Comparison of the Expressiveness of Timed Automata and Time Petri Nets. In: Pettersson, P., Yi, W. (eds.) FORMATS 2005. LNCS, vol. 3829, pp. 211–225. Springer, Heidelberg (2005)
6. Berthomieu, B., Bodeveix, J.-P., Farail, P., Filali, M., Garavel, H., Gaufillet, P., Lang, F., Vernadat, F.: Fiacre: an Intermediate Language for Model Verification in the Topcased Environment. In: ERTS 2008 (2008)
7. Bertot, Y., Castéran, P.: Interactive Theorem Proving and Program Development (Coq'Art: The Calculus of Inductive Constructions). Texts in Theoretical Computer Science. Springer (2004)

[5] Available at `http://www.irit.fr/~Manuel.Garnacho/Coq/FORMATS13`.

8. Bodeveix, J.-P., Filali, M., Garnacho, M., Spadotti, R., Yang, Z.: On the Mechanization of an AADL Subset. Science of Computer Programming: special issue on Architecture Design Language (submitted, 2013)
9. Chaki, S., Clarke, E.M., Ouaknine, J., Sharygina, N., Sinha, N.: State/Event-Based Software Model Checking. In: Boiten, E.A., Derrick, J., Smith, G.P. (eds.) IFM 2004. LNCS, vol. 2999, pp. 128–147. Springer, Heidelberg (2004)
10. Emerson, E.A., Halpern, J.Y.: Decision Procedures and Expressiveness in the Temporal Logic of Branching Time. In: STOC, pp. 169–180 (1982)
11. The FIACRE Specification Language for Real-Time Concurrent Systems, `http://projects.laas.fr/fiacre/`
12. Garnacho, M., Bodeveix, J.-P., Filali, M.: Mechanized Semantics of Concurrent Systems with Priorities. IRIT Research Report–2013-16–FR (2013), `http://www.irit.fr/~Manuel.Garnacho/Publications/MechPrio.pdf`
13. Geuvers, H., Koprowski, A., Synek, D., van der Weegen, E.: Automated Machine-Checked Hybrid System Safety Proofs. In: Kaufmann, M., Paulson, L.C. (eds.) ITP 2010. LNCS, vol. 6172, pp. 259–274. Springer, Heidelberg (2010)
14. Hale, R., Cardell-Oliver, R., Herbert, J.: An Embedding of Timed Transition Systems in HOL. FMSD 3(1/2), 151–174 (1993)
15. Henzinger, T.A., Manna, Z., Pnueli, A.: Temporal Proof Methodologies for Real-time Systems. In: POPL, pp. 353–366 (1991)
16. Henzinger, T.A., Manna, Z., Pnueli, A.: Timed Transition Systems. In: Huizing, C., de Bakker, J.W., Rozenberg, G., de Roever, W.-P. (eds.) REX 1991. LNCS, vol. 600, pp. 226–251. Springer, Heidelberg (1992)
17. Hoare, C.A.R.: Communicating Sequential Processes. Prentice-Hall (1985)
18. The ISABELLE System, `http://isabelle.in.tum.de/`
19. Leroy, X.: Formal certification of a compiler back-end, or: programming a compiler with a proof assistant. In: 33rd Symposium Principles of Programming Languages, pp. 42–54. ACM Press (2006)
20. Paulin-Mohring, C.: Modelisation of Timed Automata in Coq. In: Kobayashi, N., Babu, C. S. (eds.) TACS 2001. LNCS, vol. 2215, pp. 298–315. Springer, Heidelberg (2001)
21. Pnueli, A.: The Temporal Logic of Programs. In: FOCS, pp. 46–57 (1977)
22. Rushby, J.: Mechanized Formal Methods: Progress and Prospects. In: Chandru, V., Vinay, V. (eds.) FSTTCS 1996. LNCS, vol. 1180, pp. 43–51. Springer, Heidelberg (1996)

Quantitative Analysis of AODV and Its Variants on Dynamic Topologies Using Statistical Model Checking

Peter Höfner[1,4] and Maryam Kamali[2,3]

[1] NICTA, Australia
[2] Turku Centre for Computer Science (TUCS), Finland
[3] Åbo Akademi University, Finland
[4] University of New South Wales, Australia

Abstract. Wireless Mesh Networks (WMNs) are self-organising ad-hoc networks that support broadband communication. Due to changes in the topology, route discovery and maintenance play a crucial role in the reliability and the performance of such networks. Formal analysis of WMNs using exhaustive model checking techniques is often not feasible: network size (up to hundreds of nodes) and topology changes yield state-space explosion. Statistical Model Checking, however, can overcome this problem and allows a quantitative analysis.

In this paper we illustrate this by a careful analysis of the Ad hoc On-demand Distance Vector (AODV) protocol. We show that some optional features of AODV are not useful, and that AODV shows unexpected behaviour—yielding a high probability of route discovery failure.

1 Introduction

Route finding and route maintenance are critical for the performance of networks. Efficient routing algorithms become even more important when mobility of network nodes lead to highly dynamic and unpredictable environments. The Ad hoc On-Demand Distance Vector (AODV) routing protocol [15] is such an algorithm. It is widely used and particularly designed for Wireless Mesh Networks (WMNs), self-organising ad-hoc networks that support broadband communication.

Formal analysis of routing protocols is one way to systematically analyse protocols for flaws and to present counterexamples to diagnose them. It has been used in locating problems in automatic route-finding protocols, e.g. [1,4]. These analyses are performed on tiny static networks (up to 5 nodes). However, formal validation of protocols for WMNs remains a challenging task: network size (usually dozens, sometimes even hundreds of nodes) and topology changes yield an explosion in the state space, which makes exhaustive model checking (MC) techniques infeasible. Another limitation of MC is that a quantitative analysis is often not possible: finding a shortcoming in a protocol is great but does not show how often the shortcoming actually occurs.

Statistical model checking (SMC) [19,18] is a complementary approach that can overcome these problems. It combines ideas of model checking and simulation with the aim of supporting quantitative analysis as well as addressing the size barrier. SMC trades certainty for approximation, using Monte Carlo style sampling, and hypothesis testing to interpret the results.

In this paper we demonstrate that SMC can be used for formal reasoning of routing protocols in WMNs. We perform a careful analysis of different versions of the AODV protocol. In particular, we analyse how dynamic topologies can affect the protocol behaviour. In other words, we analyse the performance of the protocol while the network topology evolves. We show that some optional features provided by AODV should be avoided since they affect the performance of the protocol. Moreover, we show that in some scenarios the behaviour of AODV is not as intended yielding a high probability of route discovery failure. When possible we suggest improvements of the protocol.

The paper is organised as follows: in Sect. 2 we give an overview of AODV, present optional features such as the resending of route requests, and sketch the encoding of AODV in SMC-Uppaal, the statistical extension of Uppaal. In Sect. 3 we describe the mobility model, which is used for our analysis of AODV. Sect. 4 discusses the experiments performed, the main contribution of this paper: (i) We show that a single mobile node can have a massive impact on the success of route discovery. Moreover we show that some options of AODV should not be used in combination, unless the protocol specification is adapted (changed). (ii) A second category of experiments reveals a surprising observation: adding "noise" (for example an additional data packet) to a network can increase the success of route discovery. (iii) The third category discusses the consequences of different speeds of mobile nodes. The paper closes with a discussion of related work in Sect. 5 and a short outlook in Sect. 6.

2 AODV, Its Variants and Their Uppaal Models

2.1 The Basic Model

The AODV routing protocol [17] is a widely used routing protocol, particularly tailored for WMNs. It is currently standardised by the IETF MANET working group and forms the basis of new WMN routing protocols, including HWMP in the upcoming IEEE 802.11s wireless mesh network standard [12].

AODV is a reactive protocol, meaning that a route discovery process is only initiated when a node S in the network has to send data to a destination D for which it does not have a valid entry in its own routing table. The route discovery process starts with node S broadcasting a route request (RREQ) message, which is received by all nodes within S's transmission range. If a node, which is different to the destination, receives a RREQ message and does not have a valid entry for the destination in its routing table, the request is forwarded by re-broadcasting the RREQ message. During this forwarding process, the intermediate node updates its routing table and adds a "reverse route" entry with destination S into its routing table, indicating via which next hop the node S can be reached, and the distance in number of hops. To avoid unnecessary message sending each RREQ has a unique identifier which allows nodes to ignore RREQ messages that they have handled before.

As soon as the RREQ is received by the destination itself or by a node that knows a valid route to the destination, a route reply (RREP) is generated. In contrast to RREQ messages, a RREP message is unicast, i.e., it is only sent to a single node, not to all nodes within transmission range. The RREP message travels from its generator (either D or an intermediate node knowing a route to D) back along the established route towards S, the originator of the RREQ message. All intermediate nodes on the selected route will process the RREP message and, in most cases, forward it towards S. However, there are scenarios where RREP message are discarded (see below). By passing a RREP message towards S, a node adds a "forward route" entry to its routing table.

The route discovery process is completed when the RREP reaches node S; an end-to-end route from S to D has been established, and data packets can start to flow. If any link breaks down (e.g. by a node moving out of transmission range), the node that detects the break broadcasts a route error (RERR) message.[1] All notified nodes invalidate their routing table entries that use the broken link and forward the RERR message if necessary.

Full details can be found in RFC 3561 [15], the de facto standard of AODV.

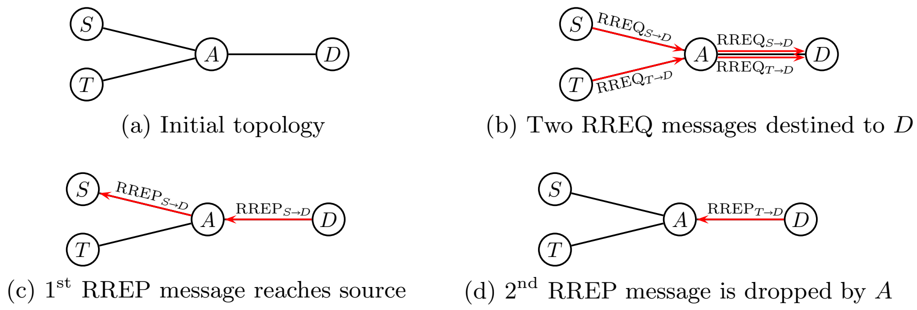
2.2 Variants of AODV

The specification of AODV [15] offers optional features, which yield different variants of the routing protocol. One aim of this paper is to compare versions of AODV with different features turned on.

Destination Only (D) Flag. Each RREQ message contains a field called *destination only flag*. If the value of this Boolean flag is `true`, it indicates that only the destination node is allowed to respond to this RREQ. That means that the RREQ travels through the entire network until it reaches the destination. Only then a reply is sent back. By this a *bi-directional link* between the source and the destination is (usually) established.

Resending a Route Request. The basic version of AODV, as presented in the previous section, suffers the problem that some routes, although they do exist, are not discovered. Reasons for route discovery failure can be message transmission failures (the receiver of a unicast message has moved out of transmission range) or the dropping of RREP messages, that should be forwarded. With respect to the latter, the problem is that a node only forwards a RREP message if it is not the originator node, *and* it has created or updated a routing table entry to the destination node described in the RREP message. [15]

[1] Following the RFC, a node uses precursor lists to store those nodes that are interested in some particular routes—when sending an RERR message only those neighbours are informed. However, precursor lists do not contain all neighbours that are interested in a particular route (e.g. [8]); that is why we model an improved version of AODV where RERR messages are broadcast.



(a) Initial topology (b) Two RREQ messages destined to D

(c) 1st RREP message reaches source (d) 2nd RREP message is dropped by A

Fig. 1. Route Discovery Failure

An example for route discovery failure, taken from [10], is sketched in Fig. 1.[2] On the 4-node topology depicted in Part (a) nodes S and T, resp., initiate a route discovery process to search for a route to D. The messages travel through the network and reach the destination D (Part (b)). We assume that RREQ$_{S\to D}$, the request stemming from S, reaches node D first. In Part (c), D handles RREQ$_{S\to D}$, creates an entry for S in its routing table[3] and unicasts a RREP message to A. Node A updates its routing table (creates an entry for D) and forwards the message to the source S. In Part (d), D handles RREQ$_{T\to D}$, creates an entry for T in its routing table and unicasts a RREP message to A. Since RREP$_{T\to D}$ does not contain new information for A (a route to D is already known), node A does not update its routing table and, according to the specification, will not forward the RREP message to the source T. This leads to an unsuccessful route discovery process for node T.

The solution proposed by the RFC is to initiate a new route discovery process, if no route has been established 2 seconds after the first request was sent; the number of retries is flexible, but the specification recommends one retry only. In the example node T would initiate another route request; node A, which receives the RREQ message, will immediately unicast a RREP message back to T.

Local Repair. In case of a link break, the node upstream of that break can choose to repair the link locally if the destination was no farther away than a predefined number of hops (the number is specified by the user and often depends on the network size). When a node receives a RREP message or a data packet destined for a node for which it does not have a valid route, the node buffers the message and initiates a new route discovery process. As soon as a route has been re-established, the buffered message is sent.

2.3 Modelling AODV and Its Variants in Uppaal

Table 1 lists all variants of AODV that are modelled, analysed and compared in this paper. The analysis is performed by SMC-Uppaal, the statistical extension

[2] A similar example has been published at the IETF mailing list in 2004; `http://www.ietf.org/mail-archive/web/manet/current/msg05702.html`.

[3] Routing tables are not presented in the figure.

Table 1. Different Variants of AODV

name	**optional features**	**remark**
basic	none	follows description of Sect. 2.1
resend	resending RREQ	"standard" configuration of AODV
dflag	D-flag	the flag is set for all route discovery processes
dflag_res	D-flag and resending	this configuration has a flaw (see below)
dflag_res'	D-flag and resending	not following the RFC literally, but flaw fixed
repair	local repair	use local repair

of Uppaal [3]. The modelling language for SMC-Uppaal is the same as for "standard" Uppaal, namely networks of guarded, timed and probabilistic automata.

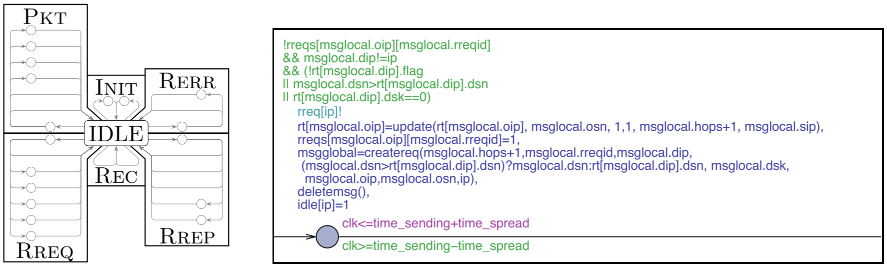
Our 6 models of (all variants of) AODV are based on a single untimed Uppaal model that was used to analyse some basic qualitative properties [7].[4] Since we are interested in a quantitative analysis of the protocol, the model had to be equipped with time and probability. The latter is needed to model dynamic topologies and mobile nodes. Hence, the (untimed) model was significantly redesigned and extended to include timing constraints on sending messages between nodes. Both the untimed and the timed model were systematically derived from an unambiguous process-algebraic model that models the intention of the RFC and does not contain contradictions. Communication between nodes had to be modelled so that the unicast behaviour of AODV was correctly rendered using SMC-Uppaal's (only) broadcast mechanism.

Each node of a network is modelled by two timed automata: the first models a message queue that buffers received messages, the other models the AODV routine. This main routine consists of ~ 20 locations, 1 clock measuring the sending time, and a complicated data structure with approx. 10 variables. The latter includes an array `rt` of length N modelling the routing table, where N is the number of nodes in the network. The overall structure of the main automaton is depicted in Fig. 2(a), it consists of 7 regions. If the automaton is in the region IDLE, which consists of one location only, then AODV does not perform any action in the moment and the automaton is ready to receive messages. This happens in REC if there is at least one message buffered. The regions RREQ, RREP, RERR and PKT perform actions depending on the type of the received message. RREQ for example handles route request messages. INIT initiates the transmission of data injected by the user as soon as the route is established.

Message handling often contains actions for updating the internal data (such as routing tables) and sending of a message. Fig. 2(b) gives an impression of such an update by showing a snippet of the automaton modelling the forwarding of a RREQ message.

Message sending is the only action that takes time: according to the specification of AODV [15], the most time consuming activity is the communication between nodes, which takes on average 40 milliseconds; all other times are marginal and assumed to be 0.

[4] Our models can be found at `http://www.hoefner-online.de/formats2013/`.



```
!rreqs[msglocal.oip][msglocal.rreqid]
&& msglocal.dip!=ip
&& (!rt[msglocal.dip].flag
|| msglocal.dsn>rt[msglocal.dip].dsn
|| rt[msglocal.dip].dsk==0)
   rreq[ip]!
   rt[msglocal.oip]=update(rt[msglocal.oip], msglocal.osn, 1,1, msglocal.hops+1, msglocal.sip),
   rreqs[msglocal.oip][msglocal.rreqid]=1,
   msgglobal=createreq(msglocal.hops+1,msglocal.rreqid,msglocal.dip,
    (msglocal.dsn>rt[msglocal.dip].dsn)?msglocal.dsn:rt[msglocal.dip].dsn, msglocal.dsk,
     msglocal.oip,msglocal.osn,ip),
   deletemsg(),
   idle[ip]=1
        clk<=time_sending+time_spread
        clk>=time_sending−time_spread
```

(a) Structure (b) Detail: updating data and forwarding a message

Fig. 2. Overall structure of the SMC-Uppaal model of AODV

Our models cover all core components of AODV. However, we encoded one main assumption: whenever a message is sent and the receiver of the message is within transmission range, the message will be received. In reality message loss during transmission happens regularly, for example due to communication failures or packet collisions. This loss could easily be modelled using Uppaal's broadcast mechanism in combination with probabilistic automata. However, this abstraction enables us to interpret a failure of guaranteed message delivery as an imperfection in the protocol, rather than as a result of a chosen formalism not allowing guaranteed delivery. Due to lack of space we cannot give more details about the modelling; more details about the model *basic* can be found in [11].

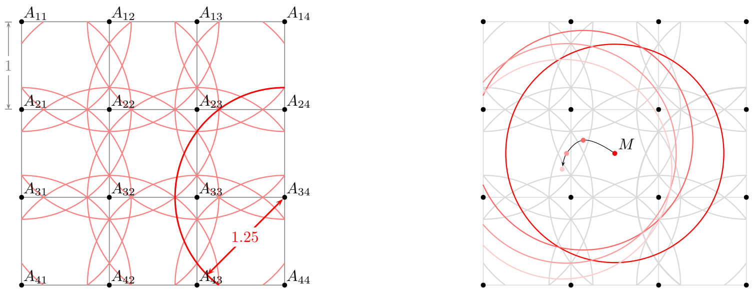
Next to the automata modelling the behaviour of the node, two additional automata are needed: the first is a scenario generator initiating the route discovery process, i.e., it forces one of the nodes to generate and broadcast a RREQ message. The second automaton models the mobility within the network.

3 Modelling Dynamic Topologies

To analyse quantitative properties of AODV and to compare different variants in a dynamic network, we use a *topology-based mobility model* [9]. It reflects the impact of mobility on the network topology and distinguishes static and mobile nodes; only connections to and from mobile nodes can change. Each movement is characterised either by adding a new link to or by removing an existing link from the *connectivity graph*. Whenever a mobile node M enters the transmission range of a node A, a new link is established between M and A. If M leaves the transmission range, the link between these two nodes is removed.

To decrease the number of possible topology changes due to a large number of mobile nodes, we set up the topology as follows: the network consists of 16 static and one mobile node.[5] The static nodes form a 2-dimensional rectangular grid with grid size 1, i.e. the smallest distance between two nodes is 1 unit (cf. Fig. 3(a)); the transmission range is set to 1.25. In reality, 1 unit might correspond to 100 metres, the transmission range to 125 m, a realistic value.

[5] We also performed experiments with more than one mobile node; but these experiments do not show new (odd) results.



(a) grid with 16 nodes (static topology) (b) node moving between zones (transmission ranges are indicated)

Fig. 3. Topology-based mobility model

The transmission ranges of the nodes A_{ij} $(1 \leq i, j \leq 4)$ split the grid into 102 different zones. The different zones are shown in Fig. 3(a). When a mobile node M moves within a zone, the exact position of the node does not matter, since it does not enter or leave the transmission range of any node—the connectivity graph stays the same. For example any node that is within the central zone is connected to nodes A_{22}, A_{23}, A_{32} and A_{33} (cf. Fig. 3(b)) When M transits the border of a zone, it triggers a network topology change. Only the change of the connectivity graph is considered, other details such as the exact direction and angle of transmitting are not needed for characterising the dynamic network. In the example given in Fig. 3(b), M moves to the left and enters the transmission range of A_{21}. Next, in fainter colours, the node enters transmission range of A_{31} and leaves the range of A_{23}.

The topology-based model captures the topology changes as a Markovian transition function $\texttt{prob}(T_1, T_2)$, that assigns to two topologies T_1 and T_2 a transition probability. The probability of moving from one zone to a neighbouring zone is based on the ratio of the length the two zones share compared to the overall border length of the zone in which the node is in. For instance, the probability of transiting from the central segment of the grid to any adjacent zone is $\frac{1}{8}$, due to equal segment lengths.

Our model sets the speed of the mobile node in such a way that the node has to change zones every 35–45 time units, where the probability to leave the zone at time t is equally distributed in the interval.

The zones can be grouped by their shapes; each shape forms an equivalence class. The Uppaal model reflects this observation. Each mobile node is modelled by a separate timed and probabilistic automaton; each location of the automaton characterises exactly one equivalence class. (See [9] for details.)

4 Experiments

Our experiments analyse the impact of mobile nodes and dynamic topologies on AODV; they are grouped into several categories: the first category analyses the probability of route establishment for a single route discovery process, i.e., an originator node `oip` is searching for a route to `dip`; the second category analyses the likelihood of route establishment between `oip` and `dip` when additional route discovery processes occur; the last category changes the speed of the mobile node.

Before discussing the experiments, we briefly describe some foundations of statistical model checking. SMC [19,18] combines ideas of model checking and simulation with the aim of supporting quantitative analysis as well as addressing the size barrier that prevents useful analysis of large models. By trading certainty for approximation, it uses Monte Carlo style sampling, and hypothesis testing to interpret the results. The sampling follows the probability distribution defined by the non-deterministic and probabilistic automata. Parameters setting thresholds on the probability of false negatives (α) and on probabilistic uncertainty (ε) can be used to specify the statistical confidence on the result. SMC-Uppaal computes the number of simulation runs needed by using Chernoff-Hoeffding bounds, which is independent of the size of the model; it generates an interval $[p-\varepsilon, p+\varepsilon]$ for estimating p, the probability of CTL-property ψ holding w.r.t. the underlying probability distribution.

For most of our experiments we use “only” a confidence level of 95% and allow a large probabilistic interval of 10%—this is the default setting of SMC-Uppaal and means that both α and ε are set to 5%. When using this set up, SMC-Uppaal simulates 738 runs to determine the probability of a property.

Experiments with $\alpha = \varepsilon = 1\%$ (26492 runs) are also feasible with a standard desktop machine, but require much more time. While an experiment using a confidence level of 95% takes only a couple of minutes; an experiment using a level of 99% takes more than 3 hours. We illustrate this by our first experiment.

4.1 Single Route Discovery Process

Our first experiment is based on 17 nodes; 16 forming a grid (Fig. 3(a)) and one mobile node M which is located in the middle of the grid at the beginning of the experiment. After a delay between 140 and 160 time units (the time that the mobile node needs to perform four movements) the first RREQ message is broadcast. By this delay, the location of M is random at the point the route discovery process is initiated.

In the experiment A_{11} searches for a route to A_{44}, that means it initiates a route discovery process. We are interested whether (and at which time) A_{11} establishes a route to A_{44}. In Uppaal syntax this reachability property is

```
Pr[<=2000](<>A11.rt[A44].nhop!=0) .                (1)
```

Checking this query determines the probability (`Pr`) satisfying the CTL-path expression `<>(A11.rt[A44].nhop!=0)` with a time bound of 2000 time units; we choose this bound as a conservative upper bound to ensure that the analyser explores paths to a depth where the protocol is guaranteed to have terminated. The term `ip.rt[dip]` refers to a route to `dip` stored inside the routing table of node `ip`. Whenever the next hop `nhop` is set ($\neq 0$), a route has been established.

The results are summarised in Table 2. From an experimental point of view, the table shows that a confidence level of 99% does not yield much better results

Table 2. Single Route Discovery Ratio (confidence level 95% and 99%)[6]

	model	**conf. level**	**probability route discovery**	**time route discovery**	**Uppaal running time**
1.	*basic*	95%	[55.4336, 65.4336]	595.67	$4m\,18s$
2.	*basic*	99%	[59.7806, 61.7806]	597.52	$157m\,44s$
3.	*resend*	95%	[95.00, 100.00]	847.04	$6m\,03s$
4.	*resend*	99%	[98.9623, 100.00]	836.87	$209m\,43s$
5.	*dflag*	95%	[54.4851, 64.4851]	597.50	$4m\,52s$
6.	*dflag*	99%	[59.5655, 61.5655]	597.63	$164m\,21s$
7.	*dflag_res*	95%	[64.5122, 74.5122]	698.70	$7m\,47s$
8.	*dflag_res*	99%	[68.2133, 70.2133]	688.79	$249m\,12s$
9.	*dflag_res'*	95%	[81.3144, 91.3144]	822.89	$7m\,08s$
10.	*dflag_res'*	99%	[83.4104, 85.4104]	807.43	$230m\,57s$
11.	*repair*	95%	[59.7696, 69.7696]	607.86	$7m\,31s$
12.	*repair*	99%	[63.5742, 65.5742]	606.53	$165m\,11s$

than a confidence level of 95%; but the running times of Uppaal (last column) are much higher (in average by a factor of 33.6).

Next to the running times of Uppaal the table lists the model (first column), the probability of a successful route discovery (third column) and the average time needed to establish a route between A_{11} and A_{44} (fourth column). It is no surprise that the models *basic* and *dflag* yield the same results—in this setting they behave identically. Furthermore, it is obvious that the probability for successful route discovery increases when using the *resend* option, while at the same time the discovery time increases as well. However, the experiments reveal three surprising and unexpected observations concerning AODV.

Observation 1. A single mobile node can already have a massive impact on the success of route discovery. In our setting the probability of route discovery can decrease by about 40%.

Using the same setting without mobility (e.g., the mobile node does not exist or keeps sitting in the centre of the grid), the probability of route discovery success is 100%. The success rate in our experiment using AODV *basic*, is only 60.78 ±1% (Row 2 of Table 2). The setting of the experiment guarantees that the RREQ reaches the destination A_{44} and that A_{44} will

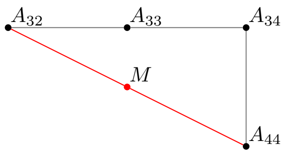


Fig. 4. Mobile node shortens distance

[6] We use a standard computer equipped with a 3.1 GHz Intel Pentium 5 CPU, 16 GB memory, running a Mac OS operating system. As SMC-tool, we use SMC-Uppaal, the Statistical extension of Uppaal (release 4.1.11) [3], which supports both timed and probabilistic systems. Timing aspects are heavily needed to model AODV (cf. Sect. 2); the topology-based mobility model relies on probabilistic choices to determine the movement of the node.

generate a route reply. It means that the route reply, which is unicast back via a previously established path gets lost. Since the experiment consists of a single request only, RREP messages are not dropped and situations as the one sketched in Fig. 1 cannot occur. As a consequence, failure in route discovery means that a RREP message could not be unicast, which means that the established route from A_{44} to A_{11} uses the mobile node.

At first glance it seems to be impossible that 40% of all established routes use the mobile node as intermediate hop. But a closer analysis on time interval when a route for A_{11} is discovered shows that this is in fact the case since a route via a mobile node can shorten the distance between originator and destination. In general, AODV prefers shorter routes, hence it would choose the route via the mobile node M. Fig. 4 illustrates how a mobile node decreases the distance between A_{32} and A_{44} from 3 hops to 2. The lesson learned is that static nodes should be set up in a way that it is unlikely for a mobile node to shorten the distance, or static and mobile nodes should be distinguished and routes via static nodes only should be preferred, even if they are longer.

Observation 2. The model *dflag_res* does not yield much improvement w.r.t. route discovery compared to *basic* and is much worse than using *resend* alone.

The chance that a route is established by the first route discovery process is around 60% (cf. *basic*). In case no route is established (chance $\sim 40\%$), a new request is issued; the chance that this second request yields a route establishment between A_{11} and A_{44} is again 60%. Putting these numbers together the success rate for *dflag_res* should be $0.6 + 0.4 \cdot 0.6 \approx 0.84 = 84\%$. Surprisingly, the probability determined by our experiments is only around 70% in case of *dflag_res* (Row 7 and 8 of Table 2). That means that many RREP messages are lost (using the same reasoning as before, no RREQ message is lost). The explanation lies in the RREP-forwarding mechanism of AODV. As explained in Sect. 2, RREP messages are not forwarded if they do not contain new information. Let us now assume that the first RREQ reaches the destination A_{44}, which unicasts a RREP message to the next hop on the route back to A_{11}, say to node A_{34}. This reply gets lost afterwards. Since the resend-option is set, the originator issues another request, which also reaches A_{44}. In case the route to A_{11} is not changed in A_{44}'s routing table, A_{44} sends another RREP message to A_{34}. This message does not contain new information and is dropped by the intermediate node.

To repair this flaw, we change the RREP-generation procedure. Whenever a RREP message is generated, a counter (the sequence number), which indicates the freshness of the message is incremented.[7] This change is implemented in *dflag_res'*; the evaluation results for this model are now as expected.

Observation 3. AODV's option of intermediate route reply should be used.

Let us have a look at the models *resend* and *dflag_res'*. The difference between the two models is that in the former model intermediate nodes are allowed to reply.

[7] In fact AODVv2 and LOADng, the successor protocols of AODV (still under development), implement exactly this variant.

Looking at the results, we notice a dramatic difference in the likelihood of route discovery. In the model *resend* the second request is followed by the generation of more than one RREP message. In fact, each node that established a route to A_{44} during the first RREQ-RREP-cycle (before the reply was lost), will generate a RREP message. Due to this, route establishment is guaranteed. In contrast, there is only one RREP message for each and every request in *dflag_res'*. This observation clearly indicates that intermediate route reply is a useful feature. Interestingly, there seems to be the tendency of preferring protocols without this feature: the two successors of AODV, AODVv2 [16] and LOADng [6] follow this philosophy and set the D-flag as default—if at all, they allow intermediate route reply as an optional feature.

Other Originator Nodes
The first set of experiments considered a route discovery from A_{11} to A_{44}, the largest distance a packet can travel in our set up. We expected to see the clearest results by using this distance. However, we also performed experiments with all other pairs of nodes. Fig. 5 summarises some results. It illustrates the probability of route-discovery (y-axis) depending on the distance between originator and destination (x-axis). Of course, the larger the distance between originator and destination, the smaller the chance of route establishment. Interestingly, there is a clear drop down at a distance of four nodes. It seems that from this point on resending guarantees the success. Moreover, the graph illustrates that exhaustive MC cannot help: MC is usually limited to topologies of up to 6 nodes, distances of 5 hops and more are not possible if one considers a non-linear topology.

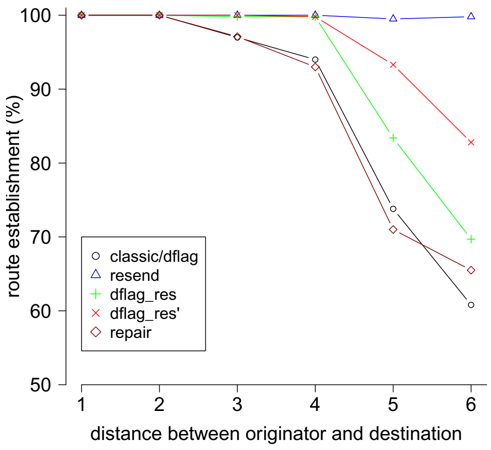


Fig. 5. Probability of route-establishment

4.2 Two Independent Route Discovery Processes

In order to evaluate the performance of variants of AODV under different network (traffic) load, we check the probability of route discovery when two route discovery processes are performed in parallel. For this second set of experiments, we are again interested in a route from A_{11} to A_{44}. However, shortly (35-45 milliseconds) after the packet is handed over to A_{11}, a second data packet is injected at another node, destined for some destination; in fact we did experiments for all destinations, but present only two observations—due to lack of space.

Observation 4. RREP messages are dropped more often than expected.

Table 3. Two route discovery processes looking for the same destination A_{44}

distance between orig.	**orginator of 2^{nd} request**	**class**	**probability (avg.)**
1	$\{A_{12}, A_{21}\}$	nodes at border	43.36%
2	$\{A_{13}, A_{31}\}$	nodes at border	17, 75%
	$\{A_{22}\}$	inner node	13.28%
3	$\{A_{14}, A_{41}\}$	nodes at border	43, 10%
	$\{A_{23}, A_{32}\}$	inner nodes	29.74%
4	$\{A_{24}, A_{42}\}$	nodes at border	71, 61%
	$\{A_{33}\}$	inner node	47.29%
5	$\{A_{34}, A_{43}\}$	nodes at border	80.42%

We consider the scenario where the second request is sent from A_{22} to A_{44}. Since some nodes drop RREP messages (cf. Sect. 2.2), the probability of route establishment between A_{11} and A_{44} should decrease (compared to the 60% of Table 2). However, SMC-Uppaal shows that the probability of A_{11} finding a route to A_{44} is in the probability interval $[8.27913, 18.2791]$, i.e., a route discovery is unlikely. More results for the *basic* model are summarised in Table 3, grouped by the distance between the two originators. The table lists only the originator of the second route request; both the originator (A_{11}) of the first request and the destination (A_{44}) of both requests are fixed.

There is a correspondence between the success of route discovery and the distance between the two originators; moreover inner nodes have more influence on route discovery than nodes lying on the border of the network. This shows that the example of Fig. 1 occurs regularly. However, if the second originator $\texttt{oip}_2$ is far away from the first originator A_{11} no RREP message is dropped, since a route between $\texttt{oip}_2$ and A_{44} is established before the RREQ from A_{11} reaches $\texttt{oip}_2$. In the case of $\texttt{oip}_2 \in \{A_{24}, A_{42}, A_{34}, A_{43}\}$, the probability even increases. This is in line with Observation 3: when intermediate route reply is enabled, more RREP message are generated and the probability of route discovery success grows.

Observation 5. "Busy" mobile nodes increase the chance of route discovery.

One could rephrase this observation to "adding noise sometimes increases performance". At first glance it seems that adding additional network traffic—here a second route discovery processes—should not increase performance. But, let us look the scenario where the first data packet needs to be send from A_{11} to A_{44} (as before); the second packet is sent from A_{31} to the mobile node M. While handling the second RREQ most of the nodes will not learn about A_{44} and A_{11}. However it turns out that in the *basic* model, the probability of route discovery increases from around 60% to 72%. One reason is that the mobile node handles the request and generates a RREP message. While doing this it cannot handle the first RREQ; in case the first RREQ is sent to M and it is handling a different messages (is busy), the message is buffered. If the message is buffered for a while, the chance that the RREQ from A_{11} reaches A_{44} via a path without M as an

Table 4. different mobile node speed and impact on AODV variants

model	$\textbf{probability}_{\text{fast}}$	$\textbf{probability}_{\text{moderate}}$	$\textbf{probability}_{\text{slow}}$
basic	[48.5230,58.5230]	[55.4336,65.4336]	[61.9377,71.9377]
resend	[94.8645,100.00]	[95.00,100.00]	[94.3225,100.00]
dflag	[50.0136,60.0136]	[54.4851,64.4851]	[63.2927,73.2927]
dflag_res	[60.1762,70.1762]	[64.5122,74.5122]	[70.6098,80.6098]
dflag_res'	[75.8943,85.8943]	[81.3144,91.3144]	[85.5149,95.5149]
repair	[54.0786,64.0786]	[57.6016,67.6016]	[65.5962,75.5962]

intermediate hop, increases. Hence not the shortest, but the "fastest" route is established from A_{44} to A_{11}; this route is then used to send the RREP, since it does not use the mobile node as intermediate hop, the RREP is not lost.

4.3 Influence of Speed of Mobile Nodes

In our experiments the topology changes within a time frame of 35 to 45 milliseconds; This also determines the speed of the mobile node. One might argue that the speed of the mobile node affects our analysis and that other speeds could yield different behaviour. As shown in Table 4, this is not the case—the probabilities slightly change, but stay in the same ball park. Moreover the relationship between the different variants stays the same; a variant that is more reliable with a fast mobile node, is also more reliable with a slower node. For this category of experiments we enforce a topology change within the interval $[25, 35]$ (fast), $[35, 45]$ (moderate), and $[95, 105]$ (slow), respectively.

5 Related Work

Model checking has been used to analyse routing protocols in general and AODV in particular. For example, Bhargavan et al. [1] were amongst the first to use model checking—they used the SPIN model checker—on a draft of AODV, demonstrating the feasibility and value of automated verification of routing protocols. Musuvathi et al. [14] introduced the CMC model checker primarily to search for coding errors in implementations of protocols written in C. They used AODV as an example and, as well as discovering a number of errors, they also found a problem with the specification itself, which has since been corrected. Chiyangwa and Kwiatkowska [4] used the timing features of UPPAAL to study the relationship between the timing parameters and the performance of route discovery. None of these studies performed a quantitative analysis of AODV.

Statistical model checking techniques [19,18] are rather new. So far they have been used in a couple of case studies. Bulychey et al. [2] for example apply the SMC-Uppaal to an analysis of an instance of the Lightweight Media access

Control (LMAC) protocol; by this they are able to analyse ring topologies of up to 10 nodes.[8] Applications of SMC within biological systems are discussed in [5,13]. To the best of our knowledge, SMC was not used for the analysis of routing protocols so far—except in [11], where SMC-Uppaal is used to compare AODV and DYMO and to illustrate that even large topologies (up to 100 nodes) can be analysed by SMC. Our experiments are in line with this. However, it is unique in the sense that we carefully study variants of AODV.

6 Conclusion and Future Work

The aim of this paper has been a careful (quantitative) analysis of AODV and its variants using statistical model checking techniques. By this, we have made surprising observations on the behaviour of AODV. We have shown for example that some optional features (D-flag) should not be combined with others (resending). Another result shows that a well-known shortcoming occurs more often than expected and has a tremendous effect on the success of route establishment. One challenge we faced while performing our experiments has been the interpreting the data.

The results were often surprising and hard to interpret, particularly when they indicate odd behaviour. Unfortunately SMC-Uppaal does not store traces during analysis, thus it is difficult to recover counterexamples to explain the observations. At the moment counter examples are constructed "by hand" by formulating more probing queries beyond looking at overall performance. This suggests that more powerful statistical analysis such as "rare event simulation" in combination with multiple queries could be used to compile better evidence.

Next to this careful analysis, we also showed that SMC is a suitable tool for analysing WMNs. In this setting classical MC was limited to topologies with up to 6 nodes and therefore having a realistic mobility model was not possible.

Future work will be a continuation of our case study. In particular we want to look at topologies of up to 100 nodes—it has been shown that an analysis of such networks is possible [11]. However, choosing the right scenario is crucial here: Since one cannot analyse all scenarios, one has to pick the right topologies and the right mobility model(s); but in some sense finding the correct setting becomes a "stab in the dark". We hope that our previous experience helps to set the experiments right.

Acknowledgement. We thank Ansgar Fehnker, Rob van Glabbeek and Franck Cassez for fruitful discussions and their help.

NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

[8] Other case studies include firewire, bluetooth, and a train gate (see `http://people.cs.aau.dk/~adavid/smc/cases.html` for an overview).

References

1. Bhargavan, K., Obradovic, D., Gunter, C.A.: Formal verification of standards for distance vector routing protocols. J. ACM 49(4), 538–576 (2002)
2. Bulychev, P., David, A., Guldstrand Larsen, K., Legay, A., Mikučionis, M., Bøgsted Poulsen, D.: Checking and distributing statistical model checking. In: Goodloe, A.E., Person, S. (eds.) NFM 2012. LNCS, vol. 7226, pp. 449–463. Springer, Heidelberg (2012)
3. Bulychev, P., David, A., Larsen, K., Mikučionis, M., Bøgsted Poulson, D., Legay, A., Wang, Z.: UPPAAL-SMC: Statistical model checking for priced timed automata. In: Wiklicky, H., Massink, M. (eds.) Quantitative Aspects of Programming Languages. EPTCS, vol. 85, pp. 1–16. Open Publishing Association (2012)
4. Chiyangwa, S., Kwiatkowska, M.: A timing analysis of AODV. In: Steffen, M., Zavattaro, G. (eds.) FMOODS 2005. LNCS, vol. 3535, pp. 306–321. Springer, Heidelberg (2005)
5. Clarke, E.M., Faeder, J.R., Langmead, C.J., Harris, L.A., Jha, S.K., Legay, A.: Statistical Model Checking in BioLab: Applications to the automated analysis of T-cell receptor signaling pathway. In: Heiner, M., Uhrmacher, A.M. (eds.) CMSB 2008. LNCS (LNBI), vol. 5307, pp. 231–250. Springer, Heidelberg (2008)
6. Clausen, T., Colin de Verdiére, A., Yi, J., Niktash, A., Igarashi, Y., Satoh, H., Herberg, U., Lavenu, C., Lys, T., Perkins, C., Dean, J.: The lightweight on-demand ad hoc distance-vector routing protocol - next generation (LOADng). Internet Draft (Standards Track) (2013), `http://tools.ietf.org/html/draft-clausen-lln-loadng-08`
7. Fehnker, A., van Glabbeek, R., Höfner, P., McIver, A., Portmann, M., Tan, W.L.: Automated analysis of AODV using UPPAAL. In: Flanagan, C., König, B. (eds.) TACAS 2012. LNCS, vol. 7214, pp. 173–187. Springer, Heidelberg (2012)
8. Fehnker, A., van Glabbeek, R.J., Höfner, P., McIver, A., Portmann, M., Tan, W.L.: A process algebra for wireless mesh networks used for modelling, verifying and analysing AODV. Tech. Rep. 5513, NICTA (2013), `http://www.nicta.com.au/pub?id=5513`
9. Fehnker, A., Höfner, P., Kamali, M., Mehta, V.: Topology-based mobility models for wireless networks. In: Joshi, K., Siegle, M., Stoelinga, M., D'Argenio, P.R. (eds.) QEST 2013. LNCS, vol. 8054, pp. 389–404. Springer, Heidelberg (2013)
10. Höfner, P., van Glabbeek, R., Tan, W., Portmann, M., McIver, A., Fehnker, A.: A rigorous analysis of AODV and its variants. In: Modeling, Analysis and Simulation of Wireless and Mobile Systems, MSWIM 2012, pp. 203–212. ACM (2012)
11. Höfner, P., McIver, A.: Statistical model checking of wireless mesh routing protocols. In: Brat, G., Rungta, N., Venet, A. (eds.) NFM 2013. LNCS, vol. 7871, pp. 322–336. Springer, Heidelberg (2013)
12. IEEE P802.11s: IEEE draft standard for information technology—telecommunications and information exchange between systems—local and metropolitan area networks—specific requirements—part 11: Wireless LAN Medium Access Control (MAC) and physical layer (PHY) specifications-amendment 10: Mesh networking (July 2010)
13. Jha, S.K., Clarke, E.M., Langmead, C.J., Legay, A., Platzer, A., Zuliani, P.: A bayesian approach to model checking biological systems. In: Degano, P., Gorrieri, R. (eds.) CMSB 2009. LNCS, vol. 5688, pp. 218–234. Springer, Heidelberg (2009)
14. Musuvathi, M., Park, D.Y.W., Chou, A., Engler, D.R., Dill, D.L.: CMC: a pragmatic approach to model checking real code. In: Operating Systems Design and Implementation, OSDI 2002 (2002)

15. Perkins, C., Belding-Royer, E., Das, S.: Ad hoc on-demand distance vector (AODV) routing. RFC 3561 (Experimental) (2003), `http://www.ietf.org/rfc/rfc3561`
16. Perkins, C., Chakeres, I.: Dynamic MANET on-demand (AODVv2) routing. Internet Draft (Standards Track) (2013), `http://tools.ietf.org/html/draft-ietf-manet-dymo-25`
17. Perkins, C., Royer, E.: Ad-hoc On-Demand Distance Vector Routing. In: 2nd IEEE Workshop on Mobile Computing Systems and Applications, pp. 90–100 (1999)
18. Sen, K., Viswanathan, M., Agha, G.A.: Vesta: A statistical model-checker and analyzer for probabilistic systems. In: Quantitative Evaluaiton of Systems, QEST 2005, pp. 251–252. IEEE (2005)
19. Younes, H.: Verification and Planning for Stochastic Processes with Asynchronous Events. Ph.D. thesis, Carnegie Mellon University (2004)

More or Less True
DCTL for Continuous-Time MDPs

David N. Jansen

Radboud Universiteit, Model-Based System Development,
Nijmegen, The Netherlands
dnjansen@cs.ru.nl

Abstract. Discounted Computation Tree Logic is a logic that measures utility (as a real value in the interval [0,1]) instead of discrete truth (only 0 or 1). It is able to express properties that give more weight to the near future than to the far future. This article extends earlier work on DCTL with time, to continuous-time Markov chains and continuous-time Markov decision processes. It presents model checking algorithms for the two possible semantics of DCTL.

A long version of this article containing full proofs is available as [4].

1 Introduction

In what context is it appropriate to announce: "It is going to rain"? Unless the world goes away in the meantime, one could say that this statement is always true: it will eventually rain again, perhaps this afternoon, perhaps next week, or perhaps in three months. However, we are tempted to say that the statement is more appropriate in a situation where we expect rain very soon.

Similarly, all computer systems will fail eventually. Therefore, a requirement like "The system will forever fulfil its task" is, in some sense, never satisfied. Models to verify requirements on reactive systems (which engage in an ongoing interaction with their environment and are not meant to terminate) typically assume that the system has infinitely long behaviours, while every implementation will stop fulfilling its task at some time. In such situations, it is more appropriate to abandon speaking about absolute truth and rather look at the *utility* of a system. The longer a system fulfils its task, the higher its utility.

In another situation, namely testing, engineers are satisfied if the system fulfils its requirements for a certain amount of time. An error appearing after tenthousand successful tests is perhaps ascribed to mistyping some input, or it may just end up in the list of known bugs. An error appearing in the third test will cause a much stronger response: the system will have to be repaired.

In these situations, the point is that utility requirements give more weight to the near future than to the far future. The logic Discounted Computation Tree Logic (DCTL) was defined in [1] to express this kind of utility requirements. DCTL is interpreted in a quantitative setting, where any real number between 0 and 1 is a truth value (0 corresponding to "false" or "useless" and 1 to "true" or "most useful"), similar to a "degree of truth" in fuzzy logic [8]. To achieve the difference in weight between near and far future, it introduces *discounting:* an influence on the utility (a rainy state, an error in the system under test) is weighted by the length of the path to reach it.

Another possibility to regard DCTL properties is to think of an *impatient* game player. Assume somebody plays a (single-player) game that may involve both random choices and strategic or nondeterministic choices, to be resolved by the player. In some states, the player wins a reward. The player, however, is not ready to play the game infinitely long, but decides at any moment to abort the game with some probability. The utility expressed by a suitable DCTL property is a measure of the expected reward from the game.

DCTL was defined in [1] for discrete-time Markov decision processes (MDP), and their special cases discrete-time Markov chains (DTMC) and labelled transition systems (LTS). In these models, the only way to measure the length of a path is to count the number of transitions. While this works well when each step takes approximately the same amount of time, it becomes unexact when step timings differ.

Contributions. In this article, I therefore undertake to extend the work of [1] to continuous-time Markov decision processes (CTMDP) and their special case continuous-time Markov chains (CTMC). These formalisms equip each transition with a *rate* indicating how fast (on average) the transition is taken.

DCTL temporal operators, interpreted over DTMCs, were shown in [1] to have two differing semantics: a *fixpoint* and a *path* semantics. I show that these variants also exist in the continuous-time model and provide model checking algorithms for both. To resolve the nondeterminism in CTMDPs, one needs a scheduler or a strategy. The fixpoint semantics by definition only allows so-called positional schedulers, i. e. schedulers that always choose the same action in a given state. In the path semantics, in principle, several classes of schedulers are possible. However, as in the discrete-time case, I only provide a model checking algorithm for CMTCs, systems without nondeterminism.

2 Preliminaries

Notations. We denote the minimum of two values by $r \sqcap s := \min\{r, s\}$ and their maximum by $r \sqcup s := \max\{r, s\}$. For a $n \times n$-matrix A, let $A|_{i\ldots j}$ be the $n \times n$-matrix where all rows except the rows $i, i+1, \ldots, j$ have been set to 0.

Definition 1. *A continuous-time Markov decision process (CTMDP) is a tuple $\mathcal{M} = (S, A, R, AP, L)$ consisting of*

- *a nonempty set of states S;*
- *a nonempty, finite set of actions A;*
- *a transition rate matrix $R : S \times A \times S \to \mathbb{R}_{\geq 0}$, written $R^a(s, s')$.*
- *a nonempty, finite set of atomic propositions AP;*
- *a labelling function $L : S \times AP \to [0, 1]$ (here we extend the usual definition).*

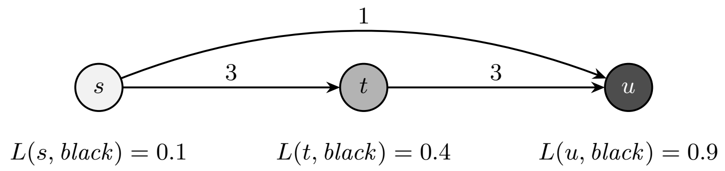


Fig. 1. Example CTMC

We denote the components of $\mathcal{M}$ by $S_{\mathcal{M}}$, $A_{\mathcal{M}}$, $R_{\mathcal{M}}$, $AP_{\mathcal{M}}$, $L_{\mathcal{M}}$, repectively, and we omit the subscript $_{\mathcal{M}}$ if $\mathcal{M}$ is clear from the context.

Informally speaking, the behaviour of a CTMDP is as follows: It is always in a state, say s. The atomic proposition p has utility $L(s, p) \in [0, 1]$. Transitions are triggered according to the following rule: The CTMDP first chooses nondeterministically an action a. Then, the probability that the transition to s' becomes enabled within at most t time units is $1 - e^{-R^a(s,s')t}$. If there are several states s' with $R^a(s, s') > 0$, the first transition that becomes enabled is taken. In effect, this leads to an exponential distribution with the so-called exit rate of $E^a(s) := \sum_{s' \in S} R^a(s, s')$. Its cumulative density function is $\Pr\left(s \xrightarrow[a]{\leq t} \mid a\right) = 1 - e^{-E^a(s)t}$ and its probability density function is $\mathrm{pdf}\left(s \xrightarrow[a]{t} \mid a\right) = E^a(s)e^{-E^a(s)t}$.

The probability to go to state s', if in state s action a is chosen, is then $P^a(s, s') := R^a(s, s')/E^a(s)$. Together, the transition to s' will be taken in time at most t with probability

$$\Pr\left(s \xrightarrow[a]{\leq t} s' \mid a\right) = P^a(s, s')(1 - e^{-E^a(s)t}) \quad .$$

We will also use $E_{\mathcal{M}}(s) := \max_{a \in A} E^a_{\mathcal{M}}(s)$ and $E_{\mathcal{M}} := \max_{s \in S} E_{\mathcal{M}}(s)$.

It may happen that for some states and actions, $E^a(s) = 0$. In that case, we say that s is *a-absorbing.* If the CTMDP is in state s and chooses action a, the behaviour stops, and the CTMDP stays in s forever. If state s is a-absorbing for each action $a \in A$, we call *s (strictly) absorbing.*

A *continuous-time Markov chain* (CTMC) is a CTMDP that does not contain nondeterministic choices, i. e., a CTMDP $\mathcal{M}$ with $|A_{\mathcal{M}}| = 1$.

Example 2. Figure 1 depicts an example of a simple continuous-time Markov chain. The proposition *black* denotes the “blackness” of a state. There is, a. o., a transition from s to t with rate $R^a(s, t) = 3$. State u is absorbing. The unique action has been omitted from the figure.

A *path* is a (finite or infinite) sequence of the form

$$s_0 \xrightarrow[a_0]{t_0} s_1 \xrightarrow[a_1]{t_1} s_2 \rightarrow \cdots$$

where $s_i \in S$, $a_i \in A$ such that $R^{a_i}(s_i, s_{i+1}) > 0$, and $t_i \in \mathbb{R}_{\geq 0}$. It represents a possible behaviour of the CTMDP. We denote the set of paths by *Path* and the set of maximal paths by $\mathit{Path}^{\max}$. (Maximal paths are paths that are infinite or that end in an absorbing state.)

A *cylinder set* is a set of maximal paths of the following form:

$$C\left(s_0, a_0, I_0, s_1, a_1, I_1, \dots, a_{n-1}, I_{n-1}, s_n\right) := \\ \left\{ s_0 \xrightarrow[a_0]{t_0} s_1 \xrightarrow[a_1]{t_1} \cdots \xrightarrow[a_{n-1}]{t_{n-1}} s_n \to \cdots \;\middle|\; t_i \in I_i \text{ for all } i < n \right\}$$

where I_i are intervals in $\mathbb{R}_{\geq 0}$.

The Probability Space of Paths. A *scheduler* is a function that resolves the nondeterministic choices in a CTMDP. A CTMDP together with a measurable scheduler induces a fully probabilistic process, where a probability space can be defined.

A scheduler in principle bases its choices on the history of the system. Restricted scheduler classes only check a part of the history. The most powerful schedulers, history-dependent schedulers, are functions from non-maximal paths to actions $D : (\mathit{Path} \setminus \mathit{Path}^{\max}) \to A$. See [7] for details. However, in the rest of this article, we will mostly look at (time-abstract) *positional* schedulers, which base the choice on the current state only. They are functions $D : S \to A$.

A CTMDP (S, A, R, AP, L) and a positional scheduler D together induce a CTMC $(S, \{*\}, \tilde{R}, AP, L)$. Its transition relation is defined by $\tilde{R}^*(s, s') = R^{D(s)}(s, s')$. From this, one can define a probability space $(\mathit{Path}^{\max}, \mathcal{C}, \Pr_s^D)$. Here, $\mathcal{C}$ is the smallest σ-algebra that contains all cylinder sets. Its probability measure is the unique measure induced by:

$$\Pr_{s_0}^D(C(s_0)) := 1$$
$$\Pr_{s_0}^D\left(C\left(s_0, D(s_0), I_0, s_1, a_1, I_1, \dots, a_n, I_n, s_{n+1}\right)\right) := \\ := \Pr\left(s_0 \xrightarrow[D(s_0)]{\in I_0} s_1 \mid D(s_0)\right) \cdot \Pr_{s_1}^D\left(C\left(s_1, a_1, I_1, \dots, a_n, I_n, s_{n+1}\right)\right).$$

Lemma 3. *The function defined above is a measure.*

Proof. This lemma has been shown in [7, Theorem 2]. □

A CTMDP $\mathcal{M}$ is *uniform* if all exit rates are equal, i.e. for all states s and actions a, $E^a_{\mathcal{M}}(s) = E_{\mathcal{M}}$. For a CTMDP $\mathcal{M}$, we can find a uniform CTMDP having a similar probability distribution as follows.

We first define the CTMDP $\mathcal{M}_{\text{unif}} = (S, A, \tilde{R}, AP, L)$ which is almost the same as $\mathcal{M}$, except that we add to each state a self-loop such that the total exit rate becomes $E \geq E_{\mathcal{M}}$ for every state and action:

$$\tilde{R}^a(s, s') := \begin{cases} R^a(s, s') & \text{if } s \neq s' \\ E - \sum_{t \neq s} R^a(s, t) & \text{if } s = s' \end{cases} .$$

For a uniform (or uniformised) CTMDP, we can define its *embedded* discrete-time MDP as follows: $\mathcal{P} = \mathcal{P}_{\mathcal{M}}$ is given by (S, A, P, AP, L), where the transition probabilities $P^a(s, s')$ are given by $P^a(s, s') = R^a(s, s')/E_{\mathcal{M}}$.

Lemma 4. *Positional schedulers are invariant under uniformisation, i. e. for every scheduler on $\mathcal{M}$ there is one on $\mathcal{M}_{\text{unif}}$ and vice versa, such that the probability measures of the respective induced CTMCs are equal up to stuttering in the same state.*

Proof. As the $\mathcal{M}_{\text{unif}}$-scheduler corresponding to D on $\mathcal{M}$, we choose D again. This is possible as the state space and the set of actions do not change. The (short) remainder of the proof is in the technical report [4]. $\square$

3 The Logic DCTL

DCTL consists of state formulas, denoted φ, ψ, and path formulas, denoted Ψ. The syntax of state and path formulas is given by the following grammar:

$$\varphi \ ::= \ 1 \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \oplus_w \varphi \mid \exists\Psi \mid \forall\Psi$$

$$\Psi \ ::= \ \Diamond_\alpha \varphi \mid \Box_\alpha \varphi \mid \triangle_\alpha \varphi$$

where p is an atomic proposition, $w \in [0,1]$ is a weight, and $\alpha \in \mathbb{R}_{>0}$ is a discount rate. We call $\Diamond_\alpha \varphi$ the *discounted maximum,* $\Box_\alpha \varphi$ the *discounted minimum,* and $\triangle_\alpha \varphi$ the *discounted average* over paths.

3.1 Semantics

The semantics of a DCTL state formula φ is a mapping $[\![\varphi]\!] : S \to [0,1]$ from states to utilities. It can be defined by:

1	true	$[\![1]\!](s) = 1$
p	atomic proposition, $p \in AP$	$[\![p]\!](s) = L(s,p)$
$\neg\varphi$	negation	$[\![\neg\varphi]\!](s) = 1 - [\![\varphi]\!](s)$
$\varphi \wedge \psi$	conjunction	$[\![\varphi \wedge \psi]\!](s) = [\![\varphi]\!](s) \sqcap [\![\psi]\!](s)$
$\varphi \oplus_w \psi$	weighted sum, $w \in [0,1]$	$[\![\varphi \oplus_w \psi]\!](s) = (1-w)\,[\![\varphi]\!](s) + w\,[\![\psi]\!](s)$

The interpretation of $\exists\Psi$ and $\forall\Psi$ is closely linked to the interpretation of path formulas. Informally, the semantics of $\exists\Psi$ is the expected utility of Ψ under the *best* scheduler and $\forall\Psi$ is its expected utility under the *worst* scheduler. $\Diamond_\alpha \varphi$ is the maximum of all φ-values along the path, where we apply a discount: if a value appears after time t, it is reduced with the factor $e^{-\alpha t}$, with the result that earlier states can more easily influence the maximum than later ones. $\Box_\alpha \varphi$ is the minimum of all discounted φ-values along the path, and $\triangle_\alpha \varphi$ is the average over all φ-values, where the value after time t now gets relative weight $e^{-\alpha t}$, in accordance with the idea that earlier states are more important than later ones.

There are two possibilities to formalise the semantics of path formulas: either as the *fixpoint* of an operator, or via a look at the *complete path* at once. In the following sections, we will give the detailed definition of the two semantics.

$$\llbracket \exists \Diamond_\alpha \varphi \rrbracket^{\mathrm{f}}(s) = \mu u.\, \llbracket \varphi \rrbracket^{\mathrm{f}}(s) \sqcup \max_{a \in A} \frac{1}{E^a(s) + \alpha} \sum_{s' \in S} R^a(s, s') u(s')$$

$$\llbracket \forall \Diamond_\alpha \varphi \rrbracket^{\mathrm{f}}(s) = \mu u.\, \llbracket \varphi \rrbracket^{\mathrm{f}}(s) \sqcup \min_{a \in A} \frac{1}{E^a(s) + \alpha} \sum_{s' \in S} R^a(s, s') u(s')$$

$$\llbracket \exists \Box_\alpha \varphi \rrbracket^{\mathrm{f}}(s) = \mu u.\, \llbracket \varphi \rrbracket^{\mathrm{f}}(s) \sqcap \max_{a \in A} \frac{\alpha}{E^a(s) + \alpha} + \frac{1}{E^a(s) + \alpha} \sum_{s' \in S} R^a(s, s') u(s')$$

$$\llbracket \forall \Box_\alpha \varphi \rrbracket^{\mathrm{f}}(s) = \mu u.\, \llbracket \varphi \rrbracket^{\mathrm{f}}(s) \sqcap \min_{a \in A} \frac{\alpha}{E^a(s) + \alpha} + \frac{1}{E^a(s) + \alpha} \sum_{s' \in S} R^a(s, s') u(s')$$

$$\llbracket \exists \triangle_\alpha \varphi \rrbracket^{\mathrm{f}}(s) = \mu u.\, \max_{a \in A} \frac{\alpha}{E^a(s) + \alpha} \llbracket \varphi \rrbracket^{\mathrm{f}}(s) + \frac{1}{E^a(s) + \alpha} \sum_{s' \in S} R^a(s, s') u(s')$$

$$\llbracket \forall \triangle_\alpha \varphi \rrbracket^{\mathrm{f}}(s) = \mu u.\, \min_{a \in A} \frac{\alpha}{E^a(s) + \alpha} \llbracket \varphi \rrbracket^{\mathrm{f}}(s) + \frac{1}{E^a(s) + \alpha} \sum_{s' \in S} R^a(s, s') u(s')$$

Fig. 2. Fixpoint semantics for CTMDPs

3.2 Fixpoint Semantics

The fixpoint semantics arises by lifting the classical connection between CTL and the μ-calculus to a quantitative setting. For a transition system, we denote by $\exists \mathit{Pre}(u)$ the set of all states that have a transition into the set u. Then the set $\llbracket \exists \Diamond \varphi \rrbracket$ of all states satisfying $\exists \Diamond \varphi$ is the smallest fixpoint of the equation $u = \llbracket \varphi \rrbracket \cup \exists \mathit{Pre}(u)$, written as $\mu u.\, \llbracket \varphi \rrbracket \cup \exists \mathit{Pre}(u)$.

As explained in [1], one can lift these fixpoint equations to a quantitative setting by interpreting $\cup$ as the pointwise maximum and $\exists \mathit{Pre}(u)$ as the expected value of u, achievable in one step, under the best scheduler. Then, the semantics of $\exists \Diamond_\alpha \varphi$ is the greater of the current utility of φ and the expected utility after one transition. When we formalise this idea, we get the equations in Fig. 2 for the fixpoint semantics. Some remarks on them are in order.

Discounted Maximum Operator. For a CTMC $\mathcal{M}$, the discounted semantics of $\exists \Diamond_\alpha \varphi$ is obtained as follows. Let α be a discount rate and $u : S \to [0, 1]$ be a state utility function. Then $\exists \mathit{Pre}_\alpha^{\mathcal{M}}(u) : S \to [0, 1]$ yields at state s the expected discounted utility immediately after the first jump. That is, if we move to s' at time t, we get $e^{-\alpha t} u(s')$. Thus, $\exists \mathit{Pre}_\alpha^{\mathcal{M}}(u)(s)$ is the expected value of the random variable $e^{-\alpha T} u(X)$, where T is the random variable denoting the time of the first jump and X the random variable denoting the state reached by the first jump.

For CTMDPs, we additionally maximize over all actions $a \in A$, i.e. we choose the action with the maximal expected discounted utility. Let X_0 be the current state, and A_0 the action taken in this state. A simple calculation shows that

$$\begin{aligned} \exists \mathit{Pre}_\alpha^{\mathcal{M}}(u)(s) &= \max_{a \in A} \mathbb{E}[e^{-\alpha T} u(X) | X_0 = s, A_0 = a] \\ &= \max_{a \in A} \frac{1}{E^a(s) + \alpha} \sum_{s' \in S} R^a(s, s') u(s'). \end{aligned}$$

Inserting these results in $[\![\exists \Diamond_\alpha \varphi]\!]^{\mathrm{f}} = \mu u.\, [\![\varphi]\!]^{\mathrm{f}} \sqcup \exists Pre_\alpha^{\mathcal{M}}(u)$ and $[\![\forall \Diamond_\alpha \varphi]\!]^{\mathrm{f}} = \mu u.\, [\![\varphi]\!]^{\mathrm{f}} \sqcup \forall Pre_\alpha^{\mathcal{M}}(u)$ yields the equations in Fig. 2.

Discounted Minimum Operator. The operator $\forall\Box_\alpha\varphi$ should be equivalent to $\neg\exists \Diamond_\alpha \neg\varphi$. Therefore, we derive the semantics as follows:

$$\begin{aligned}[\![\forall\Box_\alpha\varphi]\!]^{\mathrm{f}}(s) &= 1 - \mu u.(1 - [\![\varphi]\!]^{\mathrm{f}}(s)) \sqcup \max_{a\in A} \mathbb{E}[e^{-\alpha T}u(X)|X_0 = s, A_0 = a] \\ &= \nu\hat{u}.\, [\![\varphi]\!]^{\mathrm{f}}(s) \sqcap \underbrace{\min_{a\in A} \mathbb{E}[1 - e^{-\alpha T}(1 - \hat{u}(X))|X_0 = s, A_0 = a]}_{=:\forall\widehat{Pre}_\alpha^{\mathcal{M}}(\hat{u})}.\end{aligned}$$

Then,

$$\forall\widehat{Pre}_\alpha^{\mathcal{M}}(\hat{u})(s) = \min_{a\in A} \frac{\alpha}{E^a(s)+\alpha} + \frac{1}{E^a(s)+\alpha}\sum_{s'\in S} R^a(s,s')\hat{u}(s').$$

We will see shortly that the fixpoint is unique. Therefore, we can write μ in Fig. 2 instead of ν.

Discounted Average Operator. To calculate $[\![\exists\triangle_\alpha\varphi]\!]^{\mathrm{f}}$, we have to take an average between $[\![\varphi]\!]^{\mathrm{f}}$ and the utility after one step, giving more weight to the near future than to the far one. The natural candidate to choose the weight is the pdf of the probability to lose patience, $t \mapsto \alpha e^{-\alpha t}$. This leads to the definition:

$$\begin{aligned}[\![\exists\triangle_\alpha\varphi]\!]^{\mathrm{f}}_{\mathcal{M}}(s) = \mu u.\max_{a\in A}\sum_{s'\in S}\int_0^\infty \mathrm{pdf}\Big(s \xrightarrow[a]{t} s' \mid a\Big) \cdot \\ \cdot \left[\int_0^t \alpha e^{-\alpha\tau}\, d\tau\, [\![\varphi]\!]^{\mathrm{f}}_{\mathcal{M}}(s) + \int_t^\infty \alpha e^{-\alpha\tau}\, d\tau u(s')\right] dt\end{aligned}$$

Properties. The fixpoints are unique because the respective functions are contractions:

Lemma 5. *The functions under the fixpoint operators are contractions according to the L^∞-norm.*

Proof. See the technical report version [4]. □

Proposition 6 (Duality Laws). *For all DCTL formulas φ and all CTMDPs $\mathcal{M}$, we have*

$$[\![\neg\exists \Diamond_\alpha \varphi]\!]^{\mathrm{f}}_{\mathcal{M}} = [\![\forall\Box_\alpha\neg\varphi]\!]^{\mathrm{f}}_{\mathcal{M}} \qquad [\![\neg\exists\Box_\alpha\varphi]\!]^{\mathrm{f}}_{\mathcal{M}} = [\![\forall \Diamond_\alpha \neg\varphi]\!]^{\mathrm{f}}_{\mathcal{M}}$$

$$[\![\neg\exists\triangle_\alpha\varphi]\!]^{\mathrm{f}}_{\mathcal{M}} = [\![\forall\triangle_\alpha\neg\varphi]\!]^{\mathrm{f}}_{\mathcal{M}}$$

Theorem 7. *Let $\mathcal{M}$ be a CTMDP and $\mathcal{M}_{\mathrm{unif}}$ be its uniformization. Then we have for all DCTL formulas φ that $[\![\varphi]\!]^{\mathrm{f}}_{\mathcal{M}} = [\![\varphi]\!]^{\mathrm{f}}_{\mathcal{M}_{\mathrm{unif}}}$.*

Proof. The probability to be in a specific state at any given time, under a given scheduler, is the same for $\mathcal{M}$ and $\mathcal{M}_{\mathrm{unif}}$ (Lemma 4). Only the number of times a state is reentered changes, and so also the number of times that an action can be chosen. However, a positional scheduler has to choose the same action whenever the same state is reentered, so the scheduler cannot make a different choice in $\mathcal{M}_{\mathrm{unif}}$ than in $\mathcal{M}$. □

We will see later that so-called late schedulers, that choose the action to take not when entering, but when leaving a state, can deliver better results (higher values for $\exists \Diamond_\alpha$, lower values for $\forall \Diamond_\alpha$) than positional schedulers. Late schedulers may decide the action not only on the current state, but also on the sojourn time in this state. A scheduler that can distinguish between entering a state for the first time and reentering a state can get an *estimate* of the sojourn time and improve over a positional scheduler based on this estimate. This was a problem for model checking of non-uniform CTMDPs in [3].

3.3 Path Semantics

Another way to define a CTL semantics is to look at the set of (maximal) paths. A path satisfies $\Diamond \varphi$ if it contains at least one φ-state; a state s satisfies $\exists \Diamond \varphi$ if the set of paths starting in s contains some path satisfying $\Diamond \varphi$. If we lift this principle to a quantitative setting, we reinterpret "at least one φ-state" as "the maximum of $[\![\varphi]\!]$", and we reinterpret "existence of a path" as "supremum over all paths". This leads to $[\![\exists \Diamond \varphi]\!] = \sup_{\sigma \in \mathit{Path}} \max_{s \in \sigma} [\![\varphi]\!](s)$. In a Markov decision process, one would additionally consider the probabilities of paths.

In the path semantics of DCTL, we give separate semantics to the operators $\exists$ and $\Diamond_\alpha$. Let us first look at $\exists$ and $\forall$: Assume given a class of schedulers $\mathfrak{D}$. Then, we define:

$$[\![\exists \Psi]\!]^{\mathrm{p}}_{\mathcal{M}}(s) = \sup_{D \in \mathfrak{D}} \mathbb{E}^{D}_{\sigma \in C(s)} [\![\Psi]\!]^{\mathrm{p}}_{\mathcal{M}}(\sigma)$$
$$[\![\forall \Psi]\!]^{\mathrm{p}}_{\mathcal{M}}(s) = \inf_{D \in \mathfrak{D}} \mathbb{E}^{D}_{\sigma \in C(s)} [\![\Psi]\!]^{\mathrm{p}}_{\mathcal{M}}(\sigma)$$

The expected value $\mathbb{E}$ in these formulas is of course taken over the probability space of paths, under the mentioned scheduler.

The semantics of path formulas is:

$$[\![\Diamond_\alpha \varphi]\!]^{\mathrm{p}}_{\mathcal{M}}(\sigma) = \max_{t \geq 0} e^{-\alpha t} [\![\varphi]\!]^{\mathrm{p}}_{\mathcal{M}}(\sigma@t) \tag{1}$$

$$[\![\Box_\alpha \varphi]\!]^{\mathrm{p}}_{\mathcal{M}}(\sigma) = \min_{t \geq 0} 1 - e^{-\alpha t}(1 - [\![\varphi]\!]^{\mathrm{p}}_{\mathcal{M}}(\sigma@t)) \tag{2}$$

$$[\![\triangle_\alpha \varphi]\!]^{\mathrm{p}}_{\mathcal{M}}(\sigma) = \int_0^\infty \alpha e^{-\alpha t} [\![\varphi]\!]^{\mathrm{p}}_{\mathcal{M}}(\sigma@t)\, dt$$

Existence of Maximum. In an undiscounted setting and with a finite LTS, $[\![\varphi]\!]$ can only take a finite number of values on any path, so it is clear that $[\![\Diamond\,\varphi]\!]$ is not only a supremum, but really a maximum. However, with continuous discounting, we have to prove existence of the maximum.

Lemma 8. *The discounted maximum and minimum are well-defined, i. e., the maximum* (1) *and minimum* (2) *actually do exist.*

Proof. Basically, one still can find a finite subset of all $e^{-\alpha t}\,[\![\varphi]\!]^{\mathrm{p}}_{\mathcal{M}}\,(\sigma@t)$ in (1) that contains the supremum, and therefore the maximum. See [4] for details. □

Example 9. Let us calculate $[\![\exists\,\Diamond_2\,\mathit{black}]\!]^{\mathrm{p}}\,(s)$ in the Markov chain of Fig. 1. As it does not contain nondeterminism, there is only the trivial scheduler. There are two time-abstract maximal paths in this Markov chain: $s \to u$ and $s \to t \to u$. We calculate $\mathbb{E}_{\sigma\in C(s)}\,[\![\Diamond_2\,\mathit{black}]\!]^{\mathrm{p}}\,(\sigma)$ by summing over these two paths separately. For the path $s \to u$,

$$\begin{aligned}\mathbb{E}_{\sigma=(s\to u)}\,[\![\Diamond_2\,\mathit{black}]\!]^{\mathrm{p}}\,(\sigma) &= \int_0^\infty [\![\Diamond_2\,\mathit{black}]\!]^{\mathrm{p}}\left(s \xrightarrow{t} u\right)\mathrm{pdf}\left(s \xrightarrow{t} u\right)dt\\ &= \int_0^\infty \max_{\tau\geq 0} e^{-2\tau}\,[\![\mathit{black}]\!]^{\mathrm{p}}\left(\left[s \xrightarrow{t} u\right]@\tau\right)R(s,u)e^{-E(s)t}\,dt\\ &= \int_0^\infty \max\left\{e^{-2t}\tfrac{9}{10},\tfrac{1}{10}\right\}e^{-4t}\,dt = \tfrac{1459}{9720} \approx 0.15\end{aligned}$$

Similarly, for the path $s \to t \to u$, we have that

$$\mathbb{E}_{\sigma=(s\to t\to u)}\,[\![\Diamond_2\,\mathit{black}]\!]^{\mathrm{p}}\,(\sigma) = \tfrac{1591}{5400} \approx 0.29$$

So, in the end,

$$[\![\exists\,\Diamond_2\,\mathit{black}]\!]^{\mathrm{p}}\,(s) = \tfrac{1459}{9720} + \tfrac{1591}{5400} = \tfrac{10807}{24300} \approx 0.44.$$

Lemma 10.

$$[\![\Diamond_\alpha\,\varphi]\!]^{\mathrm{p}}_{\mathcal{M}}\,(\sigma) = \max_{t\geq 0}\,[\![0 \oplus_{e^{-\alpha t}}\,\varphi]\!]^{\mathrm{p}}_{\mathcal{M}}\,(\sigma@t)$$

$$[\![\Box_\alpha\varphi]\!]^{\mathrm{p}}_{\mathcal{M}}\,(\sigma) = \min_{t\geq 0}\,[\![1 \oplus_{e^{-\alpha t}}\,\varphi]\!]^{\mathrm{p}}_{\mathcal{M}}\,(\sigma@t)$$

Proposition 11. *The following dualities between $\Diamond_\alpha$ and $\Box_\alpha$ hold. $\triangle_\alpha$ is its own dual.*

$$[\![\exists\,\Diamond_\alpha\,\neg\varphi]\!]^{\mathrm{p}} = [\![\neg\forall\Box_\alpha\varphi]\!]^{\mathrm{p}} \qquad [\![\forall\,\Diamond_\alpha\,\neg\varphi]\!]^{\mathrm{p}} = [\![\neg\exists\Box_\alpha\varphi]\!]^{\mathrm{p}}$$

$$[\![\exists\triangle_\alpha\neg\varphi]\!]^{\mathrm{p}} = [\![\neg\forall\triangle_\alpha\varphi]\!]^{\mathrm{p}}$$

Proposition 12. *The fixpoint and path semantics of $\triangle_\alpha$ coincide.*

Proof. The proof of this property exactly follows the lines of [1, Thm. 2]. □

4 Model Checking DCTL

The model checking problem for DCTL consists in computing the value of $[\![\varphi]\!]^{\mathrm{f}}_{\mathcal{M}}(s)$, given a formula φ and a state s in a CTMDP $\mathcal{M}$. Based on the equations found in the previous section, we propose the following algorithms to solve this problem.

Finding the utility of the formulas 1 and p (for $p \in AP$) is trivial. Checking the formulas $\neg\varphi$, $\varphi \wedge \psi$, and $\varphi \oplus_w \psi$ is easy and straightforward, if we have already found the truth values of the subformulas. We will not treat them in the rest of the article, but concentrate on the operators $\exists\Diamond_\alpha$, $\exists\Box_\alpha$ and $\exists\triangle_\alpha$. The duality laws (Props. 6 and 11) can be used to check the operators $\forall\Diamond_\alpha$, $\forall\Box_\alpha$ and $\forall\triangle_\alpha$.

4.1 Model Checking the Fixpoint Semantics

Reduction to DCTL Semantics for MDPs. The fixpoint equations in Fig. 2, expressing the semantics for DCTL over CTMDPs, are very similar to the fixpoint equations for discrete-time MDPs. For example, in the discrete-time case, we have

$$[\![\exists\Box_\rho\varphi]\!]^{\mathrm{f}}_{\mathcal{P}}(s) = \mu u.\, [\![\varphi]\!]^{\mathrm{f}}_{\mathcal{P}}(s) \sqcap \left[(1-\rho) + \rho \max_{a\in A} \sum_{s'\in S} P^a(s,s')u(s')\right]$$

In fact, they are so analogous that the DCTL model checking problem for CT-MDPs reduces to the DCTL model checking problem for MDPs: We can model check a DCTL formula φ over a uniform CTMDP $\mathcal{M}$ by model checking a formula φ' over the embedded Markov decision process. Here φ' arises from φ by changing the discount factors $\Diamond_\alpha$ etc. to $\Diamond_\rho$ in φ, where $\rho = E_{\mathcal{M}}/(E_{\mathcal{M}}+\alpha)$. Note that, as the fixpoint semantics only uses positional schedulers, it is possible to analyse other CTMDPs after uniformising them.

Theorem 13. *Let $\mathcal{M}$ be a CTMDP, let $\mathcal{P}$ be its embedded discrete-time MDP (possibly after uniformisation) and let $\rho = E_{\mathcal{M}}/(E_{\mathcal{M}}+\alpha)$. Then we have for all DCTL formulas φ that*

$$[\![\exists\Diamond_\alpha\varphi]\!]^{\mathrm{f}}_{\mathcal{M}} = [\![\exists\Diamond_\rho\varphi']\!]^{\mathrm{f}}_{\mathcal{P}} \quad [\![\exists\Box_\alpha\varphi]\!]^{\mathrm{f}}_{\mathcal{M}} = [\![\exists\Box_\rho\varphi']\!]^{\mathrm{f}}_{\mathcal{P}} \quad [\![\exists\triangle_\alpha\varphi]\!]^{\mathrm{f}}_{\mathcal{M}} = [\![\exists\triangle_\rho\varphi']\!]^{\mathrm{f}}_{\mathcal{P}}$$

$$[\![\forall\Diamond_\alpha\varphi]\!]^{\mathrm{f}}_{\mathcal{M}} = [\![\forall\Diamond_\rho\varphi']\!]^{\mathrm{f}}_{\mathcal{P}} \quad [\![\forall\Box_\alpha\varphi]\!]^{\mathrm{f}}_{\mathcal{M}} = [\![\forall\Box_\rho\varphi']\!]^{\mathrm{f}}_{\mathcal{P}} \quad [\![\forall\triangle_\alpha\varphi]\!]^{\mathrm{f}}_{\mathcal{M}} = [\![\forall\triangle_\rho\varphi']\!]^{\mathrm{f}}_{\mathcal{P}}$$

Since the reduction above is clearly linear in the size of $\mathcal{M}$, the complexity of model checking the DCTL fixpoint semantics is the same for CTMDPs and discrete-time MDPs. Thus, we obtain the following corollary from [1, Thm. 7].

Corollary 14. *The problem of model checking $[\![\varphi]\!]^{\mathrm{f}}_{\mathcal{M}}$ can be solved in nondeterministic polynomial time in the size of $\mathcal{M}$ and exponential in $|\varphi|$.*

4.2 Model Checking the Path Semantics

The article [1] proposes a model checking procedure for discrete-time Markov chains of $\exists \Diamond_\alpha \varphi$ that can be adapted to continuous-time Markov chains. (Note that we assume there is no nondeterminism here.) For any path σ, the value $[\![\Diamond_\alpha \varphi]\!]^{\mathrm{P}}(\sigma)$ is reached at a time when σ enters a state that is better than $\sigma@0$ (i. e., $e^{-\alpha t}[\![\varphi]\!]^{\mathrm{P}}(\sigma@t) > [\![\varphi]\!]^{\mathrm{P}}(\sigma@0)$ for some t). We therefore order the states according to their value $[\![\varphi]\!]^{\mathrm{P}}$:

$$[\![\varphi]\!]^{\mathrm{P}}(s_1) \geq [\![\varphi]\!]^{\mathrm{P}}(s_2) \geq \cdots \geq [\![\varphi]\!]^{\mathrm{P}}(s_n)$$

In state s_1, no other state can improve $[\![\varphi]\!]^{\mathrm{P}}$, so we know that $[\![\exists \Diamond_\alpha \varphi]\!]^{\mathrm{P}}(s_1) = [\![\varphi]\!]^{\mathrm{P}}(s_1)$.

In state s_2, the only possible improvement is to reach s_1 at some time t with $[\![\varphi]\!]^{\mathrm{P}}(s_2) < [\![\varphi]\!]^{\mathrm{P}}(s_1)e^{-\alpha t} = [\![\exists \Diamond_\alpha \varphi]\!]^{\mathrm{P}}(s_1)e^{-\alpha t}$. We calculate the probability that s_1 is reached early enough and the resulting discounted value of $[\![\exists \Diamond_\alpha \varphi]\!]^{\mathrm{P}}(s_2)$ via a variant of time-bounded reachability.

In general, if a path starts in s_i, improvements occur if some state s_j is reached at a time t such that $[\![\varphi]\!]^{\mathrm{P}}(s_i) < [\![\exists \Diamond_\alpha \varphi]\!]^{\mathrm{P}}(s_j)e^{-\alpha t}$. We can base our algorithm on the observation that *after the first such state s_j, no more improvement is possible:* any further improvement is already taken into account by $[\![\exists \Diamond_\alpha \varphi]\!]^{\mathrm{P}}(s_j)$. The second idea used in the algorithm is that of *cutoff time:* After some time, the discount factor $e^{-\alpha t}$ is so small that reaching s_{i-1} will no longer improve over s_i. From that moment on, we could as well presume that any state s_j (for $j = i, \ldots, n$) had $[\![\varphi]\!]^{\mathrm{P}}(s_j) = [\![\varphi]\!]^{\mathrm{P}}(s_{i-1})$. Therefore, it is enough to regard the current path until the cutoff time, after which we can continue analysis in a simplified Markov chain.

This allows to use a sequence of CTMCs $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_n$ where $\mathcal{M}_i$ has the same structure as $\mathcal{M}$ but change $[\![\varphi]\!]^{\mathrm{P}}$ to:

$$[\![\varphi]\!]^{\mathrm{P}}_{\mathcal{M}_i}(s_j) := [\![\varphi]\!]^{\mathrm{P}}_{\mathcal{M}}(s_{\min\{i,j\}}).$$

Note that one can easily prove that $[\![\exists \Diamond_\alpha \varphi]\!]^{\mathrm{P}}_{\mathcal{M}_i}(s_j) = [\![\exists \Diamond_\alpha \varphi]\!]^{\mathrm{P}}_{\mathcal{M}}(s_j)$ if $j \leq i$. We now calculate $[\![\exists \Diamond_\alpha \varphi]\!]^{\mathrm{P}}_{\mathcal{M}_i}$ recursively:

Base case. Trivial: $[\![\exists \Diamond_\alpha \varphi]\!]^{\mathrm{P}}_{\mathcal{M}_1}(s_i) = [\![\varphi]\!]^{\mathrm{P}}_{\mathcal{M}}(s_1)$ for all $1 \leq i \leq n$.

Iteration step. Assume that $[\![\exists \Diamond_\alpha \varphi]\!]^{\mathrm{P}}_{\mathcal{M}_i}(s_j)$ has been calculated. We know that it is equal to $[\![\exists \Diamond_\alpha \varphi]\!]^{\mathrm{P}}_{\mathcal{M}}(s_j)$ if $j \leq i$.

One can calculate of $[\![\exists \Diamond_\alpha \varphi]\!]^{\mathrm{P}}_{\mathcal{M}_{i+1}}$ by solving a reachability problem in a (slightly modified) CTMC $\mathcal{M}'_{i+1}$. First, select some path σ through $\mathcal{M}'_{i+1}$ and at a suitable moment t_{iter} take the utility $[\![\exists \Diamond_\alpha \varphi]\!]^{\mathrm{P}}_{\mathcal{M}_i}(\sigma@t_{\mathrm{iter}})$. If σ starts in $s_1, \ldots, s_i$, we can pick $t_{\mathrm{iter}} = 0$. To simplify the choice, these states are absorbing in $\mathcal{M}'_{i+1}$, so that any value for t_{iter} is equivalent to picking $t_{\mathrm{iter}} = 0$.

Otherwise, we follow the transitions of σ until it reaches some state in $s_1, \ldots, s_i$; then, t_{iter} is the time of reaching such a state for the first time. To take into account the discount factor, we add an (absorbing) state $s_\perp$ with $[\![\varphi]\!]^{\mathrm{p}}_{\mathcal{M}'_{i+1}}(s_\perp) := 0$, and from each state in $s_{i+1}, \ldots, s_n$, we add a transition with rate α to $s_\perp$. However, we never wait longer than $t_{\mathrm{cut}} := [\ln [\![\varphi]\!]^{\mathrm{p}}_{\mathcal{M}}(s_i) - \ln [\![\varphi]\!]^{\mathrm{p}}_{\mathcal{M}}(s_{i+1})]/\alpha$: If no such state is reached before t_{cut}, we pick $t_{\mathrm{iter}} = t_{\mathrm{cut}}$.

Overall, $\mathcal{M}'_{i+1}$ has state space $S_{\mathcal{M}} \cup \{s_\perp\}$, rate matrix

$$R_{\mathcal{M}'_{i+1}} = \left(\begin{array}{ccc|c} & & & 0 \\ & \mathbf{0} & & \vdots \\ & & & 0 \\ \hline & & & \alpha \\ & (R_{\mathcal{M}})_{i+1\ldots n} & & \vdots \\ & & & \alpha \\ \hline 0 & \ldots & 0 & 0 \end{array}\right) \begin{array}{l} \left.\vphantom{\begin{array}{c}0\\ \vdots\\ 0\end{array}}\right\} i \text{ zero rows} \\ \left.\vphantom{\begin{array}{c}\alpha\\ \vdots\\ \alpha\end{array}}\right\} n-i \text{ rows from } R_{\mathcal{M}} = R_{\mathcal{M}_{i+1}} \\ \} \text{ zero row for } s_\perp \end{array} \qquad (3)$$

and the other parts of $\mathcal{M}'_{i+1}$ coincide with $\mathcal{M}_{i+1}$. This rate matrix has the additional property that we can wait until t_{cut} in all cases: one only chooses an earlier moment for t_{iter} if one has reached $s_1, \ldots, s_i$, and these states are absorbing. This leads to the following lemma:

Lemma 15. *With the sequence of $\mathcal{M}_i$ introduced above, and $Q = R_{\mathcal{M}} - E_{\mathcal{M}} \cdot I$ the infinitesimal generator of $\mathcal{M}$,*

$$[\![\exists \Diamond_\alpha \varphi]\!]^{\mathrm{p}}_{\mathcal{M}_{i+1}} = \exp\big((Q - \alpha I)|_{i+1\ldots n}\, t_{\mathrm{cut}}\big)\, [\![\exists \Diamond_\alpha \varphi]\!]^{\mathrm{p}}_{\mathcal{M}_i}$$

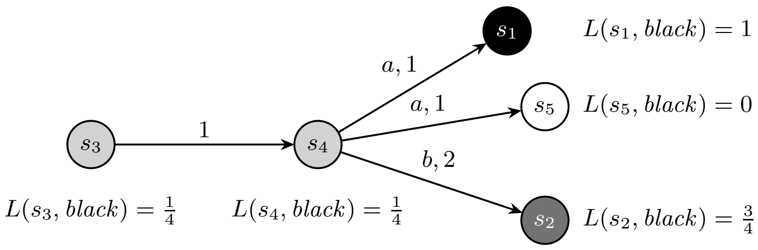
Proof. See the technical report version [4]. □

Time Complexity. Checking one $\exists \Diamond_\alpha$ operator requires to solve a sequence of related CTMC reachability problems. Using Jensen's method (similar to checking CSL properties of CTMCs, see [3,9]), we know that it takes $O((E_{\mathcal{M}} + \alpha)t|R| \cdot |S|)$ arithmetic operations, where t is the maximal time bound. This is comparable to [1, Lemma 19]: The time bound for checking the corresponding formula in a DTMC is in $O(|S|^3 \cdot K)$, where K is the maximal step bound. The time bound t is, in principle, the total of all cutoff times; however, if it is very large, the discount $e^{-\alpha t}$ becomes so small that one can abort the calculation before all cutoff times have passed. For an error bound ε, we have that $e^{-\alpha t} < \varepsilon$ iff $t > \frac{1}{\alpha} \ln \frac{1}{\varepsilon}$.

Proposition 16. *The problem of model checking $[\![\varphi]\!]^{\mathrm{p}}_{\mathcal{M}}$ (not containing $\triangle_\alpha$ operators) can be solved in time $O(|\varphi| \cdot |R| \cdot |S| \frac{E+\alpha}{\alpha} \ln \frac{1}{\varepsilon})$.*

5 Other Scheduler Classes

The article [1] uses the best and worst *history-dependent* scheduler. In the above definitions, we looked for the best and worst positional scheduler. In this section, we illustrate how the choice of scheduler class influences the semantics. To make the scheduler class more clear, we will add a subscript $\mathfrak{D}$ to $\exists$, the main operator



s_1
$L(s_1, black) = 1$
$a, 1$
$a, 1$
s_5
$L(s_5, black) = 0$
1
s_3
s_4
$b, 2$
$L(s_3, black) = \frac{1}{4}$
$L(s_4, black) = \frac{1}{4}$
s_2
$L(s_2, black) = \frac{3}{4}$

Fig. 3. Timed schedulers are better than positional ones

that chooses from the set of schedulers $\mathfrak{D}$. In the following, we will give some examples for the $\exists_{\mathfrak{D}}$ operator for several classes of schedulers. The treatment of the $\forall_{\mathfrak{D}}$ operator always follows the same lines and does not enlighten more.

We designate the (time-abstract) positional schedulers used above by $\mathfrak{P}$, and we will introduce the classes $\mathfrak{TTP}$ of total-time positional schedulers and $\mathfrak{LP}$ of late time-abstract positional schedulers.

5.1 Late Schedulers ($\mathfrak{LP}$)

A CTMDP is *locally uniform* if in every state, the exit rate does not depend on the action chosen (i. e., $\forall s \in S : \forall a, b \in A : E^a(s) = E^b(s)$). In a locally uniform CTMDP, it is possible to delay the choice for one action or another until the moment that the sojourn time in a state has passed. So-called *late* schedulers, defined in [6], are functions from the current state and the sojourn time in this state to an action that is enabled: $D : S \times \mathbb{R}_{\geq 0} \to A$. As in time-bounded reachability [6], a late scheduler can improve on the value of a formula.

Example 17. Consider the CTMDP in Fig. 3, with initial state s_4. The best positional scheduler for $[\![\exists \Diamond_1 \, black]\!]^{\mathrm{P}}(s_4)$ always chooses action b, as the expected utility is higher than when choosing a. However, a late scheduler can base its choice also on t_4, the sojourn time in state s_4. For times $t_4 \in [\ln 3, \ln 4)$, choosing b will not improve $[\![\exists \Diamond_1 \, black]\!]^{\mathrm{P}}(s_4)$ because $e^{-t_4} [\![black]\!]^{\mathrm{P}}(s_2) \leq [\![black]\!]^{\mathrm{P}}(s_4)$, but by choosing a, it improves with probability of $\frac{1}{2}$, because $e^{-t_4} [\![black]\!]^{\mathrm{P}}(s_1) > [\![black]\!]^{\mathrm{P}}(s_4)$. For very short sojourn times $t_4 \in [0, S)$, for some $S \leq \ln 3$, it is more advantageous to choose b. To find S, one solves the integral:

$$[\![\exists_{\mathfrak{LP}} \Diamond_1 \, black]\!]^{\mathrm{P}}(s_4) = \int_0^\infty \mathrm{pdf}\Big(s_4 \xrightarrow{t_4}\Big) \Big(\underbrace{\tfrac{1}{2}\left(\tfrac{1}{4} \sqcup e^{-t_4}\right) + \tfrac{1}{2} \cdot \tfrac{1}{4}}_{\text{action } a \text{ is chosen}} \sqcup \underbrace{\tfrac{3}{4} e^{-t_4}}_{\text{action } b}\Big) \, dt_4$$

The exact switching point is therefore $S = \ln 2$.

5.2 Total-Time Positional Schedulers ($\mathfrak{TTP}$)

A total-time positional scheduler bases its choice not only on the current state, but also on the total time that the path has taken until now. Formally, such a scheduler is a function $D : S \times \mathbb{R}_{\geq 0} \to A$ but the real parameter now indicates the total time spent before entering the current state.

Example 18. Consider the CTMDP in Fig. 3 again. We now regard s_3 as the initial state. Similar to Example 17, the best positional scheduler would always choose b in s_4, but a $\mathfrak{TTP}$-scheduler can improve on this value because it bases its decision also on t_3, the sojourn time in state s_3. A similar integral as above shows that an optimal scheduler chooses b for times $t_3 \in [0, R)$, where $R \approx 0.41903$, and chooses a for $t_3 \in (R, \ln 4)$. Note that this scheduler has to estimate how long the sojourn time in s_4 will be when it makes its decision; therefore, it is less exact than the late scheduler of Example 17.

5.3 History-Dependent Schedulers ($\mathfrak{H}$)

A history-dependent scheduler, in principle, has access to the full history, i. e. to the complete path to make a decision. In practice, it is enough to use the highest utility $u_{\max}$ achieved until now to decide which actions still could improve on the utility of $[\![\exists_{\mathfrak{H}} \Diamond_\alpha \varphi]\!]^{\mathrm{P}}$. Additionally, similar in Example 18, the decision should take into account the discount incurred between the time $t_{\max}$ when that utility was achieved and the current time t. Therefore, a history-dependent scheduler has enough information if it knows the value $u_{\max} e^{\alpha(t - t_{\max})}$.

This is basically the idea used by [1] to find a model checking algorithm for the path semantics of MDPs with history-dependent schedulers. Why can't we use this idea for CTMDPs? The problem is that in discrete-time systems, discount factors are powers of a constant $\rho \in (0, 1)$. As a consequence, only a finite number of the powers $u_{\max}, \rho^{-1} u_{\max}, \rho^{-2} u_{\max}, \ldots$ is relevant to find $[\![\exists_{\mathfrak{H}} \Diamond_\rho \varphi]\!]^{\mathrm{P}}_{\mathcal{P}}$. This allows to define a linear program with finitely many variables to calculate the utilities. However, in the continuous-time case, the corresponding linear program would have infinitely many variables, so one cannot prove termination.

6 Conclusion

It is possible to extend the DCTL semantics with time, to continuous-time Markov chains and MDPs. No big surprises have happened; the extension is pretty much straightforward. The distinction between the fixpoint and the path semantics known from the discrete-time case also applies to the continuous-time case. Properties interpreted under the fixpoint semantics can be easily checked using the same algorithms as in the discrete-time case; because fixpoint semantics do not look further ahead than one step, a variation on the expected time to take a transition (the reciprocal of the exit rate) is the discount per step. Properties interpreted under the path semantics are more difficult to verify; this article provides an algorithm for CTMCs (without nondeterminism).

To simplify the analysis, I have chosen to fix a single discount rate per property. This corresponds to an exponential decay of utility over time. One can without much work extend the current presentation and give each state its own discount rate (e. g. by adapting the matrix (3)).

Another way to make analysis of such temporal properties possible is taking a finite horizon. Everything before the time limit is equally relevant, and everything after is equally irrelevant. For example, time-bounded until formulas in CSL [3,9] use a finite horizon. Depending on the situation, one or the other way of giving more weight to the near future is better-suited. If one does not want to predict for how long exactly a computer system will be used, it seems advantageous to use an analysis method that does not completely ignore a possible breakdown shortly after some deadline.

Rewards in Markov chains can also be used to express quantitative properties [2]. DCTL still allows to combine multiple aspects of utility in ways that are not normally offered by analysis methods for chains with (multiple) rewards.

Acknowledgement. This work has been partially supported by DFG/NWO Bilateral Research Programme ROCKS. I thank Axel Legay (INRIA/IRISA Rennes, France) and Mariëlle Stoelinga (Universiteit Twente, The Netherlands) for helpful discussions, partly during a visit to INRIA/IRISA Rennes. I have used Maple [5] to check the example calculations.

References

1. de Alfaro, L., Faella, M., Henzinger, T.A., Majumdar, R., Stoelinga, M.: Model checking discounted temporal properties. Theoretical Computer Science 345(1), 139–170 (2005)
2. Baier, C., Cloth, L., Haverkort, B.R., Hermanns, H., Katoen, J.-P.: Performability assessment by model checking of Markov reward models. Formal Methods in System Design 36(1), 1–36 (2010)
3. Baier, C., Hermanns, H., Katoen, J.-P., Haverkort, B.R.: Efficient computation of time-bounded reachability probabilities in uniform continuous-time Markov decision processes. Theoretical Computer Science 345(1), 2–26 (2005)
4. Jansen, D.N.: More or less true: DCTL for continuous-time MDPs. Tech. Rep. ICIS–R13006, Radboud Universiteit, Nijmegen (2013), `http://www.cs.ru.nl/research/reports`
5. Maplesoft: technical computing software for engineers, mathematicians, scientists, instructors and students, `http://www.maplesoft.com/`
6. Neuhäußer, M.R., Stoelinga, M., Katoen, J.-P.: Delayed nondeterminism in continuous-time Markov decision processes. In: de Alfaro, L. (ed.) FOSSACS 2009. LNCS, vol. 5504, pp. 364–379. Springer, Heidelberg (2009)
7. Wolovick, N., Johr, S.: A characterization of meaningful schedulers for continuous-time Markov decision processes. In: Asarin, E., Bouyer, P. (eds.) FORMATS 2006. LNCS, vol. 4202, pp. 352–367. Springer, Heidelberg (2006)
8. Zadeh, L.A.: Fuzzy sets, fuzzy logic, and fuzzy systems. World Scientific, Singapore (1996)
9. Zhang, L., Jansen, D.N., Nielson, F., Hermanns, H.: Efficient CSL model checking using stratification. Logical Methods in Computer Science 8(2), paper 17 (2012)

Incremental Language Inclusion Checking for Networks of Timed Automata

Willibald Krenn, Dejan Ničković, and Loredana Tec[*]

AIT Austrian Institute of Technology GmbH
Vienna, Austria

Abstract. Checking the language inclusion between two models is a fundamental problem arising in application areas such as formal verification or refinement in top-down design. We propose an incremental procedure for checking the language inclusion between two real-time specifications, modeled as networks of deterministic timed automata, where the two specifications are equivalent up to one component. For such classes of systems we aim to improve the efficiency of the language inclusion check by exploiting the compositional nature of the problem and avoiding the explicit parallel composition of the timed automata in the network. We first develop a generic procedure that gives freedom to specific implementation choices. We then propose an instantiation of the procedure that is based on bounded model checking techniques. We illustrate the application of our approach in a case study and discuss promising experimental results.

1 Introduction

In formal methods theory and applications, deciding *language inclusion* is a fundamental problem. Checking language inclusion between two models A' and A, denoted by $L(A') \subseteq L(A)$, consists in checking whether the set of traces of A' is included in the set of traces specified by A. The problem naturally arises in several applications, such as model checking or refinement checking in a top-down design process. In model checking we are interested in verifying whether a formal model of a system A satisfies a high-level property P, denoted by $A \models P$. This amounts to checking the language inclusion $L(A) \subseteq L(P)$. In a top-down design process, language inclusion can be used as a *refinement* relation between a system represented at two levels of abstraction. It follows that a model A' is a refinement of A if and only if $L(A') \subseteq L(A)$, meaning that A' is more precise and closer to an actual implementation of the design than A. We note that if the model A satisfies a property P and A' refines A, then A' also satisfies P.

[*] The research leading to these results has received funding from ARTEMIS Joint Undertaking under grants agreement numbers 269335 (MBAT), 269265 (pSafeCer) and 295373 (nSafeCer), and from national fundings (Federal Ministry for Transport, Innovation and Technology and Austrian Research Promotion Agency) under program line Trust in IT Systems, project number 829583 (TRUFAL).

In the context of real-time systems, the language inclusion problem $L(A') \subseteq L(A)$ between two timed automata (TA) models A' and A is decidable if A is *deterministic* [3]. In addition, we have the following compositionality result for timed automata distinguishing between *input* and *output* actions (TAIO). Suppose we are given three *receptive* TAIOs A', A, and B. If A' refines A then $A' \parallel B$ refines $A \parallel B$ (see [11]).

In this paper we consider the problem of language inclusion between two TAIO models A' and A. We assume that both models are given as networks of communicating deterministic and receptive TAIOs. We consider the setting where the two specifications are equivalent up to one component, that is $A' = A'_1 \parallel A_2 \parallel \cdots \parallel A_n$ and $A = A_1 \parallel A_2 \cdots \parallel A_n$. The problem of checking language inclusion for this class of specifications is motivated by practical considerations as it arises in several application areas such as mutation-based test case generation and top-down incremental design.

Mutation-based test case generation [1,14,7,10] is a special instance of model-based test case generation [16], in which a specification model A is deliberately altered by inserting a small fault, resulting in a *mutant* A' of the original model. The inserted fault plays the role of a *test purpose* and is used to guide the test case generation process. The main step in mutation-based test case generation consists in checking $L(A') \subseteq L(A)$. If the language of the mutant A' is not included in the language of the original model A, a *witness* trace $\sigma \in L(A') \backslash L(A)$ is obtained and is used to create a test case.

In top-down incremental design, the problem arises whenever an individual component A_1 of the specification model $A = A_1 \parallel A_2 \parallel \cdots \parallel A_n$ is refined to a more precise component A'_1, resulting in a new specification model $A' = A'_1 \parallel A_2 \parallel \cdots \parallel A_n$. Given this case, it is sufficient to show that A'_1 refines A_1 in order to infer that A' refines A.

We propose a procedure for *incrementally* checking the language inclusion problem $L(A') \subseteq L(A)$ for networks of timed automata, where $A' = A'_1 \parallel A_2 \parallel \cdots \parallel A_n$ and $A = A_1 \parallel A_2 \parallel \cdots \parallel A_n$. The main motivation is to improve the performance of finding witnesses for language inclusion violation, as witnesses are used to construct test cases. Intuitively, the procedure exploits the compositional structure of the model and avoids taking the full composition of the automata, thus aiming at improving the efficiency of the language inclusion check for this class of real-time models. It first checks the simpler problem $L(A'_1) \subseteq L(A_1)$. If the check is successful, then we use the compositionality result to infer that $L(A') \subseteq L(A)$. Otherwise, a trace $\sigma \in L(A'_1) \backslash L(A_1)$ witnessing the non language inclusion between A'_1 and A_1 is obtained. Given σ, we check whether it can be extended to a counter-example witnessing $L(A') \not\subseteq L(A)$. If such an extension exists, we are done. Otherwise we refine A'_1 by removing a set of spurious behaviors extrapolated from σ and repeat the procedure.

We first develop a generic incremental procedure for checking language inclusion (Section 3). This procedure uses abstract operations defined in terms of the properties they need to satisfy. In other words, the operations used in the generic procedure are non-effective and allow for various implementation choices. Then, we refine the generic algorithm and propose its instantiation based on bounded symbolic model checking techniques (Section 4). We illustrate the approach on a real-time variant of the dining philosophers problem and present promising experimental results (Section 5). We show that the algorithm can be used to incrementally check timed input/output conformance (tioco) [11] between a specification and an implementation. Finally, we illustrate how this concrete procedure is applied in the context of mutation-based testing (Section 6) .

This paper complements [17] and [11], which derive compositionality results for language inclusion checking of untimed and timed systems, respectively, but do not consider incremental checking of the problem. In [4], the authors propose another variant of incremental language inclusion checking for timed systems. They check whether a timed specification, expressed as a network of timed components satisfies an ω-regular property. In their iterative algorithm, the timed specification is first abstracted to an untimed approximation, and in every successive step, some of the time constraints are re-added to the model. Our framework is similar to the general counter-example guided abstraction refinement (CEGAR) [8]. In our case the original system is naturally abstracted by leaving out components from the original model in the language inclusion check, and then the abstracted model is refined in successive steps from spurious counter-examples. Finally, a similar approach using a satisfiability modulo theory (SMT) solver is developed in [15] to incrementally synthesize time-triggered schedules.

2 Timed Automata with Inputs and Outputs

The time domain that we consider is the set $\mathbb{R}_{\geq 0}$ of non-negative reals. We denote by Σ the finite set of actions, partitioned into disjoint sets Σ^I and Σ^O of input and output actions. A *time sequence* is a finite non-decreasing sequence of non-negative reals. A *timed trace* σ defined over Σ is a finite alternating sequence of time delays and actions of the form $t_1 \cdot a_1 \cdots t_k \cdot a_k$, where for all $1 \leq i \leq k$, $a_i \in \Sigma$ and $(t_i)_{i \geq 1}$ is a time sequence. We denote by ϵ the *empty* timed trace.

Let $\mathcal{C}$ be a finite set of *clock* variables. Clock *valuation* $v(c)$ is a function $v : \mathcal{C} \rightarrow \mathbb{R}_{\geq 0}$ assigning a real value to every clock $c \in \mathcal{C}$. We denote by $\mathcal{H}$ the set of all clock valuations and by $\mathbf{0}$ the valuation assigning 0 to every clock in $\mathcal{C}$. Let $v \in \mathcal{H}$ be a valuation and $t \in \mathbb{R}_{\geq 0}$, we then have $v + t$ defined by $(v + t)(c) = v(c) + t$ for all $c \in \mathcal{C}$. For a subset ρ of $\mathcal{C}$, we denote by $v[\rho]$ the valuation such that for every $c \in \rho$, $v[\rho](c) = 0$ and for every $c \in \mathcal{C} \backslash \rho$, $v[\rho](c) = v(c)$. A *clock constraint* φ is a conjunction of predicates over clock variables in $\mathcal{C}$ defined by the grammar $\varphi ::= c \circ k \mid \varphi_1 \wedge \varphi_2$, where $c \in \mathcal{C}$, $k \in \mathbb{N}$ and $\circ \in \{<, \leq, =, \geq, >\}$. Given a clock valuation $v \in \mathcal{H}$, we write $v \models \varphi$ when v satisfies φ. We are now ready to formally define *input/output* timed automata.

Definition 1 (TAIO). *An* input/output timed automaton (TAIO) *A is a tuple $(Q_A, \hat{q}_A, \Sigma_A^I, \Sigma_A^O, \mathcal{C}_A, I_A, \Delta_A, F_A)$, where:*

– *Q_A is a finite set of* locations *and $\hat{q}_A \in Q_A$ is the* initial *location;*

– Σ_A^I *is a finite set of* input actions *and* Σ_A^O *is a finite set of* output actions, *such that* $\Sigma_A^I \cap \Sigma_A^O = \emptyset$*; we denote by* Σ_A *the set of actions* $\Sigma_A^I \cup \Sigma_A^O$*;*
– $\mathcal{C}_A$ *is a finite set of* clock *variables;*
– I_A *is the location* invariant*, a conjunction of constraints of the form* $c < d$ *or* $c \leq d$*, where* $c \in \mathcal{C}_A$ *and* $d \in \mathbb{N}$*;*
– Δ_A *is a finite set of* transitions *of the form* (q, a, g, ρ, q')*, where*

  - $q, q' \in Q_A$ *are the* source *and the* target *locations;*
  - $a \in \Sigma_A$ *is the transition* action*;*
  - g *is a* guard*, a conjunction of constraints of the form* $c \circ d$*, where* $c \in \mathcal{C}_A$*,* $\circ \in \{<, \leq, =, \geq, >\}$ *and* $d \in \mathbb{N}$*;*
  - $\rho \subseteq \mathcal{C}_A$ *is a set of clocks to be* reset*;*

– $F_A \subseteq Q_A$ *is the set of* accepting *locations.*

We say that a TAIO A is *deterministic* if for all transitions (q, a, g_1, ρ_1, q_1) and (q, a, g_2, ρ_2, q_2) in Δ_A, $q_1 \neq q_2$ implies that $g_1 \wedge g_2 = \emptyset$. Given an arbitrary location $q \in Q_A$, and an arbitrary action $a \in \Sigma_A^I$, let $g^{q,a} = (g^1 \vee \ldots \vee g^k)$, where $\{g^i\}_i$ are guards of the outgoing transitions of q labeled by a. We say that a TAIO A is *receptive* if and only if for all $q \in Q_A$ and $a \in \Sigma_A^I$, $I_A(q) \rightarrow g^{q,a}$. From now on, we restrict our attention to deterministic and receptive TAIOs.

Semantics. The *semantics* of a TAIO is given by the *timed input/output transition system* (TIOTS) $[[A]] = (S_A, \hat{s}_A, \mathbb{R}_{\geq 0}, \Sigma_A, T_A, \mathcal{F}_A)$, where

– $S_A = \{(q, v) \in Q_A \times \mathcal{H} \mid v \models I_A(q)\}$ and $\hat{s}_A = (\hat{q}_A, \mathbf{0})$;
– $T_A \subseteq S_A \times (\Sigma_A \cup \mathbb{R}_{\geq 0}) \times S_A$ is the transition relation consisting of *discrete* and *timed* transitions such that:

  - *Discrete transitions:* $((q, v), a, (q', v')) \in T_A$, where $a \in \Sigma_A$, if there exists a transition (q, a, g, ρ, q') in Δ_A, such that: (1) $v \models g$; (2) $v' = v[\rho]$ and (3) $v' \models I_A(q')$; and
  - *Timed transitions:* $((q, v), t, (q, v + t)) \in T_A$, where $t \in \mathbb{R}_{\geq 0}$, if $v + t \models I_A(q)$;

– $\mathcal{F}_A \subseteq S_A$ such that $(q, v) \in \mathcal{F}_A$ iff $q \in F_A$.

A *run* r of a TAIO A is the sequence of alternating timed and discrete transitions of the form

$$(q^1, v^1) \xrightarrow{t^1} (q^1, v^1 + t^1) \xrightarrow{\delta^1} (q^2, v^2) \xrightarrow{t^2} \cdots \xrightarrow{t^k} (q^k, v^k + t^k) \xrightarrow{\delta^k} (q^{k+1}, v^{k+1}),$$

where $q^1 = \hat{q}_A$, $v^1 = \mathbf{0}$ and $\delta^i = (q^i, a^i, g^i, \rho^i, q^{i+1}) \in \Delta_A$. A run r is *accepting* if the last location q^{k+1} is accepting. The run r of A induces the timed trace $\sigma = t^1 \cdot a^1 \cdot t^2 \cdots t^k \cdot a^k$ defined over Σ_A. We denote by $L(A)$ the set of timed traces induced by all accepting runs of A.

Parallel Composition. In order to model a network of interacting components, we need the *parallel composition* operator. We first define the notion of *composabilty* of two TAIOs, where two TAIOs A and B are composable if their sets of output actions are disjoint.

Definition 2 (Composability). *Let A and B be two* TAIO*s. We say that A and B are* composable *if $\Sigma_A^O \cap \Sigma_B^O = \emptyset$.*

Definition 3 (Parallel composition). *Let A and B be two composable* TAIO*s. Then their* parallel composition*, denoted by $A \parallel B$, is the* TAIO $(Q_{A\parallel B}, \hat{q}_{A\parallel B}, \Sigma^I_{A\parallel B}, \Sigma^O_{A\parallel B}, \mathcal{C}_{A\parallel B}, I_{A\parallel B}, \Delta_{A\parallel B}, F_{A\parallel B})$*, where*

- $Q_{A\parallel B} = Q_A \times Q_B$ *and* $\hat{q}_{A\parallel B} = (\hat{q}_A, \hat{q}_B)$;
- $\Sigma^O_{A\parallel B} = \Sigma^O_A \cup \Sigma^O_B$ *and* $\Sigma^I_{A\parallel B} = (\Sigma^I_A \cup \Sigma^I_B) \backslash \Sigma^O_{A\parallel B}$;
- $\mathcal{C}_{A\parallel B} = \mathcal{C}_A \cup \mathcal{C}_B$;
- $I_{A\parallel B}(q_A, q_B) = I_A(q_A) \wedge I_B(q_B)$;
- $\Delta_{A\parallel B}$ *is the smallest set of transitions such that:*
  - *For* $(q_A, q_B) \in Q_A \times Q_B$ *and* $a \in (\Sigma^I_A \backslash \Sigma^O_B) \cup (\Sigma^O_A \backslash \Sigma^I_B)$*, if* $(q_A, a, g, \rho, q'_A) \in \Delta_A$*, then* $((q_A, q_B), a, g, \rho, (q'_A, q_B)) \in \Delta_{A\parallel B}$;
  - *For* $(q_A, q_B) \in Q_A \times Q_B$ *and* $a \in (\Sigma^I_B \backslash \Sigma^O_A) \cup (\Sigma^O_B \backslash \Sigma^I_A)$*, if* $(q_B, a, g, \rho, q'_B) \in \Delta_B$*, then* $((q_A, q_B), a, g, \rho, (q_A, q'_B)) \in \Delta_{A\parallel B}$;
  - *For* $(q_A, q_B) \in Q_A \times Q_B$ *and* $a \in (\Sigma_A \cap \Sigma_B)$*, if* $(q_A, a, g_A, \rho_A, q'_A) \in \Delta_A$ *and* $(q_B, a, g_B, \rho_B, q'_B) \in \Delta_B$*, then* $((q_A, q_B), a, g_A \wedge g_B, \rho_A \cup \rho_B, (q'_A, q'_B)) \in \Delta_{A\parallel B}$;
- $F_{A\parallel B} = \{(q_A, q_B) \mid q_A \in F_A \text{ and } q_B \in F_B\}$.

The refinement expressed as language inclusion is compositional for deterministic and receptive TAIOs[1].

Proposition 1. *Let A and A' be two receptive* TAIO*s defined over the same alphabet, and B be a receptive* TAIO *composable with A and A'. Then, we have*

$$L(A') \subseteq L(A) \rightarrow L(A' \parallel B) \subseteq L(A \parallel B)$$

Product. We define the *product* of two TAIOs A and B defined over the same alphabet $\Sigma = \Sigma^I \cup \Sigma^O$ and denoted by $A \times B$, as the TAIO that contains all the traces which are both traces of A and B.

Definition 4 (Product). *Let A and B be two* TAIO*s defined over the alphabet $\Sigma^I \cup \Sigma^O$. Then the* product *of A and B, denoted by $A \times B$, is the* TAIO $(Q_{A\times B}, \hat{q}_{A\times B}, \Sigma^I, \Sigma^O, \mathcal{C}_{A\times B}, I_{A\times B}, \Delta_{A\times B}, F_{A\times B})$*, where*

- $Q_{A\times B} = Q_A \times Q_B$ *and* $\hat{q}_{A\times B} = (\hat{q}_A, \hat{q}_B)$;
- $\mathcal{C}_{A\times B} = \mathcal{C}_A \cup \mathcal{C}_B$;
- $I_{A\times B}(q_A, q_B) = I_A(q_A) \wedge I_B(q_B)$;

[1] A similar result is shown in [11] for TAIOs with urgent transitions.

– *$\Delta_{A\times B}$ is the smallest set of transitions such that for $(q_A, q_B) \in Q_A \times Q_B$ and $a \in \Sigma$, if $(q_A, a, g_A, \rho_A, q'_A) \in \Delta_A$ and $(q_B, a, g_B, \rho_B, q'_B) \in \Delta_B$, then $((q_A, q_B), a, g_A \wedge g_B, \rho_A \cup \rho_B, (q'_A, q'_B)) \in \Delta_{A\times B}$;*
– *$F_{A\times B} = \{(q_A, q_B) \mid q_A \in F_A \text{ and } q_B \in F_B\}$.*

Proposition 2. *Let A and B be two* TAIO*s defined over the alphabet $\Sigma^I \cup \Sigma^O$. Then, we have that $L(A \times B) = L(A) \cap L(B)$.*

3 Incremental Language Inclusion - A Generic Algorithm

In this section, we propose a generic incremental language inclusion checking algorithm for networks of timed automata. Given three deterministic and receptive TAIOs A, A' and B, the algorithm checks whether $L(A' \parallel B) \subseteq L(A \parallel B)$ and gives a counter-example in the failing case. The algorithm is generic due to the fact that it applies abstract operations which are defined in terms of their properties. Abstract operations allow for different implementations, as long as their instantiation satisfies the required properties. We first introduce abstract operations which are used in the algorithm, and then present the algorithm itself.

Language Inclusion. Given two TAIOs A and A' defined over the same alphabet, the language inclusion operation $\mathrm{LI}(A, A')$ is a function that checks whether the language of A' is contained in the language of A. It gives a counter-example in the case of non-containment.

Definition 5. *Let A and A' be two* TAIO*s defined over the alphabet Σ. The language inclusion operation* LI *is a function such that*

$$\mathrm{LI}(A', A) = \begin{cases} (\mathrm{true}, \epsilon) \ \textit{if}\ L(A') \subseteq L(A) \\ (\mathrm{false}, \sigma) \ \textit{otherwise} \end{cases}, \textit{where}\ \sigma \in L(A')\backslash L(A).$$

Trace Extrapolation Given two TAIOs A and A' sharing the same alphabet and $\sigma \in L(A')\backslash L(A)$, the trace extrapolation operation, denoted by T, transforms σ into a TAIO which contains σ together with the set of other witnesses of non-containment of $L(A')$ in $L(A)$.

Definition 6. *Let A' and A be two* TAIO*s defined over the alphabet Σ such that $L(A') \not\subseteq L(A)$, and let $\sigma \in L(A')\backslash L(A)$. Then, the trace extrapolation operation, denoted by T, is a function $B = T(A', A, \sigma)$, where B is a* TAIO *defined over Σ such that $\sigma \in L(B)$ and $L(B) \subseteq L(A')\backslash L(A)$.*

Emptiness. Given two composable TAIOs A and B, the emptiness operation checks if the parallel composition $A \parallel B$ is empty, and returns a witness of non-emptiness in the failing case.

Definition 7. *Let A and B be two composable* TAIO*s. Then the emptiness operation, denoted by* Emp*, is a function such that*

$$\textsc{Emp}(A, B) = \begin{cases} (\mathrm{true}, \epsilon) \ \textit{if}\ L(A \parallel B) = \emptyset \\ (\mathrm{false}, \sigma) \ \textit{otherwise} \end{cases}, \textit{where}\ \sigma \in L(A \parallel B).$$

TAIO Difference given two TAIOs A and A' defined over the same alphabet, the difference operation, denoted by DIFF, computes a new TAIO which contains all of the behaviors that are in A' but not in A.

Definition 8. *Let A and A' be two* TAIO*s defined over the alphabet Σ. Then, the difference operation, denoted by* DIFF*, transforms A and A' into a* TAIO *B, such that B is defined over the alphabet Σ and $L(B) = L(A')\backslash L(A)$.*

Algorithm. We are now ready to describe the incremental language inclusion generic algorithm[2] INCLI, shown in Algorithm 1, which works as follows:

1. Check $L(A') \subseteq L(A)$ with LI (line 3). If the language of A' is contained in the one of A, then the language of $A' \parallel B$ is also included in the language of $A \parallel B$ (Proposition 1) and the algorithm terminates giving an empty witness trace. Otherwise, a trace σ' witnessing non-containment of traces of A' in A is returned and the algorithm proceeds to step 2.
2. The witness trace σ' is transformed into a TAIO A'' by applying T (line 7), and A'' defines a set of traces which are all in A' but not in A.
3. The algorithm checks with EMP whether $A'' \parallel B$ is empty (line 8). If the composition is non-empty, then there exists a trace σ which is in $A' \parallel B$ but not in $A \parallel B$, thus witnessing violation of language inclusion of $A' \parallel B$ into $A \parallel B$. In that case, the algorithm terminates and gives the witness trace σ. Otherwise, the algorithm proceeds to step 4.
4. Applying DIFF, the TAIO A' is replaced by $A'\backslash A''$ (line 10), thus removing the behaviors of A'' in A'. The algorithms backtracks to step 1.

Theorem 1 (Partial correctness). *Let A, A' and B be* TAIO*s such that A and A' are defined over the same alphabet and B is composable with A and A' and let $(b, \sigma) =$* INCLI*(A, A', B). We have that*

1. *if $b = \text{false}$, then $L(A' \parallel B) \not\subseteq L(A \parallel B)$, $\sigma \in L(A' \parallel B)$ and $\sigma \notin L(A \parallel B)$;*
2. *if $b = \text{true}$, then $L(A' \parallel B) \subseteq L(A \parallel B)$.*

4 *k*-Bounded Incremental Language Inclusion

In this section, we develop an instantiation of Algorithm 1 based on bounded model checking techniques. It extends the language inclusion procedure for flat TAIOs presented in [2]. Similar encodings for TA decision problems were proposed in [13,5] for reachability and in [6] for language inclusion.

[2] We restrict the algorithm to networks of 2 automata for simplicity of presentation, its extension to a network of n automata is straight-forward.

Algorithm 1. IncLI

```
Input: $A, A', B$
Output: $(b, \sigma)$
 1: $done \leftarrow false$
 2: while $\neg done$ do
 3:     $(b, \sigma') \leftarrow \mathrm{LI}(A', A)$
 4:     if $b$ then
 5:         $done \leftarrow true$
 6:     else
 7:         $A'' \leftarrow T(A, A', \sigma')$
 8:         $(b, \sigma) \leftarrow \textsc{Emp}(A'', B)$
 9:         if $b$ then
10:             $A' \leftarrow \textsc{Diff}(A', A'')$
11:         else
12:             $done \leftarrow true$
13:         end if
14:     end if
15: end while
16: return $(b, \sigma)$
```

Preliminaries. Let A be a TAIO and k a bound. We denote by $\mathrm{loc}_A : Q_A \rightarrow \{1, \ldots, |Q_A|\}$ the function assigning a unique integer to every location in A. We denote by $\mathrm{act}_A : \Sigma_A \rightarrow \{1, \ldots, |\Sigma_A|\}$ the function assigning a unique integer to every action in A. Let $\mathcal{A} = \{\alpha^1, \ldots, \alpha^k\}$ be the set of variables ranging over Σ_A, where α^i encodes the action in A applied in the i^{th} discrete step. We denote by $D = \{d^1, \ldots, d^k\}$ the set of real-valued variables, where d^i encodes the delay action in A applied in the i^{th} time step. X_A the set of variables $\{x^1, \ldots, x^{k+1}\}$ that range over the domain $\{1, \ldots, |Q|\}$, where x^i encodes the location of A after the i^{th} step. Let $\mathrm{acc}_A(x^i)$ be a predicate which is true iff x^i encodes an accepting location. Let C_A^i denote the set of real variables obtained by renaming every clock $c \in \mathcal{C}_A$ by c^i. We denote by $C_A = \bigcup_{i=1}^{k+1} C_A^i \cup \bigcup_{i=1}^{k+1} C_A^{*,i}$ the set of real (clock valuation) variables, where $c^{*,i} \in C_A^{*,i}$ and $c^i \in C_A^i$ encode the valuation of the clock $c \in \mathcal{C}_A$ after the i^{th} timed and discrete step, respectively.

Path Predicate. Let A be a TAIO. We encode a valid path in A of size k with the path_A^k predicate, defined as

$$\mathrm{path}_A^k(\mathcal{A}, D, X_A, C_A) \equiv \mathrm{init}_A(X_A, C_A) \wedge \bigwedge_{i=1}^{k} \mathrm{step}_A^i(\mathcal{A}, D, X_A, C_A). \tag{1}$$

Formally, we have the initial state predicate defined as

$$\mathrm{init}_A(X_A, C_A) \equiv x^1 = \mathrm{loc}_A(\hat{q}_A) \wedge \bigwedge_{c \in \mathcal{C}_A} (c^1 = 0). \tag{2}$$

A step in the path of A consists of a timed step followed by a discrete step, formalized by the predicate

$$\begin{aligned} \mathrm{step}_A^i(\mathcal{A}, D, X_A, C_A) \equiv & \bigvee_{q \in Q_A} \mathrm{tStep}_{A,q}^i(D, X_A, C_A) \;\; \wedge \\ & \bigvee_{\delta \in \Delta_A} \mathrm{dStep}_{A,\delta}^i(\mathcal{A}, X_A, C_A). \end{aligned} \tag{3}$$

The i^{th} time step in a location $q \in Q$ is expressed with

$$\mathrm{tStep}_{A,q}^i(D, X_A, C_A) \equiv x^i = \mathrm{loc}_A(q) \wedge \mathrm{tDelay}_A^i(D, C_A) \wedge I_A(q)[\mathcal{C}_A \backslash C_A^{*,i}], \tag{4}$$

where $I_A(q)[\mathcal{C}_A \backslash C_A^{*,i}]$ is the invariant of q, and every clock $c \in \mathcal{C}_A$ appearing in the invariant is replaced by $c^{*,i}$, while $\mathrm{tDelay}_A^i(D, C_A)$ is the predicate expressing the passage of time:

$$\mathrm{tDelay}_A^i(D, C_A) \equiv \bigwedge_{c \in \mathcal{C}_A} (c^{*,i} - c^i) = d^i. \tag{5}$$

Finally, a discrete step in A is formalized with the predicate

$$\begin{aligned}\mathrm{dStep}^i_{A,\delta}(\mathcal{A}, X_A, C_A) \equiv x^i = \mathrm{loc}_A(q) \wedge \alpha^i = \mathrm{act}_A(a) \wedge g[\mathcal{C}_A \backslash C_A^{*,i}] \wedge \\ \mathrm{res}^i_{A,\rho}(C_A) \wedge x^{i+1} = \mathrm{loc}_A(q') \wedge I_A(q')[\mathcal{C}_A \backslash C_A^{i+1}],\end{aligned} \tag{6}$$

where $g[\mathcal{C}_A \backslash C_A^{*,i}]$ denotes the guard of δ, in which every clock $c \in \mathcal{C}$ is substituted by $c^{*,i}$, and the reset action is expressed with

$$\mathrm{res}^i_{A,\rho}(C_A) \equiv \bigwedge_{c \in \rho} c^{i+1} = 0 \wedge \bigwedge_{c \notin \rho} c^{i+1} = c^{*,i}. \tag{7}$$

***k*-Bounded Language Inclusion.** Let A and A' be two TAIOs defined over the same alphabet Σ. Given an integer $k > 0$, we define the predicate $\textsc{BndLI}^k_{A,A'}$ as follows:

$$\begin{aligned}\textsc{BndLI}^k_{A,A'} \equiv & \textstyle\bigwedge_{i=1}^k (d^i \geq 0 \wedge \alpha^i \geq 1 \wedge \alpha^i \leq |\Sigma|) & \wedge \\ & i \geq 1 \wedge i \leq k & \wedge \\ & \mathrm{path}^i_{A'}(\mathcal{A}, D, X_{A'}, C_{A'}) \wedge \mathrm{acc}_{A'}(x^{i+1}_{A'}) & \wedge \\ & \mathrm{path}^{i-1}_A(\mathcal{A}, D, X_A, C_A) & \wedge \\ & ((\mathrm{step}^i_A(\mathcal{A}, D, X_A, C_A) \wedge \neg\mathrm{acc}_A(x^{i+1}_A)) & \vee \\ & \overline{\mathrm{step}}^i_A(\mathcal{A}, D, X_A, C_A)) &\end{aligned} \tag{8}$$

where $\overline{\mathrm{step}}^i_A$ expresses the fact that no transition is enabled in the last step[3] in A and is defined as

$$\begin{aligned}\overline{\mathrm{step}}^i_A(\mathcal{A}, D, X_A, C_A) &\equiv \textstyle\bigwedge_{q \in Q_A} \overline{\mathrm{tStep}}^i_{A,q}(D, X_A, C_A) \quad \vee \\ &\quad \textstyle\bigwedge_{\delta \in \Delta_A} \overline{\mathrm{dStep}}^i_{A,\delta}(\mathcal{A}, X_A, C_A). \\ \overline{\mathrm{tStep}}^i_{A,q}(D, X_A, C_A) &\equiv x^i \neq \mathrm{loc}_A(q) \vee (\mathrm{tDelay}^i_A(D, C_A) \wedge \neg I_A(q)[\mathcal{C}_A \backslash C_A^{*,i}]), \\ \overline{\mathrm{dStep}}^i_{A,\delta}(\mathcal{A}, X_A, C_A) &\equiv x^i \neq \mathrm{loc}_A(q) \vee \alpha^i \neq \mathrm{act}_A(a) \vee \neg g[\mathcal{C}_A \backslash C_A^{*,i}] \quad \vee \\ &\quad (x^{i+1} = \mathrm{loc}_A(q') \wedge \mathrm{res}^i_{A,\rho}(C_A) \wedge \neg I_A(q')[\mathcal{C}_A \backslash C_A^{i+1}]).\end{aligned} \tag{9}$$

Since A is deterministic, if the formula $\textsc{BndLI}^k_{A,A'}$ is satisfiable, then $L(A') \not\subseteq L(A)$ and for $\sigma = d^1 \cdot \alpha^1 \cdots d^i \cdot \alpha^i$, we have $\sigma \in L(A')$ and $\sigma \notin L(A)$.

***k*-Bounded Emptiness.** Let A be a TAIO. Given an integer $k > 0$, we define the predicate $\textsc{BndEmp}^k_A$ as follows:

$$\begin{aligned}\textsc{BndEmp}^k_A \equiv & \textstyle\bigwedge_{i=1}^k (d^i \geq 0 \wedge \alpha^i \geq 1 \wedge \alpha^i \leq |\Sigma_A|) & \wedge \\ & i \geq 1 \wedge i \leq k & \wedge \\ & \mathrm{path}^i_A(\mathcal{A}, D, X_A, C_A) & \wedge \\ & \mathrm{acc}_A(x^{i+1}_A) &\end{aligned}$$

If the formula $\textsc{BndEmp}^k_A$ is satisfiable, then $L(A) \neq \emptyset$ and for $\sigma = d^1 \cdot \alpha^1 \cdots d^i \cdot \alpha^i$, we have $\sigma \in L(A)$.

[3] Note that it is not equivalent to $\neg\mathrm{step}^i_A$ because special care must be given to time delay and clock updates.

Witness Extrapolation. Let A and A' be two TAIOs such that $L(A') \nsubseteq L(A)$, and $\sigma = \sigma' \cdot t \cdot a$ a timed trace of size k such that $\sigma' \in L(A') \cap L(A)$, $\sigma \in L(A')$ and $\sigma \notin L(A)$. Let $r_{A'} = q^1_{A'}, \delta^1_{A'}, \ldots, q^k_{A'}, \delta^k_{A'} q^{k+1}_{A'}$ be the run of A' induced by the timed word σ, and $r_A = q^1_A, \delta^1_A, \ldots, q^{k-1}_A, \delta^{k-1}_A q^k_A$ be the run of A induced by the timed word σ'. We define the *witness extrapolation* function $B = \text{witness}^{\#}(\sigma, A, A')$, where $B = (Q_B, \hat{q}_B, \Sigma^I_B, \Sigma^O_B, \mathcal{C}_B, I_B, \Delta_B, F_B)$ such that:

- $Q_B = \{q^1_B, \ldots, q^{k+1}_B\}$ such that $\hat{q}_B = q^1_B$ and $q^i_B = (q^i_{A'}, q^i_A)$ for all $i \leq k$;
- $\Sigma^I_B = \Sigma^I_A$ and $\Sigma^O_B = \Sigma^O_A$;
- $\mathcal{C}_B = \mathcal{C}_A \cup \mathcal{C}_{A'}$;
- $I_B(q^i_B) = I_A(q^i_A) \wedge I_{A'}(q^i_{A'})$;
- $\Delta_B = \{\delta^1_B, \ldots, \delta^k_B\}$, where δ^i is of the form
  - $(q^i_B, a^i, g^i_B, \rho^i_B, q^{i+1}_B)$, where $g^i_B = g^i_A \wedge g^i_{A'}$, and $\rho^i_B = \rho^i_A \cup \rho^i_{A'}$, for all $i < k$; and
  - $(q^k_B, a^k, g^k_B, \{\}, q^{k+1}_B)$, where $g^k_B = g^k_{A'} \wedge \bigwedge_{i \in I} \neg g^{k,i}_A$, such that $\bigwedge_{i \in I} \neg g^{k,i}_A$ negates guards $g^{k,i}_A$ in all outgoing transitions from q^k_A labeled by a^k;
- $F_B = \{q^{k+1}_B\}$.

We illustrate $B = \text{witness}^{\#}(\sigma, A, A')$ in Figure 1. In this example, $\sigma = 1 \cdot a \cdot 3 \cdot b$ is a trace that is included in the language of A' but not in the language of A. This trace is extrapolated to the TAIO B, which accepts a set of traces, including σ, which are all included in the language of A', but not in the language of A. This example highlights the need for having only the last location in B being accepting. If all the locations were accepting, then B would also accept prefixes of witness traces, such as $\sigma' = 1 \cdot a$, which are included in both languages of A and A', thus violating the desired property of the witness extrapolation procedure.

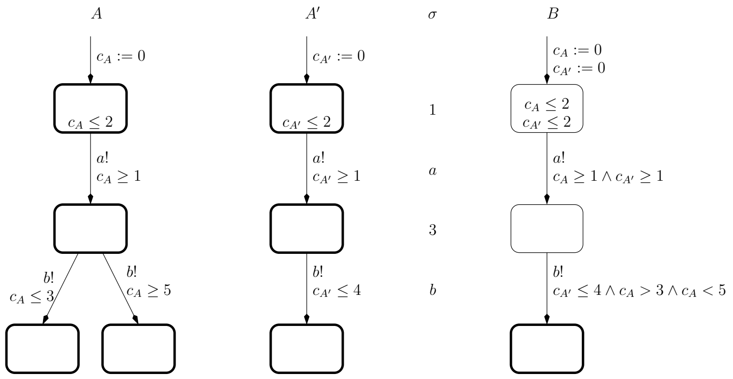


Fig. 1. Witness extrapolation example

TAIO Difference. We instantiate the operation $B \leftarrow \textsc{Diff}(A, A')$ such that $L(B) = L(A) \backslash L(A')$, by using the standard construction $B = A \times \bar{A}'$, where $\bar{A}'$ is the complement of A'. The complement of A' can be computed because we require A' to be deterministic, and it consists of completing A' followed by inversing the set of accepting locations in the completed TAIO [3].

Algorithm. $\textsc{BndIncLI}(A, A', B, k, k')$ takes in addition two bounds k and k', and consists of replacing abstract operations in $\textsc{IncLI}(A, A', B)$ by the concrete ones as follows: (1) $\text{unsat}(\textsc{BndLI}^{k}_{A',A})$ replaces $\text{LI}(A', A)$; (2) $\text{witness}^{\#}(\sigma, A, A')$ replaces $T(A, A', \sigma)$; (3) $\text{unsat}(\textsc{BndEmp}^{k+k'}_{A''\|B})$ replaces $\textsc{Emp}(A'' \| B)$; (4) $A' \times \bar{A}''$ replaces $\textsc{Diff}(A', A'')$.

Given three TAIOs A, A' and B such that A and A' are defined over the same alphabet and B is composable with both A and A' and $(b, \sigma) = \textsc{BndIncLI}(A, A', B, k, k')$, we have that if $b = \mathit{false}$, then $L(A' \| B) \not\subseteq L(A \| B)$, $\sigma \in L(A' \| B)$ and $\sigma \notin L(A \| B)$.

5 Case Study and Experimental Results

We have implemented the bounded language inclusion checking procedure presented in Section 4. In our implementation we rely on Uppaal [12] to model the timed automata components and Z3 [9] to solve SMT formulas. We use a real-time variant of the Dining Philosophers (DP) problem as our case study.

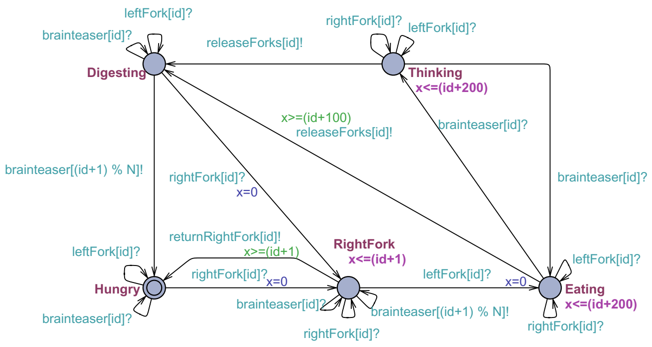


Fig. 2. Real-time dining philosopher P_{id}: N denotes the total number of philosophers

The DP problem consists of n interacting philosophers sitting around a table that has a single fork per guest. In order to eat, each philosopher needs to get hold of two forks. This setting likely leads to a deadlock situation: each philosopher picks the fork to his right, resulting in all forks being in the possession of one philosopher, and then waits forever for the fork to his left to become available. We depict our real-time variant of the DP problem in Figure 2. To avoid the deadlock situation, each DP is required to release the right fork after a time-out period of waiting for the fork to his left. We extend the classical DP problem with additional behavior: philosophers exchange brainteasers with their immediate neighbors. Our model of a philosopher is deterministic and receptive. The latter explains the number of self-loops in the model as philosophers ignore brainteasers in locations Hungry, RightFork and Digesting, any presented fork in Eating and Thinking, the right fork in the RightFork and the left fork in the LeftFork locations.

For the experimental evaluation we consider a network $P = \|_{i=1}^{n} P_i$ of n philosophers. We conduct several experiments in which we alter the behavior of the last philosopher P_n, resulting in the modified network $M = \|_{i=1}^{n-1} P_i \parallel M_n$. We use our incremental procedure (Incremental) and the one implemented in [2] (Classic) to find a trace allowed in M but not in P. Classic first computes the parallel compositions P and M explicitly before checking the trace containment of M in P. Tests were done on a machine with 8GB RAM, SSD, an Intel i5-2520M @ 2.50GHz running in a full power setting, Windows 7 and Z3 4.3 64 bit.

First Experiment. We changed P_n into M_n by making the philosopher P_n ignore the brainteaser in location Thinking, resulting in $L(M_n) \nsubseteq L(P_n)$ and the witness Short of size 6. A similar mutation resulted in another witness Long of size 11. We compared the time needed to find the two witnesses by the Incremental and Classic approaches for an increasing number of philosophers, as depicted in Figure 3. In this experiment, the Incremental approach achieved an impressive speedup of several orders of magnitude compared to the Classic one. While Classic could scale up to 3 and 4 philosophers for the Long and Short witnesses respectively, the Incremental procedure could scale up to 10 philosophers with ease. In the scenario with 3 philosophers and the Long witness it took Classic $255494s$ ($\sim 71h$) to find the witness while Incremental found the same witness in $31s$. Similarly, in the scenario with 4 philosophers and the Short witness, the witness was found in $32176s$ ($\sim 9h$) by Classic and in $5s$ by the Incremental procedure. The Incremental procedure was able to achieve this impressive speedup in this experiment because it could directly extend the witness of $L(M_n) \nsubseteq L(P_n)$ into the witness of $L(M) \nsubseteq L(P)$ without backtracking.

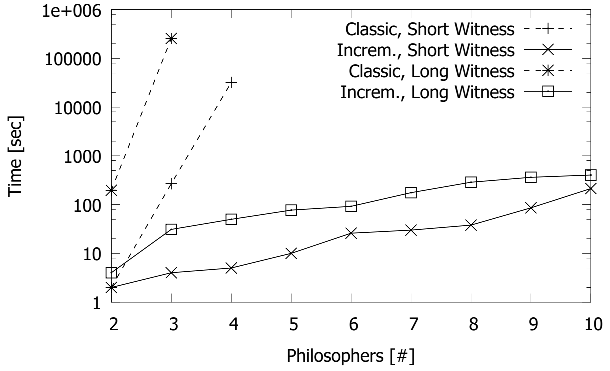


Fig. 3. First experiment: run time for Incremental and Classic

Second Experiment. We altered P_n into a model M_n, such that M_n contains two types of traces witnessing $L(M_n) \nsubseteq L(P_n)$: (1) "spurious" traces which could not be extended to witnesses of $L(M) \nsubseteq L(P)$; and (2) traces that could be extended to witnesses of $L(M) \nsubseteq L(P)$. The Incremental procedure was able to find the witness of size 5 of $L(M) \nsubseteq L(P)$ in $178s$ after a single backtracking operation in the scenario with 10 philosophers. This is better than the $214s$

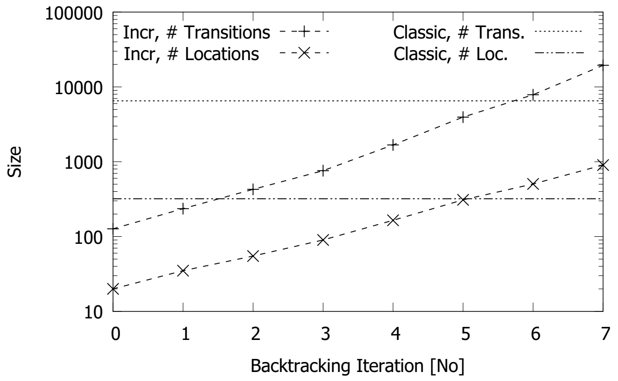


Fig. 4. Size of $M_n \times \bar{P}_n$ over multiple backtrackings compared to $M \times \bar{P}$

Incremental needed to come up with the witness of size 6 in our first experiment in the same setting. Hence this experiment hints that a single backtracking in the Incremental procedure might not induce a lot of overhead.

Third Experiment. In this final setting, we explored the effect of multiple backtrackings in the Incremental procedure. We altered P_n into M_n, in a way that $L(M_n) \not\subseteq L(P_n)$, but $L(M) \subseteq L(P)$. It follows that Incremental was finding "local" witnesses of $L(M_n) \not\subseteq L(P_n)$ that could not be extended to a witness of $L(M) \not\subseteq L(P)$, resulting in an increasing number of backtrackings in which these spurious witnesses were removed from M_n. Each backtracking consists in computing the product of M_n with the negation of the extrapolated witness, resulting in an exponential growth of the size of M_n in the number of iterations. This means that after each backtracking, checking $L(M_n) \subseteq L(P_n)$ becomes computationally more expensive: the efficiency of Incremental is highly sensitive to the number of backtracking iterations needed before finding a witness of trace non containment. In our example we could backtrack five times before the local inclusion check reached a complexity comparable to that of Classic (c.f. Figure 4).

6 Application to Mutation-Based Test Case Generation

This work was mainly motivated by its application to model-based mutation testing [1,14,7,10]. It is a specific type of model-based testing [16], in which faults are deliberately injected into the specification model. The aim of mutation-based testing techniques is to generate test cases that can detect the injected errors. This means that a generated test case shall fail if it is executed on a (deterministic) system-under-test (SUT) that implements the faulty model.

We developed a framework for real-time mutation-based test case generation (TCG) in [2]. The overview of the framework is illustrated in Figure 5. Given a specification S of the SUT, expressed as a deterministic TAIO, S is altered using predefined mutation operators, resulting in a set of mutants. In order to generate a test case from the high-level specification S and its mutant M, the mutant is

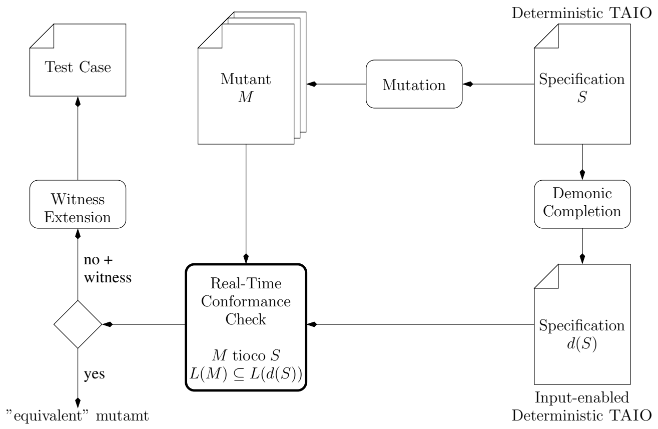


Fig. 5. Mutation-based Test Case Generation Framework for TAIOs - Overview

checked for tioco conformance [11] to the original specification. If M tioco S, we say that M is an equivalent mutant to S, and no test case is generated. Otherwise, a witness of non-conformance is obtained, and it is used as a skeleton to build the actual test case. Checking tioco directly is not easy, and the problem is effectively solved in two steps. In the first step, the original non-receptive specification S is completed using a procedure called *demonic completion*, resulting in another TAIO $d(S)$, which preserves all essential properties of S with respect to the tioco relation. In fact, one can show that the problem of conformance checking M tioco S can be reduced to a simpler problem of language inclusion checking $L(M) \subseteq L(d(S))$. In the second step, $L(M) \subseteq L(d(S))$ is actually checked. The framework in [2] proposes an algorithm for k-bounded language inclusion checking between two *flat* TAIOs based on bounded model checking techniques.

Given a specification S consisting of a network $S_1 \parallel S_2 \parallel \ldots \parallel S_n$, its mutant M is of the form $M_1 \parallel S_2 \parallel \ldots \parallel S_n$, since M has by definition a single altered feature. The TCG procedure from [2] first needs to flatten S and M by computing the parallel composition before checking language inclusion. We instead propose to replace the language inclusion check from [2] with its incremental variant from Section 3. As a consequence, mutation-based TCG represents a natural application domain for incremental language inclusion checking.

7 Conclusion

In this paper, we proposed an incremental language inclusion checking procedure for networks of timed automata and applied it to a real-time variant of the Dining Philosophers problem. The experimental results look promising and indicate that the procedure finds witnesses of language inclusion violation efficiently when the number of backtracking operations is limited to a small number. The approach seems in particular effective in the context of mutation-based test case generation, where a mutant is typically expected to have behaviors not contained in the original specification. These preliminary results suggest that the incremental approach will enable us to generate test cases for larger systems which we could not handle with the classical approach.

In the future, we plan to evaluate our procedure on other examples. In addition, we plan to develop different variants of the witness extrapolation operation, and study the criteria that would guarantee termination of our procedure.

Acknowledgements. We would like to thank Cezara Dragoi, Oded Maler and anonymous reviewers for their comments and suggestions.

References

1. Aichernig, B.K., Brandl, H., Jöbstl, E., Krenn, W.: Uml in action: a two-layered interpretation for testing. ACM SIGSOFT Software Engineering Notes 36(1), 1–8 (2011)
2. Aichernig, B.K., Lorber, F., Ničković, D.: Time for mutants – model-based mutation testing with timed automata. In: Veanes, M., Viganò, L. (eds.) TAP 2013. LNCS, vol. 7942, pp. 20–38. Springer, Heidelberg (2013)
3. Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. 126(2), 183–235 (1994)
4. Alur, R., Itai, A., Kurshan, R.P., Yannakakis, M.: Timing verification by successive approximation. Inf. Comput. 118(1), 142–157 (1995)
5. Audemard, G., Cimatti, A., Kornilowicz, A., Sebastiani, R.: Bounded model checking for timed systems. In: Peled, D.A., Vardi, M.Y. (eds.) FORTE 2002. LNCS, vol. 2529, pp. 243–259. Springer, Heidelberg (2002)
6. Badban, B., Lange, M.: Exact incremental analysis of timed automata with an SMT-solver. In: Fahrenberg, U., Tripakis, S. (eds.) FORMATS 2011. LNCS, vol. 6919, pp. 177–192. Springer, Heidelberg (2011)
7. Brillout, A., He, N., Mazzucchi, M., Kroening, D., Purandare, M., Rümmer, P., Weissenbacher, G.: Mutation-based test case generation for simulink models. In: de Boer, F.S., Bonsangue, M.M., Hallerstede, S., Leuschel, M. (eds.) FMCO 2009. LNCS, vol. 6286, pp. 208–227. Springer, Heidelberg (2010)
8. Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 154–169. Springer, Heidelberg (2000)
9. de Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008)
10. He, N., Rümmer, P., Kroening, D.: Test-case generation for embedded simulink via formal concept analysis. In: DAC, pp. 224–229 (2011)
11. Krichen, M., Tripakis, S.: Conformance testing for real-time systems. Formal Methods in System Design 34(3), 238–304 (2009)
12. Larsen, K.G., Pettersson, P., Yi, W.: Uppaal in a nutshell. STTT 1(1-2), 134–152 (1997)
13. Niebert, P., Mahfoudh, M., Asarin, E., Bozga, M., Maler, O., Jain, N.: Verification of timed automata via satisfiability checking. In: Damm, W., Olderog, E.-R. (eds.) FTRTFT 2002. LNCS, vol. 2469, pp. 225–244. Springer, Heidelberg (2002)

14. Schlick, R., Herzner, W., Jöbstl, E.: Fault-based generation of test cases from UML-models – approach and some experiences. In: Flammini, F., Bologna, S., Vittorini, V. (eds.) SAFECOMP 2011. LNCS, vol. 6894, pp. 270–283. Springer, Heidelberg (2011)
15. Steiner, W.: An evaluation of smt-based schedule synthesis for time-triggered multi-hop networks. In: RTSS, pp. 375–384 (2010)
16. Tretmans, J.: Model based testing with labelled transition systems. In: Hierons, R.M., Bowen, J.P., Harman, M. (eds.) FORTEST. LNCS, vol. 4949, pp. 1–38. Springer, Heidelberg (2008)
17. van der Bijl, M., Rensink, A., Tretmans, J.: Compositional testing with IOCO. In: Petrenko, A., Ulrich, A. (eds.) FATES 2003. LNCS, vol. 2931, pp. 86–100. Springer, Heidelberg (2004)

Nested Timed Automata

Guoqiang Li[1], Xiaojuan Cai[1], Mizuhito Ogawa[2], and Shoji Yuen[3]

[1] School of Software, Shanghai Jiao Tong University, China
{li.g,cxj}@sjtu.edu.cn
[2] Japan Advanced Institute of Science and Technology, Japan
mizuhito@jaist.ac.jp
[3] Graduate School of Information Science, Nagoya University, Japan
yuen@is.nagoya-u.ac.jp

Abstract. This paper proposes a new timed model named *nested timed automata (NeTAs)*. A NeTA is a pushdown system whose stack symbols are *timed automata (TAs)*. It either behaves as the top TA in the stack, or switches from one TA to another by pushing, popping, or changing the top TA of the stack. Different from existing component-based context-switch models such as *recursive timed automata* and *timed recursive state machines*, when time passage happens, all clocks of TAs in the stack elapse uniformly. We show that the safety property of NeTAs is decidable by encoding NeTAs to the *dense timed pushdown automata*. NeTAs provide a natural way to analyze the recursive behaviors of component-based timed systems with structure retained. We illustrate this advantage by the deadline analysis of nested interrupts.

1 Introduction

Due to the rapid development of large and complex timed systems, requirements to model and analyze complex real-time frameworks with recursive context switches have been stresses. Difficulty comes from two dimensions of infinity, a stack with unbounded number of symbols, and clocks recording dense time.

Timed automata (TAs) [1] are a finite automaton with a finite set of *clocks* that grow uniformly. A typical timed model with context switches is *timed pushdown automata (TPDAs)* [2], equipped with an unbounded stack, where clocks are not updated in the stack. This limitation is found unnatural in analyzing the timed behavior of programs since clock values should be updated in suspension. Recently, a new timed pushdown model, *dense timed pushdown automata (DTPDAs)* [3] has been proposed, where each symbol in the stack is equipped with local clocks named "ages", and all ages in the stack are updated uniformly for time passage. Reachability problem of DTPDAs is in EXPTIME [3].

This paper proposes a new timed model named *nested timed automata (NeTAs)*. A NeTA is a pushdown system whose stack symbols are TAs. It either behaves as the top TA in the stack, or switches from one TA to another following three kinds of transitions: pushing a new TA, popping the current TA when terminates, or replacing the top TA of the stack. This hierarchical design captures the dynamic feature of functionally independent component-based structure of timed systems. The existing models, such as *recursive timed automata* [4], and *timed recursive state machines* [5] do not update clocks in the stack when time passage happens, while in NeTAs, all clocks elapse uniformly. When a TA is pushed into the stack, a set of fresh local clocks is introduced to the system. In this respect, NeTAs may have to handle an unbounded number of local clocks. NeTAs are shown to be encoded to DTPDAs preserving the state reachability. All transitions of NeTAs are simulated by DTPDAs, and vice versa. We illustrate that NeTAs are adopted to analyze the timely deadline of nested interrupts.

The rest of the paper is organized as follows. Section 2 gives an introduction of TAs and DTPDAs. Section 3 gives the formal definition and semantics of NeTAs. Section 4 presents an encoding method from NeTAs to DTPDAs, and proves its correctness. Section 5 illustrates the usages of NeTAs by an application example. Section 6 gives the related work and Section 7 concludes the paper.

Due to the lack of space, we omit proofs of theorems, detailed explanations and nations, which can be found in its extended version [6].

2 Preliminaries

Let $\mathbb{R}^{\geq 0}$ and $\mathbb{N}$ denote the sets of non-negative real numbers and natural numbers, respectively. We define $\mathbb{N}^{\omega} := \mathbb{N} \cup \{\omega\}$, where ω is the first limit ordinal. Let $\mathcal{I}$ denote the set of *intervals*. An interval is a set of numbers, written as (a, b), $[a, b]$, $[a, b)$ or $(a, b]$, where $a \in \mathbb{N}$ and $b \in \mathbb{N}^{\omega}$. For a number $r \in \mathbb{R}^{\geq 0}$ and an interval $I \in \mathcal{I}$, we use $r \in I$ to denote that r belongs to I.

Let $X = \{x_1, \dots, x_n\}$ be a finite set of *clocks*. A *clock valuation* $\nu : X \rightarrow \mathbb{R}^{\geq 0}$, assigns a value to each clock $x \in X$. $\nu_{\mathbf{0}}$ represents all clocks in X assigned to zero. Given a clock valuation ν and a time $t \in \mathbb{R}^{\geq 0}$, $(\nu + t)(x) = \nu(x) + t$, for $x \in X$. A clock assignment function $\nu[y \leftarrow b]$ is defined by $\nu[y \leftarrow b](x) = b$ if $x = y$, and $\nu(x)$ otherwise.

2.1 Timed Automata

A timed automaton is an automaton augmented with a finite set of clocks [1,7]. Time can elapse in a location, while switches are instantaneous.

Since we focus on the *safety properties* (i.e., *emptiness* problem of a TA, or *reachability* problem of a timed transition system), we omit input symbols for all concerned automata, following the formalization in [3].

We adopt the TA definition style from that in [3], which looks different from the one in [1,7]. The main difference is that we do not adopt *invariant*, a time constraint assigned to each control location. The reason lies that invariants cause time lock problems. When context switches back, it may occur that the system can neither stay in the current control location since the invariant is violated nor transit to other control location since all constraints on transitions are violated. Nondeterministic clock updates are also taken from [9] with interval tests as diagonal free time constraints where decidability results are not affected.

Definition 1 (Timed Automata). *A timed automaton is a tuple $\mathcal{A} = (Q, q_0, F, X, \Delta) \in \mathscr{A}$, where*

- *Q is a finite set of control locations, with the initial location $q_0 \in Q$,*
- *$F \subseteq Q$ is the set of final locations,*
- *X is a finite set of clocks,*
- *$\Delta \subseteq Q \times \mathcal{O} \times Q$, where $\mathcal{O}$ is a set of* operations. *A transition $\delta \in \Delta$ is a triplet (q_1, ϕ, q_2), written as $q_1 \xrightarrow{\phi} q_2$, in which ϕ is either of*
  Local *ϵ, an* empty *operation,*
  Test *$x \in I?$ where $x \in X$ is a clock and $I \in \mathcal{I}$ is an interval, and*
  Assignment *$x \leftarrow I$ where $x \in X$ and $I \in \mathcal{I}$.*

Given a TA $\mathcal{A} \in \mathscr{A}$, we use $Q(\mathcal{A})$, $q_0(\mathcal{A})$, $F(\mathcal{A})$, $X(\mathcal{A})$ and $\Delta(\mathcal{A})$ to represent its set of control locations, initial location, set of final locations, set of clocks and set of transitions, respectively. We will use similar notations for other models.

Definition 2 (Semantics of TAs). *Given a TA $\mathcal{A} = (Q, q_0, F, X, \Delta)$, a configuration is a pair (q, ν) of a control location $q \in Q$, and a clock valuation ν on X. The transition relation of the TA is represented as follows,*

- Progress transition*: $(q, \nu) \xrightarrow{t}_{\mathscr{A}} (q, \nu + t)$, where $t \in \mathbb{R}^{\geq 0}$.*
- Discrete transition*: $(q_1, \nu_1) \xrightarrow{\phi}_{\mathscr{A}} (q_2, \nu_2)$, if $q_1 \xrightarrow{\phi} q_2 \in \Delta$, and one of the following holds,*
  - **Local** *$\phi = \epsilon$, then $\nu_1 = \nu_2$.*
  - **Test** *$\phi = x \in I?$, $\nu_1 = \nu_2$ and $\nu_2(x) \in I$ holds.*
  - **Assignment** *$\phi = x \leftarrow I$, $\nu_2 = \nu_1[x \leftarrow r]$ where $r \in I$.*

The initial configuration is $(q_0, \nu_{\mathbf{0}})$. The transition relation is $\rightarrow$ and we define $\rightarrow = \xrightarrow{t}_{\mathscr{A}} \cup \xrightarrow{\phi}_{\mathscr{A}}$, and define $\rightarrow^*$ to be the reflexive and transitive closure of $\rightarrow$.

Remark 1 (Sound Simulation). The TA definition in Definition 1 follows the style in [3]. In [1], a TA allows a logical connection of several constraint tests, e.g. $x \leq 15 \wedge y > 20$, and several resets operations of different clocks during one discrete transition. Only one test or assignment (a generalization of the reset) is allowed during one discrete transition in the definition. Since a discrete transition is followed by a progress transition where time elapses, the main ambiguity of two definitions is whether a conjunction of two tests can be checked one by one, between which the time elapses. It is shown that TA with our definition soundly simulates the timed traces in the original definition, as follows,

For $\geq$ or $>$, $c \xrightarrow{x\in[a,+\infty)?}_{\mathscr{A}} c' \xrightarrow{t}_{\mathscr{A}} c'$ is safely converted to $c \xrightarrow{t}_{\mathscr{A}} c \xrightarrow{[a,+\infty)?}_{\mathscr{A}} c'$, for some configurations c and c'.

For $\leq$ or $<$, $c \xrightarrow{t}_{\mathscr{A}} c \xrightarrow{x\in[0,a]?}_{\mathscr{A}} p'$ is safely converted to $c \xrightarrow{x\in[0,a]?}_{\mathscr{A}} c' \xrightarrow{t}_{\mathscr{A}} c'$, for some configurations c and c'.

For test transitions, a general simulation strategy is, firstly, checking the $\geq$, and $>$ one by one, then check $\leq$ and $<$ later. If there exists a "=" constraint, decomposed it into $\geq \wedge \leq$. For example, a transition $p \xrightarrow{x\leq 15\wedge y>20} q$ in the original definition is simulated by two transitions $p \xrightarrow{y\in(20,+\infty)?} p' \xrightarrow{x\in[0,15]?} q$ under the new definition, where p' is a fresh control location.

For reset transitions, a group of clocks are reset simultaneously can be simulated by resetting clocks one by one, with a zero test of the first reset clock on the tail. For example, a transition $p \xrightarrow{\{x,y\}} q$, resetting x and y simultaneously, in the original definition is simulated by $p \xrightarrow{x \leftarrow [0,0]} p' \xrightarrow{y \leftarrow [0,0]} p'' \xrightarrow{x \in [0,0]?} q$, where p', p'' are fresh control locations.

If a transition contains both test and reset operations, we firstly simulate test operation, then reset operation, following the above rules.

2.2 Dense Timed Pushdown Automata

Dense Timed Pushdown Automata (DTPDAs) [3] extend TPDAs with time update in the stack. Each symbol in the stack is equipped with a local clock named *age*, and all ages in the stack elapse uniformly.

Definition 3 (Dense Timed Pushdown Automata). *A dense timed pushdown automaton is a tuple $\mathcal{D} = \langle S, s_0, \Gamma, C, \Delta \rangle \in \mathscr{D}$, where*

- *S is a finite set of states with the initial state $s_0 \in S$,*
- *Γ is a finite stack alphabet,*
- *C is a finite set of clocks, and*
- *$\Delta \subseteq S \times \mathcal{O} \times S$ is a finite set of transitions.*

A transition $\delta \in \Delta$ is a triplet (s_1, ϕ, s_2), written as $s_1 \xrightarrow{\phi} s_2$, in which ϕ is either of

- **Local** *ϵ, an empty operation,*
- **Test** *$x \in I?$, where $x \in X$ is a clock and $I \in \mathcal{I}$ is an interval,*
- **Assignment** *$x \leftarrow I$ where $x \in C$ and $I \in \mathcal{I}$,*
- **Push** *$push(\gamma, I)$, where $\gamma \in \Gamma$ is a stack symbol and $I \in \mathcal{I}$. It pushes γ to the top of the stack, with the age in the interval I.*
- **Pop** *$pop(\gamma, I)$, where $\gamma \in \Gamma$ and $I \in \mathcal{I}$. It pops the top-most stack symbol provided that this symbol is γ, and its age belongs to I.*
- **Push$_A$** *$push(\gamma, x)$, where $\gamma \in \Gamma$ is a stack symbol and $x \in C$, and*
- **Pop$_A$** *$pop(\gamma, x)$, where $\gamma \in \Gamma$ is a stack symbol and $x \in C$.*

Definition 4 (Semantics of DTPDAs). *For a DTPDA $\langle S, s_0, \Gamma, C, \Delta \rangle$, a configuration is a triplet (s, w, ν) with $s \in S$, $w \in (\Gamma \times \mathbb{R}^{\geq 0})^*$, and a clock valuation ν on X. Time passage of the stack $w + t = (\gamma_1, t_1 + t). \cdots .(\gamma_n, t_n + t)$ for $w = (\gamma_1, t_1). \cdots .(\gamma_n, t_n)$.*

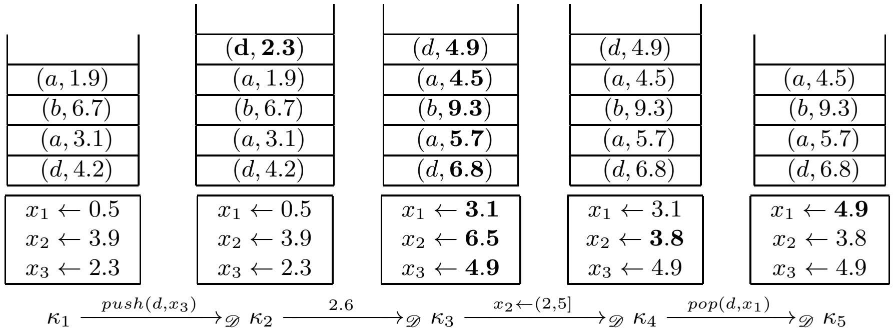
The transition relation of the DTPDA is defined as follows:

- Progress transition*: $(s, w, \nu) \xrightarrow{t}_{\mathscr{D}} (s, w + t, \nu + t)$, where $t \in \mathbb{R}^{\geq 0}$.*
- Discrete transition*: $(s_1, w_1, \nu_1) \xrightarrow{\phi}_{\mathscr{D}} (s_2, w_2, \nu_2)$, if $s_1 \xrightarrow{\phi} s_2$, and one of the following holds,*
  - **Local** *$\phi = \epsilon$, then $w_1 = w_2$, and $\nu_1 = \nu_2$.*
  - **Test** *$\phi = x \in I?$, then $w_1 = w_2$, $\nu_1 = \nu_2$ and $\nu_2(x) \in I$ holds.*
  - **Assignment** *$\phi = x \leftarrow I$, then $w_1 = w_2$, $\nu_2 = \nu_1[x \leftarrow r]$ where $r \in I$.*
  - **Push** *$\phi = push(\gamma, I)$, then $\nu_1 = \nu_2$, and $w_2 = (\gamma, r).w_1$ for some $r \in I$.*
  - **Pop** *$\phi = pop(\gamma, I)$, then $\nu_1 = \nu_2$, and $w_1 = (\gamma, r).w_2$ for some $r \in I$.*

- **Push**$_A$ $\phi = push(\gamma, x)$, *then* $\nu_1 = \nu_2$, *and* $w_2 = (\gamma, \nu_1(x))w_1$.
- **Pop**$_A$ $\phi = pop(\gamma, x)$, *then* $\nu_2 = \nu_1[x \leftarrow t]$, *and* $w_1 = (\gamma, t)w_2$.

The initial configuration $\kappa_0 = (q_0, \epsilon, \nu_{\mathbf{0}})$. *We use* $\hookrightarrow$ *to range over these transitions, and* $\hookrightarrow^*$ *is the transitive closure of* $\hookrightarrow$, *conventionally.*

Example 1. Fig. 1 shows transitions between configurations of a DTPDA consisting of a singleton state set $S = \{\bullet\}$ (omitted in the figure), clocks $C = \{x_1, x_2, x_3\}$, and stack symbols $\Gamma = \{a, b, d\}$. From κ_1 to κ_2, a discrete transition $push(d, x_3)$ pushes $(d, 2.3)$ into the stack. A time transition from κ_2 to κ_3 elapses 2.6 time units, and each value grows older for 2.6. From κ_3 to κ_4, the value of x_2 is reset to 3.8, which lies in the interval $(2, 5]$, and the last transition pops (d, x_1) and resets x_1 to 4.9.



$(a, 1.9)$
$(b, 6.7)$
$(a, 3.1)$
$(d, 4.2)$
$x_1 \leftarrow 0.5$
$x_2 \leftarrow 3.9$
$x_3 \leftarrow 2.3$
$(\mathbf{d}, \mathbf{2.3})$
$(d, \mathbf{4.9})$
$(a, \mathbf{4.5})$
$(b, \mathbf{9.3})$
$(a, \mathbf{5.7})$
$(d, \mathbf{6.8})$
$x_1 \leftarrow \mathbf{3.1}$
$x_2 \leftarrow \mathbf{6.5}$
$x_3 \leftarrow \mathbf{4.9}$
$x_2 \leftarrow \mathbf{3.8}$
$x_1 \leftarrow \mathbf{4.9}$
$\kappa_1 \xrightarrow{push(d,x_3)}_{\mathscr{D}} \kappa_2 \xrightarrow{2.6}_{\mathscr{D}} \kappa_3 \xrightarrow{x_2 \leftarrow (2,5]}_{\mathscr{D}} \kappa_4 \xrightarrow{pop(d,x_1)}_{\mathscr{D}} \kappa_5$

Fig. 1. An Example of DTPDA

Remark 2. Definition 3 extends the definition in [3] by adding **Push**$_A$ and **Pop**$_A$, which stores and recovers from clocks to ages and vice versa. This extension does not destroy decidability of state reachability of DTPDAs [8], since its symbolic encoding is easily modified to accept **Push**$_A$ and **Pop**$_A$. For instance, **Push**$_A$ is encoded similar to **Push**, except for the definition on *Reset* [3]. *Reset*$(R)[a \leftarrow I]$ symbolically explores all possibility of the fraction of an instance in I. Instead, *Reset*$(R)[a \leftarrow x]$ will assign the same integer and fraction parts to x, which means an age is simply placed at the same position to x in the region.

3 Nested Timed Automata

Nested Timed Automata (NeTAs) aim to give an operation strategy to a group of TAs, in which a TA is able to preempt the other ones. All clocks in a NeTA are local clocks, with the scope of their respective TAs. These clocks in the stack elapse simultaneously. An unbounded number of clocks may be involved in one NeTA, due to recursive preemption loops.

Definition 5 (Nested Timed automata). *A nested timed automaton is a triplet* $\mathcal{N} = (T, \mathcal{A}_0, \Delta) \in \mathscr{N}$, *where*

- *T is a finite set of timed automata, with the initial timed automaton $\mathcal{A}_0 \in T$.*
- *$\varDelta \subseteq T \times \mathcal{P} \times (T \cup \{\varepsilon\})$, where $\mathcal{P} = \{push, pop, internal\}$. A rule $(\mathcal{A}_i, \varPhi, \mathcal{A}_j) \in \varDelta$ is written as $\mathcal{A}_i \xrightarrow{\varPhi} \mathcal{A}_j$, where*

  Push $\mathcal{A}_i \xrightarrow{push} \mathcal{A}_j$,

  Pop $\mathcal{A}_i \xrightarrow{pop} \varepsilon$, *and*

  Internal $\mathcal{A}_i \xrightarrow{internal} \mathcal{A}_j$.

The initial state of NeTAs is the initial location in $\mathcal{A}_0$, s.t. $q_0(\mathcal{A}_0)$. We also assume that $X(\mathcal{A}_i) \cap X(\mathcal{A}_j) = \emptyset$, and $Q(\mathcal{A}_i) \cap Q(\mathcal{A}_j) = \emptyset$ for $\mathcal{A}_i, \mathcal{A}_j \in T$ and $i \neq j$.

The operational semantics of NeTAs is informally summarized as follows. It starts with a stack with the only symbol of the initial TA. The system has the following four behaviors: when there exists time passage, all clocks in the stack elapse simultaneously; it is able to behave like the top TA in the stack; when there exists a push transition from the top TA of the stack to the other TA, a new instance of the latter TA is pushed into the stack at any time and executed, while the suspended location of the former TA is recorded in the stack; when the top TA in the stack reaches the final location and a pop transition happens, it will be popped from the stack, and the system will run the next TA beginning with the suspended location; if an internal transition from the top TA to the other TA occurs, the top TA in the stack will be changed to a new instance of the latter TA when it reaches some final location.

Definition 6 (Semantics of NeTAs). *Given a NeTA $(T, \mathcal{A}_0, \varDelta)$, a configuration is a stack, and the stack alphabet is a tuple $\langle \mathcal{A}, q, \nu \rangle$, where $\mathcal{A} \in T$ is a timed automaton, q is the current running control location where $q \in Q(\mathcal{A})$, and ν is the clock valuation of $X(\mathcal{A})$. For a stack content $c = \langle \mathcal{A}_1, q_1, \nu_1 \rangle \langle \mathcal{A}_2, q_2, \nu_2 \rangle \ldots \langle \mathcal{A}_n, q_n, \nu_n \rangle$, let $c + t$ be $\langle \mathcal{A}_1, q_1, \nu_1 + t \rangle \langle \mathcal{A}_2, q_2, \nu_2 + t \rangle \ldots \langle \mathcal{A}_n, q_n, \nu_n + t \rangle$.*

The transition of NeTAs is represented as follows:

- *Progress transitions: $c \xrightarrow{t}_{\mathscr{N}} c + t$.*
- *Discrete transitions: $c \xrightarrow{\phi}_{\mathscr{N}} c'$ is defined as a union of the following four kinds of transition relations,*
  - **Intra-action** *$\langle \mathcal{A}, q, \nu \rangle c \xrightarrow{\phi}_{\mathscr{N}} \langle \mathcal{A}, q', \nu' \rangle c$, if $q \xrightarrow{\phi} q' \in \varDelta(\mathcal{A})$, and one of the following holds,*
    - ∗ **Local** *$\phi = \epsilon$, then $\nu = \nu'$.*
    - ∗ **Test** *$\phi = x \in I?$, $\nu = \nu'$ and $\nu'(x) \in I$ holds.*
    - ∗ **Assignment** *$\phi = x \leftarrow I$, $\nu' = \nu[x \leftarrow r]$ where $r \in I$.*
  - **Push** *$\langle \mathcal{A}, q, \nu \rangle c \xrightarrow{push}_{\mathscr{N}} \langle \mathcal{A}', q_0(\mathcal{A}'), \nu'_{\mathbf{0}} \rangle \langle \mathcal{A}, q, \nu \rangle c$, if $\mathcal{A} \xrightarrow{push} \mathcal{A}'$, and $q \in Q(\mathcal{A})$.*
  - **Pop** *$\langle \mathcal{A}, q, \nu \rangle c \xrightarrow{pop}_{\mathscr{N}} c$, if $\mathcal{A} \xrightarrow{pop} \varepsilon$, and $q \in F(\mathcal{A})$.*
  - **Inter-action** *$\langle \mathcal{A}, q, \nu \rangle c \xrightarrow{internal}_{\mathscr{N}} \langle \mathcal{A}', q_0(\mathcal{A}'), \nu'_{\mathbf{0}} \rangle c$, if $\mathcal{A} \xrightarrow{internal} \mathcal{A}'$, and $q \in F(\mathcal{A})$.*

The initial configuration $c_0 = \langle \mathcal{A}_0, q_0(\mathcal{A}_0), \nu_{\mathbf{0}} \rangle$. We use $\longrightarrow$ to range over these transitions and $\longrightarrow^*$ is the transitive closure of $\longrightarrow$, conventionally.

In followings, we focus on the state reachability that is regarded as the most important property in modelling software behavior.

Definition 7 (Safety Property). *A safety property of NeTAs is defined as the state reachability problem: Given a NeTA $\mathcal{N} = (T, \mathcal{A}_0, \Delta)$, and a control location $p_f \in Q(\mathcal{A})$ for some $\mathcal{A} \in T$, decide whether there exists a configuration c of $\mathcal{N}$ and a clock valuation ν, such that $c_0 \longrightarrow^* \langle \mathcal{A}, p_f, \nu \rangle c$.*

Example 2. We take a simple example to show the usage of NeTAs. Assume that two processes access a shared buffer. One is to read from the buffer periodically each 4 time units. It accomplishes after it reads one or more data. The other is to write to the buffer periodically. The execution time is between 3 and 5 time units. It will return after writes one or more data. The writing process may overtake the reading process which initially starts running. The NeTA is shown in Fig. 2, with three TAs. $\mathcal{A}_0$ is an empty TA for the idle state. $\mathcal{A}_1$ and $\mathcal{A}_2$ are for reading and writing processes, respectively. We have three transition rules: $\mathcal{A}_0 \xrightarrow{internal} \mathcal{A}_1$, $\mathcal{A}_1 \xrightarrow{push} \mathcal{A}_2$, and $\mathcal{A}_2 \xrightarrow{pop} \varepsilon$. The pop transition is not explicitly represented in the figure. We use dash-line frames to represent the border of TAs in the NeTA, double-line arrows to indicate the initial location/TA, and double-line circles to represent the final locations of TAs.

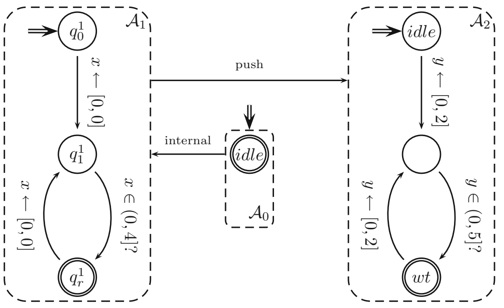


Fig. 2. An Example of NeTA

Remark 3 (Composition with timed automata). A NeTA is composed with a TA by synchronization with shared actions in Σ, where we are allowed to add input symbols as actions on transitions of NeTA. A composed TA presents behavioral properties independent of recursive context switches such as the environment. Although this extension does not increase the expressiveness of NeTAs, it is very useful to model and analyze the behavioral properties in the component-based manner [10,11]. A formal definition of the parallel composition, between a NeTA $\mathcal{N}$ and a TA $\mathcal{A}$, written as $\mathcal{N} \| \mathcal{A}$, is formally defined in [6].

4 Decidability of Safety Property

In this section we prove the *safety property* problem of NeTAs is decidable by encoding it into DTPDAs, of which state reachability is decidable.

4.1 Encoding NeTA to DTPDA

The idea to encode a NeTA to a DTPDA is straightforward, dealing with multiple clocks at push and pop operations. We adopt extra fresh locations and transitions to check whether a group of push/pop actions happens instantly.

Given a NeTA $\mathcal{N} = (T, \mathcal{A}_0, \Delta)$, we define $\mathcal{E}(\mathcal{N}) = \langle S, s_0, \Gamma, C, \nabla \rangle$ as the target of DTPDA encoding of $\mathcal{N}$. Each element is described as,

The set of states $S = S_{\mathcal{N}} \cup S_{inter}$, where $S_{\mathcal{N}} = \bigcup_{\mathcal{A}_i \in T} Q(\mathcal{A}_i)$ is the set of all locations of TAs in $T(\mathcal{N})$. S_{inter} is the set of *intermediate* states during the simulation of push, pop, and internal rules of NeTAs. We assume that $S_{\mathcal{N}}$ and S_{inter} are disjoint.

Let $n = |T|$ and $m_i = |X(\mathcal{A}_i)|$ for each $\mathcal{A}_i \in T$. S_{inter} is

$$S_{inter} = (\bigcup_{\mathcal{A}_i \in T} S^i_{reset}) \cup (\bigcup_{\mathcal{A}_i \xrightarrow{push} \mathcal{A}_j \in \Delta} S^{i,j}_{push}) \cup \{o\}$$

- For every $\mathcal{A}_i \in T$, we define $S^i_{reset} = (\bigcup_{k \in \{1 \cdots m_i+1\}} r^i_k) \cup t^i$. $r^i_k \in S^i_{reset}$ is the start state of a transition to initialize the k-th clock of $\mathcal{A}_i$ to 0. t^i is the start state of a testing transition to make sure that no time is elapsed during the sequence of initializing transitions.
- For every push rule $\mathcal{A}_i \xrightarrow{push} \mathcal{A}_j$, we define $S^{i,j}_{push} = \bigcup_{k \in \{1 \cdots m_i+1\}} p^{i,j}_k$. $p^{i,j}_k$ is the start state of a push transition that push the pair of the k-th clock of $\mathcal{A}_i$ and its value. After all clock values are stored in the stack, the last destination is the initial state $q_0(\mathcal{A}_j)$ of $\mathcal{A}_j$.
- o is a special state for repeat popping.

The initial state $s_0 = q_0(\mathcal{A}_0)$ is the initial location of the initial TA of $\mathcal{N}$.

The set of clocks $C = \{d\} \cup \bigcup_{\mathcal{A} \in T} X(\mathcal{A})$ consists of all clocks of TA in $T(\mathcal{N})$ and the special dummy clock d only to fulfill the field of push and pop rules, like $push(q, d)$ and $pop(q, d)$. (The value of d does not matter.)

The stack alphabet $\Gamma = C \cup S_{\mathcal{N}}$.

The set of transitions ∇ is the union of $\bigcup_{\mathcal{A}_i \in T} \Delta(\mathcal{A}_i)$ (as **Local** transitions of $\mathcal{E}(\mathcal{N})$) and the set of transitions described in Fig. 3. For indexes, we assume $0 \le i, j \le n-1$ and $1 \le k \le m_i$ (where i is specified in a context).

Local	$p^{i,j}_{m_i+1} \xrightarrow{\epsilon} r^j_1,\ r^i_{m_i+1} \xrightarrow{\epsilon} t^i,\ q_i \xrightarrow{\epsilon} r^j_1,\ q_i \xrightarrow{\epsilon} o$ for $q_i \in F(\mathcal{A}_i)$.
Test	$t^i \xrightarrow{x^i_1 \in [0,0]?} q_0(\mathcal{A}_i)$.
Assignment	$r^i_k \xrightarrow{x^i_k \leftarrow [0,0]} r^i_{k+1}$.
Push	$q_i \xrightarrow{push(q_i,d)} p^{i,j}_1,\ p^{i,j}_k \xrightarrow{push(x^i_k, x^i_k)} p^{i,j}_{k+1}$ *if* $k \le m_i$, for $q_i \in Q(\mathcal{A}_i)$.
Pop	$o \xrightarrow{pop(x,x)} o,\ o \xrightarrow{pop(q,d)} q$ forall $x \in X(\mathcal{A}_i)$. $q \in Q(\mathcal{A}_i)$.

Fig. 3. Transition Rules ∇ of $\mathcal{E}(\mathscr{C})$

Definition 8. *For a NeTA $\mathcal{N} = (T, \mathcal{A}_0, \Delta)$, its encoding into a DTPDA $\mathcal{E}(\mathcal{N})$ is as follows.*

$\mathcal{A}_i \xrightarrow{push} \mathcal{A}_j$	$q_i \xrightarrow{push(q_i,d)} p_1^{i,j} \xrightarrow{push(x_1^i,x_1^i)} \cdots p_{m_i}^{i,j} \xrightarrow{push(x_{m_i}^i,x_{m_i}^i)} p_{m_i+1}^{i,j} \xrightarrow{\epsilon}$ $r_1^j \xrightarrow{x_1^j \leftarrow [0,0]} r_2^j \cdots r_{m_j+1}^j \xrightarrow{\epsilon} t^j \xrightarrow{x_1^j \in [0,0]?} q_0(\mathcal{A}_j)$
$\mathcal{A}_i \xrightarrow{pop} \epsilon$	$q_i \xrightarrow{\epsilon} o \xrightarrow{pop(x_{m_i+1}^i,x_{m_i+1}^i)} \cdots \xrightarrow{pop(x_1^i,x_1^i)} o \xrightarrow{pop(q,d)} q$
$\mathcal{A}_i \xrightarrow{internal} \mathcal{A}_j$	$q_i \xrightarrow{\epsilon} r_1^j \xrightarrow{x_1^j \leftarrow [0,0]} r_2^j \cdots r_{m_j+1}^j \xrightarrow{\epsilon} t^j \xrightarrow{x_1^j \in [0,0]?} q_0(\mathcal{A}_j)$

For a *push* transition $\mathcal{A}_i \xrightarrow{push} \mathcal{A}_j$, $\mathcal{E}(\mathcal{N})$ simulates, by storing current state of $\mathcal{A}_i$ into the stack, pushing each clock with its current value as an age, and switching to the initial configuration of $\mathcal{A}_j$ (which consists of initializing each clock $x \in X(\mathcal{A}_j)$, testing that no timed transitions interleaved, and move to the initial state $q_0(\mathcal{A}_j)$).

For a *pop* transition $\mathcal{A}_i \xrightarrow{pop} \epsilon$, $\mathcal{A}_i$ has finished its run at a final state and restores the previous context. $\mathcal{E}(\mathcal{N})$ simulates, first popping and setting each clock (of $\mathcal{E}(\mathcal{N})$), and set a state to q being stored in the stack.

Note that clocks of $\mathcal{E}(\mathcal{N})$ are used for currently running TA (at the top of the stack), and ages are used to store values of clocks of suspended TAs.

Example 3. A NeTA is shown in Fig. 4, which includes two TAs $\mathcal{A}_1$ and $\mathcal{A}_2$. $p_1, p_2 \in Q(\mathcal{A}_1)$, $x_1, x_2 \in X(\mathcal{A}_1)$, $q_1, q_2 \in Q(\mathcal{A}_2)$, and $y_1, y_2 \in X(\mathcal{A}_2)$, respectively. A push transition from $\mathcal{A}_1$ to $\mathcal{A}_2$ occurs at the location p_2, and the value of x_1 and x_2 are 2.9 and 3.3, respectively. After pushing, y_1 and y_2 are reset to zero, and the system begins with q_1. The encoding DTPDA is shown in Fig. 5. p_2 is firstly pushed into the stack, and afterwards, x_1 and x_2 in $\mathcal{A}_1$ is pushed into the stack one by one, with the initial value of the age as their respective value. Then after y_1 and y_2 in $\mathcal{A}_2$ are reset to 0 through the states r_1^2, r_2^2, and r_3^2, the system moves to q_1 provided the value of y_1 is kept as 0.

Example 4. The NeTA in Fig. 2 is encoded into a DTPDA in Fig. 6.

– The larger circles are the original states from the NeTA, while the smaller ones are intermediate states.

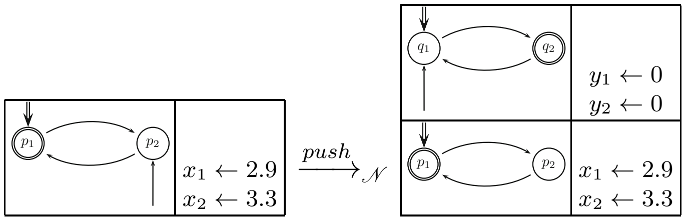


Fig. 4. A Push Transition on a Nested Timed Automaton

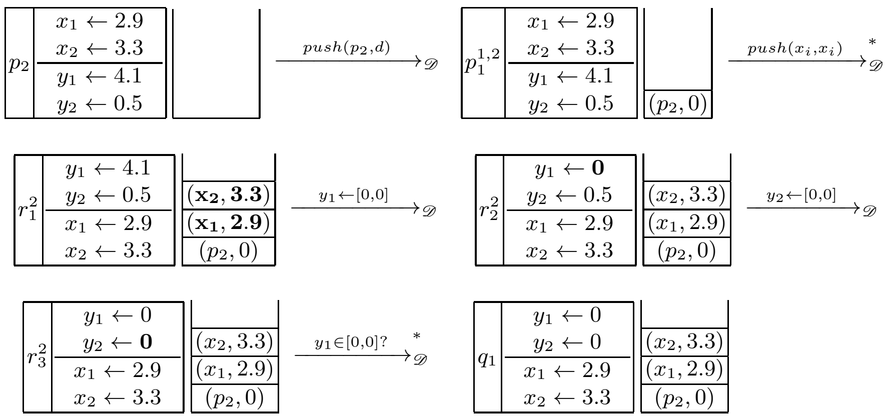


Fig. 5. Encoding the Push Transition in DTPDA

- Since $\mathcal{A}_0 \xrightarrow{internal} \mathcal{A}_1$, before q_0^0 connects to q_0^1, all clocks in $\mathcal{A}_1$ are reset to zero and kept uniformly. q_0^0 firstly is connected to r_1^1. r_1^1 and r_2^1 reset clocks and t^1 tests the uniformity of clocks.
- Since $\mathcal{A}_1 \xrightarrow{push} \mathcal{A}_2$, each state in $\mathcal{A}_1$ connects to $p_1^{1,2}$ by a transition to push itself. $p_1^{1,2}$ and $p_2^{1,2}$ push each clock in $\mathcal{A}_1$. Before connecting to $q_0^2 \in \mathcal{A}_2$, all clocks in $\mathcal{A}_2$ are similarly reset and tested, through r_1^2, r_2^2 and t^2.
- Since $\mathcal{A}_2 \xrightarrow{pop} \varepsilon$, after some final state of $\mathcal{A}_2$ is reached, each clock in the stack should be popped, through an extra state o. After that, o will connect each state in $\mathcal{A}_1$, by which the respective suspended state is popped.

4.2 Correctness of the Encoding

To reduce state reachability problem of NeTAs to that of DTPDAs, we show that transitions are preserved and reflected by the encoding.

Definition 9 (Encoded Configuration). *For a NeTA $\mathcal{N} = (T, \mathcal{A}_0, \Delta)$, its DTPDA encoding $\mathcal{E}(\mathcal{N}) = \langle S, s_0, \Gamma, C, \nabla \rangle$. and a NeTA configuration*

$$c = \langle \mathcal{A}_1, q_1, \nu_1 \rangle \langle \mathcal{A}_2, q_2, \nu_2 \rangle \dots \langle \mathcal{A}_n, q_n, \nu_n \rangle$$

let $c_{hd} = \langle \mathcal{A}_1, q_1, \nu_1 \rangle$ and $c_{tl} = \langle \mathcal{A}_n, q_n, \nu_n \rangle$. A clock valuation of c, $\overline{\nu}(c) : C \to \mathbb{R}^{\geq 0}$ is defined as $\overline{\nu}(c)(x) = \nu_1(x)$ if $x \in X(\mathcal{A}_1)$, and `any`*, otherwise.* [1] *Let $\overline{w}(c) = w_1 \cdots w_n$, where $w_i = (x^i_{m_i}, \nu_i(x^i_{m_i})) \cdots (x^i_1, \nu_i(x^i_1))(q_i, 0)$.*

We denote a configuration $(q_1, \overline{w}(c_{tl}), \overline{\nu}(c))$ of $\mathcal{E}(\mathcal{N})$ by $[\![c]\!]$. A configuration κ of DTPDA with some c and $\kappa = [\![c]\!]$ is called an encoded configuration.

[1] `any` means any value, since except for a clock in the top stack frame of a nested timed automaton, its value does not matter.

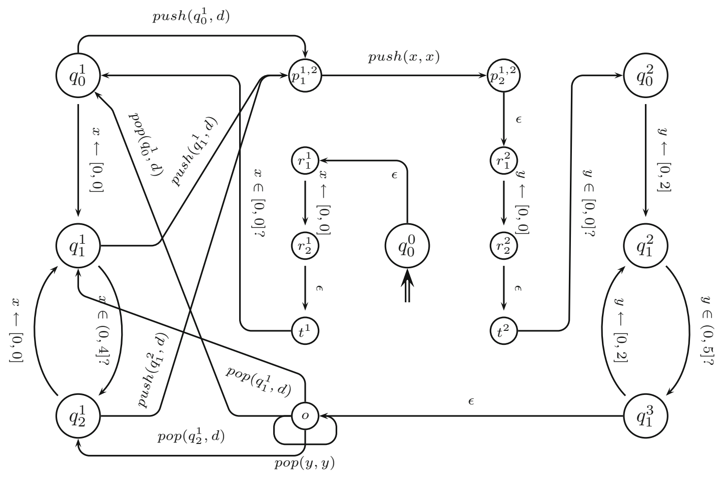


Fig. 6. Encoding the NeTA to DTPDA

Lemma 1. *Given a NeTA $\mathcal{N}$, its encoding $\mathcal{E}(\mathcal{N})$, and configurations c, c' of $\mathcal{N}$.*

- **(Preservation)** *if $c \longrightarrow c'$, then $[\![c]\!] \hookrightarrow^* [\![c']\!]$;*
- **(Reflection)** *if $[\![c]\!] \hookrightarrow^* \kappa$,*
  1. *there exists c' such that $\kappa = [\![c']\!]$ and $c \longrightarrow^* c'$, or*
  2. *κ is not an encoded configuration, and there exists c' such that $\kappa \hookrightarrow^* [\![c']\!]$ by discrete transitions (of $\mathcal{E}(\mathcal{N})$) and $c \longrightarrow^* c'$.*

From Lemma 1, we have the decidability of the safety property of NeTAs.

Theorem 1. *The state reachability problem of NeTAs is decidable.*

Remark 4 (Global clocks). We can assign a disjoint finite set of global clocks to a NeTA. These global clocks are tested and reassigned during push, pop and internal transitions, to control time conditions for push, pop and internal actions. Global clocks do not affect the safety property of a NeTA, since during the encoding to DTPDA, we just include these clocks to the set of clocks in DTPDA, keeping the copies of global clocks for all stack elements.

Fact 1 *A parallel composition of a NeTA and a TA can be encoded into a NeTA with global clocks by forgetting the synchronizing actions.*

Remark 5 (Encode DTPDAs into NeTAs). We can also encode a DTPDA into a NeTA with global clocks by regarding each state of the DTPDA as a TA with only one (local) clock. These TAs and their respective clocks can thus be used to represent pairs of stack symbols and ages.

5 Deadline Analysis for Nested Interrupts

Timely interrupt handling is part of correctness for real-timed, interrupt-driven system. It is vital for a *deadline analysis* [12,13] in such systems to check that all specified deadlines for interrupt handling will be met. Such analysis is a daunting task because of large number of different possible interrupt arrival scenarios. An *interrupt signal* from outside transfers the control to an *interrupt handler* deferring the interrupted execution. When there are more than one interrupts, an *interrupt controller* provides priorities for them according to urgency of each interrupt. An interrupt handler is suspended by another handler with higher priority. After the high priority handler is done, the previous handler is resumed. In the followings NeTA combined with TA is shown to be useful for deadline analysis with such nested interrupt handling.

The time and frequency of interrupt signals can be represented by a TA, with input actions as events that trigger interrupt handlers. For instance, Fig. 7 gives an example of a TA that trigger three interrupt handlers, by $coming_P$, $coming_Q$, and $coming_R$, respectively.

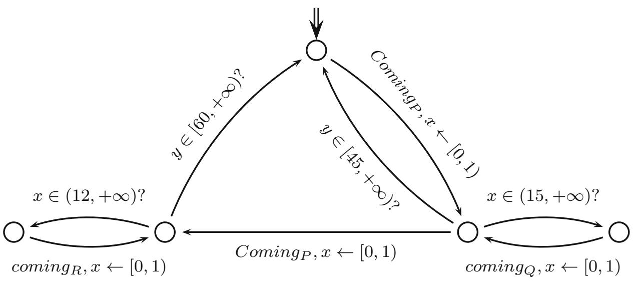

$y \in [60, +\infty)?$
$Coming_P, x \leftarrow [0, 1)$
$y \in [45, +\infty)?$
$x \in (12, +\infty)?$
$x \in (15, +\infty)?$
$coming_R, x \leftarrow [0, 1)$
$Coming_P, x \leftarrow [0, 1)$
$coming_Q, x \leftarrow [0, 1)$

Fig. 7. A Timed Automata as an Environment

Assume a finite set of interrupt handler specifications $\mathscr{H}$. Each handler is specified by $P(\mathcal{A}, D)$, where $\mathcal{A}$ is a TA to describe its behavior, and D is its *relative deadline*. A system should guarantee that each executed handler p of $P(\mathcal{A}, D)$ is executed as $\mathcal{A}$ and reached to some final location of $\mathcal{A}$ within D time units. If the handler misses the deadline, it raises an error.

An interrupt handler with relative deadline D is transformed from $\mathcal{A}$ to another TA with error location. `Guarded` : $\mathscr{H} \to \mathscr{A}$ is defined by `Guarded` $(P(\mathcal{A}, D)) = (Q^G, q_0^G, F^G, X^G, \Delta^G)$. Each element is shown as follows,

- $Q^G = Q(\mathcal{A}) \cup Q_\Delta \cup \{q_{err}\}$, where $Q_\Delta = \{q_\delta \mid \text{for each } \delta \in \Delta(\mathcal{A})\}$.
- $q_0^G = q_0(\mathcal{A})$, and $F^G = F(\mathcal{A})$.
- $X^G = X(\mathcal{A}) \cup \{x_{sch}\}$.
- $\Delta^G = \Delta_{sch} \cup \Delta_{err}$, where
  - $\Delta_{sch} = \{q \xrightarrow{o} q_\delta, q_\delta \xrightarrow{x_{sch} \in [0, D]?} q' \mid \delta = (q, o, q') \in \Delta(\mathcal{A})\}$.
  - $\Delta_{err} = \{q \xrightarrow{x_{sch} \in (D, +\infty)?} q_{err} \mid q \in Q(\mathcal{A}) \cup Q_\Delta\}$.

Given a finite set of handler specifications $\mathscr{H}$, a priority policy is described by a relation $\prec$ on $\mathscr{H}$. For instance, the most common policy is *fixed priority strategy (FPS)*, where $\prec$ is a strict partial order (irreflexive, asymmetric and transitive). An interrupt controller $\mathtt{Sch}(\mathscr{H}, \prec)$ with $\prec$ as a FPS is defined by a nested timed automaton $(T, \mathcal{A}_0, \varDelta)$ over a set of input symbols $\varSigma$ where,

- $\varSigma = \{\mathtt{Coming}_P \mid \text{for each } P \in \mathscr{H}\}$.
- $T = \{\mathtt{Guarded}(P) \mid \text{for each } P \in \mathscr{H}\} \cup \{\mathcal{A}_{idle}\}$, where $\mathcal{A}_{idle}$ is a singleton timed automaton without any transitions.
- $\mathcal{A}_0 = \mathcal{A}_{idle}$.
- $\varDelta$ is defined by $\varDelta_{idle} \cup \varDelta_{push} \cup \varDelta_{pop} \cup \varDelta_{internal}$, where
  - $\varDelta_{idle} = \{\mathcal{A}_{idle} \xrightarrow{\mathtt{Coming}_P, push} \mathcal{A} \mid \forall P \in \mathscr{H}, \mathcal{A} = \mathtt{guarded}(P)\}$.
  - $\varDelta_{push} = \{\mathcal{A} \xrightarrow{\mathtt{Coming}_{P'}, push} \mathcal{A}' \mid \forall P, P' \in \mathscr{H}, P \prec P' \wedge \mathcal{A} = \mathtt{guarded}(P) \wedge \mathcal{A}' = \mathtt{guarded}(P')\}$.
  - $\varDelta_{pop} = \{\mathcal{A} \xrightarrow{pop} \varepsilon \mid \forall P \in \mathscr{H}, \mathcal{A} = \mathtt{guarded}(P)\}$.
  - $\varDelta_{internal} = \{\mathcal{A} \xrightarrow{\mathtt{Coming}_{P'}, internal} \mathcal{A}' \mid \forall P, P' \in \mathscr{H}, P \not\prec P' \wedge P \not\succ P' \wedge \mathcal{A} = \mathtt{guarded}(P) \wedge \mathcal{A}' = \mathtt{guarded}(P')\}$.

After performing parallel composition with a TA as an environment, we are allowed to check the deadline of each interrupt handler P_i through the reachability problem on the error location in $\mathtt{Guarded}(P_i)$, considering the fact that a finite number of interrupt handlers are effectively invoked.

6 Related Work

After TAs [1] had been proposed, lots of researches were intended timed context switches. TPDAs were firstly proposed in [2], which enjoys decidability of reachability problem. Dang proved in [14] the decidability of binary reachability of TPDAs. All clocks in TPDAs were treated globally, which were not effected when the context switches.

Our model relied heavily on a recent significant result, named *dense timed pushdown automata (DTPDAs)* [3]. The difference between DTPDAs and NeTAs was the hierarchical feature. In NeTAs, a finite set of local clocks were pushed into the stack at the same time. When a pop action happens, the values of clocks belonging to popped TA were popped simultaneously and reused. This feature eased much for modelling the behavior of time-aware software. In DTPDAs, local clocks must be dealt within some proper bookkeeping process, which was not essential part of the analysis. In [15], a discrete version of DTPDAs, named *discrete timed pushdown automata* was introduced, where time was incremented in discrete steps and thus the ages of clocks and stack symbols are in the natural numbers. This made the reachability problem much simpler, and easier for efficient implementation.

Based on *recursive state machines* [16], two similar timed extensions, *timed recursive state machines (TRSMs)* [5] and *recursive timed automata (RTAs)* [4], were given independently. The main differences from NeTAs were, the two models had no stack time-update during progress transitions, and the number of clocks was essentially finite. The *hierarchical timed automata (HTAs)* [17] kept the similar structure of clocks, where only a bounded number of levels were treated, while NeTAs treated an unbounded number of levels.

The class of *extended timed pushdown automata (ETPDAs)* was proposed in [5]. An ETPDA was a pushdown automaton enriched with a set of clocks, with an additional stack used to store/restore clock valuations. Two stacks were independently. ETPDAs have the same expressiveness with TRTMs via weak timed bisimulation. The reachability problem of ETPDAs was undecidable, while the decidability held with a syntactic restriction on the stack.

Controller automata (CAs) [18,11], was proposed to analyze interrupts. In a CA, a TA was assigned to each state. A TA at a state may be preempted by another state by a labeled transition. The number of clocks of CAs were finite, and thus when existing preemption loop, only the newest timed context were kept. Given a strict partial order over states, an ordered controller automaton was able to be faithfully encoded into a TA, and thus safety property of the restrictive version was preserved.

The *updatable timed automata (UTAs)* [9] proposed the possibility of updating the clocks in a more elaborate way, where the value of a clock could be reassigned to a basic arithmetic computation result of values of other clocks. UTAs raised up another way to accumulate time when timed context switches, and thus *updatable timed pushdown automata (UTPDAs)* could be an interesting extension.

7 Conclusion

This paper proposed a timed model called *nested timed automata (NeTAs)*. A NeTA was a pushdown system with a finite set of TAs as stack symbols. All clocks in the stack elapse uniformly, capable to model the timed behavior of the suspended components in the stack. The safety property of NeTAs was shown to be decidable by encoding NeTAs to DTPDAs. As an example of its application, behavior of multi-level interrupt handling is concisely modelled and its deadline analysis is encoded as a safety property.

We are planning to develop a tool based on NeTAs. Instead of general NeTAs, we will restrict a class such that a pop action occurs only with an integer-valued age. We expect this subclass of NeTAs can be encoded into updatable TPDAs (without local age), which would be more efficiently implemented.

Acknowledgements. This work is supported by the NSFC-JSPS bilateral joint research project (61011140074), NSFC(61100052, 61003013, 61033002, 6126113 0589), and JSPS KAKENHI Grant-in-Aid for Scientific Research(B) (23300008 and 25280023).

References

1. Alur, R., Dill, D.L.: A Theory of Timed Automata. Theoretical Computer Science 126, 183–235 (1994)
2. Bouajjani, A., Echahed, R., Robbana, R.: On the Automatic Verification of Systems with Continuous Variables and Unbounded Discrete Data Structures. In: Antsaklis, P.J., Kohn, W., Nerode, A., Sastry, S.S. (eds.) HS 1994. LNCS, vol. 999, pp. 64–85. Springer, Heidelberg (1995)
3. Abdulla, P.A., Atig, M.F., Stenman, J.: Dense-Timed Pushdown Automata. In: Proceedings of the LICS 2012, pp. 35–44. IEEE Computer Society (2012)
4. Trivedi, A., Wojtczak, D.: Recursive Timed Automata. In: Bouajjani, A., Chin, W.-N. (eds.) ATVA 2010. LNCS, vol. 6252, pp. 306–324. Springer, Heidelberg (2010)
5. Benerecetti, M., Minopoli, S., Peron, A.: Analysis of Timed Recursive State Machines. In: Proceedings of the TIME 2010, pp. 61–68. IEEE Computer Society (2010)
6. Li, G., Cai, X., Ogawa, M., Yuen, S.: Nested Timed Automata. Technical Report IS-RR-2013-004, JAIST (2013)
7. Henzinger, T.A., Nicollin, X., Sifakis, J., Yovine, S.: Symbolic Model Checking for Real-Time Systems. Information and Computation 111, 193–244 (1994)
8. Ogawa, M., Cai, X.: On Reachability of Dense Timed Pushdown Automata. Technical Report IS-RR-2013-005, JAIST (2013)
9. Bouyer, P., Dufourd, C., Fleury, E., Petit, A.: Updatable Timed Automata. Theoretical Computer Science 321, 291–345 (2004)
10. Bengtsson, J., Yi, W.: Timed Automata: Semantics, Algorithms and Tools. In: Desel, J., Reisig, W., Rozenberg, G. (eds.) ACPN 2003. LNCS, vol. 3098, pp. 87–124. Springer, Heidelberg (2004)
11. Li, G., Yuen, S., Adachi, M.: Environmental Simulation of Real-Time Systems with Nested Interrupts. In: Proceedings of the TASE 2009, pp. 21–28. IEEE Computer Society (2009)
12. Brylow, D., Palsberg, J.: Deadline Analysis of Interrupt-Driven Software. IEEE Transactions on Software Engineering (TSE) 30, 634–655 (2004)
13. Fersman, E., Krcal, P., Pettersson, P., Yi, W.: Task Automata: Schedulability, Decidability and Undecidability. Information and Computation 205, 1149–1172 (2007)
14. Dang, Z.: Pushdown Timed Automata: A Binary Reachability Characterization and Safety Verification. Theoretical Computer Science 302, 93–121 (2003)
15. Abdulla, P.A., Atig, M.F., Stenman, J.: The Minimal Cost Reachability Problem in Priced Timed Pushdown Systems. In: Dediu, A.-H., Martín-Vide, C. (eds.) LATA 2012. LNCS, vol. 7183, pp. 58–69. Springer, Heidelberg (2012)
16. Alur, R., Benedikt, M., Etessami, K., Godefroid, P., Reps, T.W., Yannakakis, M.: Analysis of Recursive State Machines. ACM Transactions on Programming Languages and Systems (TOPLAS) 27, 786–818 (2005)
17. David, A., Möller, M.O.: From HUPPAAL to UPPAAL - A Translation from Hierarchical Timed Automata to Flat Timed Automata. Technical Report RS-01-11, BRICS (2001)
18. Li, G., Cai, X., Yuen, S.: Modeling and Analysis of Real-Time Systems with Mutex Components. International Journal of Foundations of Computer Science (IJFCS) 23, 831–851 (2012)

On Fixed Points of Strictly Causal Functions[⋆]

Eleftherios Matsikoudis and Edward A. Lee

University of California, Berkeley

Abstract. We ask whether strictly causal components form well defined systems when arranged in feedback configurations. The standard interpretation for such configurations induces a fixed-point constraint on the function modelling the component involved. We define strictly causal functions formally, and show that the corresponding fixed-point problem does not always have a well defined solution. We examine the relationship between these functions and the functions that are strictly contracting with respect to a generalized distance function on signals, and argue that these strictly contracting functions are actually the functions that one ought to be interested in. We prove a constructive fixed-point theorem for these functions, and introduce a corresponding induction principle.

1 Introduction

This work is part of a larger effort aimed at the construction of well defined mathematical models that will inform the design of programming languages and model-based design tools for timed systems. We use the term "timed" rather liberally here to refer to any system that will determinately order its events relative to some physical or logical clock. But our emphasis is on timed computation, with examples ranging from concurrent and distributed real-time software to hardware design, and from discrete-event simulation to continuous-time and hybrid modelling, spanning the entire development process of what we would nowadays refer to as cyber-physical systems. Our hope is that our work will lend insight into the design and application of the many languages and tools that have and will increasingly come into use for the design, simulation, and analysis of such systems. Existing languages and tools to which this work applies, to varying degrees, include hardware description languages such as VHDL and SystemC, modeling and simulation tools such as Simulink and LabVIEW, network simulation tools such as ns-2/ns-3 and OPNET, and general-purpose simulation formalisms such as DEVS (e.g., see [1]).

[⋆] This work was supported in part by the Center for Hybrid and Embedded Software Systems (CHESS) at UC Berkeley, which receives support from the National Science Foundation (NSF awards #0720882 (CSR-EHS: PRET), #0931843 (CPS: Large: ActionWebs), and #1035672 (CPS: Medium: Ptides)), the Naval Research Laboratory (NRL #N0013-12-1-G015), and the following companies: Bosch, National Instruments, and Toyota.

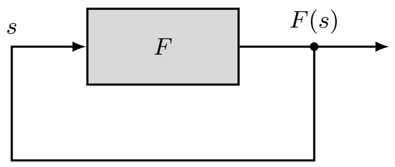


Fig. 1. Block-diagram of a functional component F in feedback. The input signal s and the output signal $F(s)$ are but the same signal; that is, $s = F(s)$.

Considering the breadth of our informal definition for timed systems, we cannot hope for a comprehensive formalism or syntax for such systems at a granularity finer than that of a network of components. We will thus ignore any internal structure or state, and think of any particular component as an opaque flow transformer. Formally, we will model such components as functions, and use a suitably generalized concept of signal as flow. This point of view is consistent with the one presented by most of the languages and tools mentioned above.

The greatest challenge in the construction of such a model is, by and large, the interpretation of feedback. Feedback is an extremely useful control mechanism, present in all but the most trivial systems. But it makes systems self-referential, with one signal depending on another, and vice versa (see Fig. 1). Mathematically, this notion of self-reference manifests itself in the form of a fixed-point problem, as illustrated by the simple block-diagram of Fig. 1: the input signal s and the output signal $F(s)$ are but the same signal transmitted over the feedback wire of the system; unless F has a fixed point, the system has no model; unless F has a unique or otherwise canonical fixed point, the model is not uniquely determined; unless we can construct the unique or otherwise canonical fixed point of F, we cannot know what the model is.

In this work, we consider the fixed-point problem of strictly causal functions, namely functions modelling components whose output at any time depends only on past values of the input. After a careful, precise formalization of this folklore, yet universally accepted, definition, we show that the property of strict causality is by itself too weak to accommodate a uniform fixed-point theory. We then consider functions that are strictly contracting with respect to a suitably defined generalized distance function on signals, and study their relationship to strictly causal ones, providing strong evidence that the former are actually the functions that one ought to be interested in. Finally, we consider the fixed-point problem of these functions, which is not amenable to classical methods (see [2, thm. A.2 and thm. A.4]), and prove a constructive fixed-point theorem, what has resisted previous efforts (e.g., see [3], [4]). We also introduce a corresponding induction principle, and discuss its relationship to the standard ones afforded by the fixed-point theories of order-preserving functions and contraction mappings.

For lack of space, we omit all proofs; they can be found in [2]. We also assume a certain level of familiarity with the theory of generalized ultrametric spaces (e.g., see [5]) and order theory (e.g., see [6]).

2 Signals

We postulate a non-empty set T of *tags*, and a total order relation $\preceq$ on T. We use T to represent our time domain. The order relation $\preceq$ is meant to play the role of a chronological precedence relation, and therefore, it is reasonable to require that $\preceq$ be a total order. We note, however, that such a requirement is often unnecessary, and in fact, all results of Sect. 5 remain valid when $\langle \mathrm{T}, \preceq \rangle$ is only partially ordered.

We would like to define signals as functions over an independent variable ranging over T. But being primarily concerned with computational systems, we should expect our definition to accommodate the representation of variations that may be undefined for some instances or even periods of time. In fact, we think of such instances and periods of time as part of the variational information. Such considerations lead directly to the concept of partial function.

We postulate a non-empty set V of *values.*

Definition 1. *An* event *is an ordered pair* $\langle \tau, v \rangle \in \mathrm{T} \times \mathrm{V}$*.*

Definition 2. *A* signal *is a single-valued set of events.*

We write S for the set of all signals.

Our concept of signal is based on [7]. But here, unlike in [7], we restrict signals to be single-valued.

Notice that the empty set is vacuously single-valued, and hence, a signal, which we call the *empty signal.*

We adopt common practice in modern set theory and identify a function with its graph. A signal is then a function with domain some subset of T, and range some subset of V, or in other words, a partial function from T to V.

For every $s_1, s_2 \in \mathrm{S}$ and $\tau \in \mathrm{T}$, we write $s_1(\tau) \simeq s_2(\tau)$ if and only if one of the following is true:

1. $\tau \notin \mathsf{dom}\, s_1$ and $\tau \notin \mathsf{dom}\, s_2$;
2. $\tau \in \mathsf{dom}\, s_1$, $\tau \in \mathsf{dom}\, s_2$, and $s_1(\tau) = s_2(\tau)$.

There is a natural, if abstract, notion of distance between any two signals, corresponding to the largest segment of time closed under time precedence, and over which the two signals agree; the larger the segment, the closer the two signals. Under certain conditions, this can be couched in the language of metric spaces (e.g., see [7], [8], [9]). All one needs is a map from such segments of time to non-negative real numbers. But this step of indirection excessively restricts the kind of ordered sets that one can use as models of time (see [4]), and can be avoided as long as one is willing to think about the notion of distance in more abstract terms, and use the language of generalized ultrametric spaces instead (see [5]).

We write d for a map from $\mathrm{S} \times \mathrm{S}$ to $\mathscr{L}\langle \mathrm{T}, \preceq \rangle$ such that for every $s_1, s_2 \in \mathrm{S}$,[1]

$$\mathrm{d}(s_1, s_2) = \{\tau \mid \tau \in \mathrm{T}, \text{ and for every } \tau' \preceq \tau,\ s_1(\tau') \simeq s_2(\tau')\} \ .$$

Proposition 1. $\langle \mathrm{S}, \mathscr{L}\langle \mathrm{T}, \preceq \rangle, \supseteq, \mathrm{T}, \mathrm{d} \rangle$ *is a generalized ultrametric space.*

The following is immediate, and indeed, equivalent:

Proposition 2. *For every* $s_1, s_2, s_3 \in \mathrm{S}$*, the following are true:*

1. $\mathrm{d}(s_1, s_2) = \mathrm{T}$ *if and only if* $s_1 = s_2$*;*
2. $\mathrm{d}(s_1, s_2) = \mathrm{d}(s_2, s_1)$*;*
3. $\mathrm{d}(s_1, s_2) \supseteq \mathrm{d}(s_1, s_3) \cap \mathrm{d}(s_3, s_2)$*.*

We refer to clause 1 as the *identity of indiscernibles*, clause 2 as *symmetry*, and clause 3 as the *generalized ultrametric inequality*.

Proposition 3. $\langle \mathrm{S}, \mathscr{L}\langle \mathrm{T}, \preceq \rangle, \supseteq, \mathrm{T}, \mathrm{d} \rangle$ *is spherically complete.*

Spherical completeness implies Cauchy-completeness, but the converse is not true in general (see [10, prop. 10], [2, exam. 2.6]). The importance of spherical completeness will become clear in Section 4 (see Theorem 2).

There is also a natural order relation on signals, namely the prefix relation on signals.

We write $\sqsubseteq$ for a binary relation on S such that for every $s_1, s_2 \in \mathrm{S}$,

$$s_1 \sqsubseteq s_2 \iff \begin{array}{l}\text{for every } \tau, \tau' \in \mathrm{T}, \text{ if } \tau \in \mathsf{dom}\, s_1 \text{ and } \tau' \preceq \tau, \text{ then} \\ s_1(\tau') \simeq s_2(\tau').\end{array}$$

We say that s_1 is a *prefix* of s_2 if and only if $s_1 \sqsubseteq s_2$.

It is easy to see that for every $s_1, s_2 \in \mathrm{S}$, $s_1 \sqsubseteq s_2$ if and only if there is $L \in \mathscr{L}\langle \mathrm{T}, \preceq \rangle$ such that $s_1 = s_2 \restriction L$,[3] and in particular, $s_1 \sqsubseteq s_2$ if and only if $s_1 = s_2 \restriction \mathrm{d}(s_1, s_2)$ (see [2, prop. 2.12 and thm. 2.13]).

Notice that for every $s \in \mathrm{S}$, $\emptyset \sqsubseteq s$; that is, the empty signal is a prefix of every signal.

Proposition 4. $\langle \mathrm{S}, \sqsubseteq \rangle$ *is a complete semilattice.*

For every $C \subseteq \mathrm{S}$ such that C is consistent in $\langle \mathrm{S}, \sqsubseteq \rangle$, we write $\bigsqcup C$ for the least upper bound of C in $\langle \mathrm{S}, \sqsubseteq \rangle$.

For every $s_1, s_2 \in \mathrm{S}$, we write $s_1 \sqcap s_2$ for the greatest lower bound of s_1 and s_2 in $\langle \mathrm{S}, \sqsubseteq \rangle$.

[1] For every ordered set $\langle P, \leqslant \rangle$, we write $\mathscr{L}\langle P, \leqslant \rangle$ for the set of all lower sets[2] of $\langle P, \leqslant \rangle$.

[2] For every ordered set $\langle P, \leqslant \rangle$, and every $L \subseteq P$, L is a *lower set* (also called a *down-set* or an *order ideal*) of $\langle P, \leqslant \rangle$ if and only if for any $p_1, p_2 \in P$, if $p_1 \leqslant p_2$ and $p_2 \in L$, then $p_1 \in L$.

[3] For every function f and every set A, we write $f \restriction A$ for the *restriction* of f to A, namely the function $\{\langle a, b \rangle \mid \langle a, b \rangle \in f \text{ and } a \in A\}$.

3 Causal and Strictly Causal Functions

Causality is a concept of fundamental importance in the study of timed systems. Informally, it represents the constraint that, at any time instance, the output events of a component do not depend on its future input events. This is only natural for components that model or simulate physical processes, or realize online algorithms; an effect cannot precede its cause.

Assume a non-empty partial function F on S.

We say that F is *causal* if and only if there is a partial function f such that for every $s \in \mathsf{dom}\, F$ and every $\tau \in \mathrm{T}$,

$$F(s)(\tau) \simeq f(s \restriction \{\tau' \mid \tau' \preceq \tau\}, \tau) \ .$$

Example 1. Suppose that $\mathrm{T} = \mathbb{R}$, and $\preceq$ is the standard order on $\mathbb{R}$. Let p be a positive real number, and F a function on S such that for every $s \in \mathrm{S}$ and every $\tau \in \mathrm{T}$,

$$F(s)(\tau) \simeq \begin{cases} s(\tau) & \text{if there is } i \in \mathbb{Z} \text{ such that } \tau = p \cdot i; \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Clearly, F is causal.

The function of Example 1 models a simple sampling process.

Now, as explained in Section 1, due to its relevance to the interpretation of feedback, of special interest is whether any particular partial function F on S has a fixed point, that is, whether there is $s \in \mathrm{S}$ such that $s = F(s)$ (see Fig. 1), and whether that fixed point is unique. The function of Example 1 has uncountably many fixed points, namely every $s \in \mathrm{S}$ such that

$$\mathsf{dom}\, s \subseteq \{\tau \mid \text{there is } i \in \mathbb{Z} \text{ such that } \tau = p \cdot i\} \ .$$

And it is easy to construct a causal function that does not have a fixed point.

Example 2. Let τ be a tag in T, v a value in V, and F a function on S such that for every $s \in \mathrm{S}$,

$$F(s) = \begin{cases} \emptyset & \text{if } \tau \in \mathsf{dom}\, s; \\ \{\langle \tau, v \rangle\} & \text{otherwise.} \end{cases}$$

It is easy to verify that F is causal. But F has no fixed point; for $F(\{\langle \tau, v \rangle\}) = \emptyset$, whereas $F(\emptyset) = \{\langle \tau, v \rangle\}$.

The function of Example 2 models a component whose behaviour at τ resembles a logical inverter, turning presence of event into absence, and vice versa.

Strict causality is causality bar instantaneous reaction. Informally, it is the constraint that, at any time instance, the output events of a component do not depend on its present or future input events. This operational definition has its origins in natural philosophy, and is of course inspired by physical reality: every physical system, at least, in the classical sense, is a strictly causal system.

We say that F is *strictly causal* if and only if there is a partial function f such that for every $s \in \mathsf{dom}\, F$ and every $\tau \in \mathrm{T}$,

$$F(s)(\tau) \simeq f(s \upharpoonright \{\tau' \mid \tau' \prec \tau\}, \tau) \ .$$

Proposition 5. *If F is strictly causal, then F is causal.*

Of course, the converse is false. For example, the sampling function of Example 1 is causal but not strictly causal.

Example 3. Suppose that $\mathrm{T} = \mathbb{R}$, and $\preceq$ is the standard order on $\mathbb{R}$. Let F be a function on S such that for every $s \in \mathrm{S}$ and every $\tau \in \mathrm{T}$,

$$F(s)(\tau) \simeq s(\tau - 1) \ .$$

Clearly, F is stricty causal.

The function of Example 3 models a simple constant-delay component. It is in fact a "delta causal" function, as defined in [7] and [8], and it is not hard to see that every such function is strictly causal (as is every "Δ-causal" function, as defined in [11] and [12]). The function of our next example models a variable reaction-time component, and is a strictly causal function that is not "delta causal" (nor "Δ-causal").

Example 4. Suppose that $\mathrm{T} = [0, \infty)$, $\preceq$ is the standard order on $[0, \infty)$, and $\mathrm{V} = (0, \infty)$. Let F be a function on S such that for every $s \in \mathrm{S}$ and any $\tau \in \mathrm{T}$,

$$F(s)(\tau) \simeq \begin{cases} 1 & \text{if there is } \tau' \in \mathsf{dom}\, s \text{ such that } \tau = \tau' + s(\tau'); \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Clearly, F is strictly causal.

Now, the function of Example 3 has uncountably many fixed-points, namely every $s \in \mathrm{S}$ such that for every $\tau \in \mathbb{R}$,

$$s(\tau) \simeq s(\tau + 1) \ .$$

And the function of Example 4 has exactly one fixed point, namely the empty signal. And having ruled out instantaneous reaction, the reason behind the lack of fixed point in Example 2, one might expect that every strictly causal function has a fixed point. But this is not the case.

Example 5. Suppose that $\mathrm{T} = \mathbb{Z}$, $\preceq$ is the standard order on $\mathbb{Z}$, and $\mathrm{V} = \mathbb{N}$. Let F be a function on S such that for every $s \in \mathrm{S}$ and every $\tau \in \mathrm{T}$,

$$F(s)(\tau) \simeq \begin{cases} s(\tau - 1) + 1 & \text{if } \tau - 1 \in \mathsf{dom}\, s; \\ 0 & \text{otherwise.} \end{cases}$$

Clearly, F is strictly causal. However, F does not have a fixed point; any fixed point of F would be an order-embedding from the integers into the natural numbers, which is of course impossible.

Example 5 alone is enough to suggest that the classical notion of strictly causality is by itself too general to support a useful theory of timed systems.

4 Contracting and Strictly Contracting Functions

There is another, intuitively equivalent way to articulate the property of causality: a component is causal just as long as any two possible output signals differ no earlier than the input signals that produced them (see [8, p. 36], [13, p. 11], and [14, p. 383]). And this can be very elegantly expressed using the generalized distance function of Section 2.

We say that F is *contracting* if and only if for every $s_1, s_2 \in \mathsf{dom}\, F$,

$$\mathrm{d}(F(s_1), F(s_2)) \supseteq \mathrm{d}(s_1, s_2) \enspace .$$

In other words, a function is contracting just as long as the distance between any two signals in the range of the function is smaller than or equal to that between the signals in the domain of the function that map to them.

In [4, def. 5], causal functions were defined to be the contracting functions. Here, we prove that indeed they are.

Theorem 1. *F is causal if and only if F is contracting.*

Following the same line of reasoning, one might expect that a component is strictly causal just as long as any two possible output signals differ later, if at all, than the signals that produced them (see [8, p. 36]).

We say that F is *strictly contracting* if and only if for every $s_1, s_2 \in \mathsf{dom}\, F$ such that $s_1 \neq s_2$,

$$\mathrm{d}(F(s_1), F(s_2)) \supset \mathrm{d}(s_1, s_2) \enspace .$$

Proposition 6. *If F is strictly contracting, then F is contracting.*

In [3, p. 484], Naundorf defined strictly causal functions as the functions that we here call strictly contracting, and in [4, def. 6], that definition was rephrased using the generalized distance function to explicitly identify strictly causal functions with the strictly contracting functions. But the relationship between the proposed definition and the classical notion of strict causality was never formally examined. The next theorem, which is a direct application of the fixed-point theorem of Priess-Crampe and Ribenboim for strictly contracting functions on spherically complete generalized ultrametric spaces (see [5, thm. 1]), implies that, in fact, the two are not the same.

Assume non-empty $X \subseteq \mathrm{S}$.

Theorem 2. *If $\langle X, \mathscr{L}\langle \mathrm{T}, \preceq \rangle, \supseteq, \mathrm{T}, \mathrm{d} \rangle$ is spherically complete, then every strictly contracting function on X has exactly one fixed point.*

By Theorem 2, the function of Example 5 is not strictly contracting. What is then the use, if any, of strictly contracting functions in a fixed-point theory for strictly causal functions? The next couple of theorems are key in answering this question.

Theorem 3. *If F is strictly contracting, then F is strictly causal.*

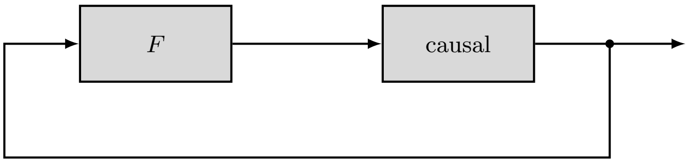


Fig. 2. A functional component realizes a strictly contracting function F if and only if the cascade of the component and any arbitrary causal component has a unique, well defined behaviour when arranged in a feedback configuration.

Theorem 4. *If* $\langle \mathsf{dom}\, F, \mathscr{L}\langle \mathrm{T}, \preceq\rangle, \supseteq, \mathrm{T}, \mathrm{d}\rangle$ *is spherically complete, then* F *is strictly contracting if and only if for every causal function* F' *from* $\mathsf{ran}\, F$ *to* $\mathsf{dom}\, F$*,* $F' \circ F$ *has a fixed point.*

An informal but informative way of reading Theorem 4 is the following: a functional component realizes a strictly contracting function if and only if the cascade of the component and any arbitrary causal filter has a unique, well defined behaviour when arranged in a feedback configuration (see Fig. 2); that is, the components that realize strictly contracting functions are those functional components that maintain the consistency of the feedback loop no matter how we chose to filter the signal transmitted over the feedback wire, as long as we do so in a causal way.

Theorem 4 completely characterizes strictly contracting functions in terms of the classical notion of causality, identifying the class of all such functions as the largest class of functions that have a fixed point not by some fortuitous coincidence, but as a direct consequence of their causality properties.

The implication of Theorem 3 and 4, we believe, is that the class of strictly contracting functions is the largest class of strictly causal functions that one can reasonably hope to attain a uniform fixed-point theory for.

Finally, if we further require that the domain of any signal in the domain of a function is well ordered under $\prec$, then the difference between a strictly causal function and a strictly contracting one vanishes.

Theorem 5. *If for every* $s \in \mathsf{dom}\, F$*,* $\langle \mathsf{dom}\, s, \prec\rangle$ *is well ordered, then* F *is strictly causal if and only if* F *is strictly contracting.*

For example, the variable-reaction-time function of Example 4 is not strictly contracting, as can be witnessed by the signals $\{\langle \frac{1}{2n+1}, \frac{2}{4n^2-1}\rangle \mid n \in \mathbb{N}\}$ and $\{\langle \frac{1}{2n}, \frac{1}{2n(n-1)}\rangle \mid n \in \mathbb{N} \text{ and } n \geq 2\}$, but its restriction to the set of all discrete-event signals, for instance, is.

Theorem 5, immediately applicable in the case of discrete-event systems, is most pleasing considering our emphasis on timed computation. It implies that for all kinds of computational timed systems, where components are expected to operate on discretely generated signals, including all programming languages and model-based design tools mentioned in the beginning of the introduction, the strictly contracting functions are exactly the strictly causal ones.

5 Fixed-Point Theory

We henceforth concentrate on the strictly contracting functions, and begin to develop the rudiments of a constructive fixed-point theory for such functions.

For every partial endofunction F on S, and any $s \in \mathsf{dom}\, F$, we say that s is a *post-fixed point* of F if and only if $s \sqsubseteq F(s)$.

Assume non-empty $X \subseteq$ S.

Lemma 1. *If $\langle X, \sqsubseteq\rangle$ is a subsemilattice of $\langle \mathrm{S}, \sqsubseteq\rangle$, then for every contracting function F on X, and every $s \in X$, the following are true:*

1. *$F(s) \sqcap F(F(s))$ is a post-fixed point of F;*
2. *if s is a post-fixed point of F, then $s \sqsubseteq F(s) \sqcap F(F(s))$.*

Lemma 2. *If $\langle X, \sqsubseteq\rangle$ is a subsemilattice of $\langle \mathrm{S}, \sqsubseteq\rangle$, then for every contracting function F on X, and any set P of post-fixed points of F, if P has a least upper bound in $\langle X, \sqsubseteq\rangle$, then $\bigsqcup_X P$ is a post-fixed point of F.*

Lemma 1 and 2 contain nearly all the ingredients of a transfinite recursion facilitating the construction of a chain that will converge to the desired fixed point. We may start with any arbitrary post-fixed point of the function F, and iterate through the function $\lambda x : X \,.\, F(x) \sqcap F(F(x))$ to form an ascending chain of such points. Every so often, we may take the supremum in $\langle X, \sqsubseteq\rangle$ of all signals theretofore constructed, and resume the process therefrom, until no further progress can be made. Of course, the phrase "every so often" is to be interpreted rather liberally here, and certain groundwork is required before we can formalize its transfinite intent.

We henceforth assume some familiarity with transfinite set theory, and in particular, ordinal numbers. The unversed reader may refer to any introductory textbook on set theory for details (e.g., see [15]).

Assume a subsemilattice $\langle X, \sqsubseteq\rangle$ of $\langle \mathrm{S}, \sqsubseteq\rangle$, and a function F on X.

We write $\mathsf{1m2}\, F$ for a function on X such that for any $s \in X$,

$$(\mathsf{1m2}\, F)(s) = F(s) \sqcap F(F(s)) \ .$$

In other words, $\mathsf{1m2}\, F$ is the function $\lambda x : X \,.\, F(x) \sqcap F(F(x))$.

Assume a directed-complete subsemilattice $\langle X, \sqsubseteq\rangle$ of $\langle \mathrm{S}, \sqsubseteq\rangle$, a contracting function F on X, and a post-fixed point s of F.

We let

$$(\mathsf{1m2}\, F)^0(s) = s \ ,$$

for every ordinal α,

$$(\mathsf{1m2}\, F)^{\alpha+1}(s) = (\mathsf{1m2}\, F)((\mathsf{1m2}\, F)^{\alpha}(s)) \ ,$$

and for every limit ordinal λ,

$$(\mathsf{1m2}\, F)^{\lambda}(s) = \bigsqcup\nolimits_X \{(\mathsf{1m2}\, F)^{\alpha}(s) \mid \alpha \in \lambda\} \ .$$

The following implies that for every ordinal α, $(\mathsf{1m2}\, F)^{\alpha}(s)$ is well defined:

Lemma 3. *If $\langle X, \sqsubseteq \rangle$ is a directed-complete subsemilattice of $\langle \mathrm{S}, \sqsubseteq \rangle$, then for every contracting function F on X, any post-fixed point s of F, and every ordinal α,*

1. $(\mathsf{1m2}\,F)^{\alpha}(s) \sqsubseteq F((\mathsf{1m2}\,F)^{\alpha}(s))$*;*
2. *for any $\beta \in \alpha$, $(\mathsf{1m2}\,F)^{\beta}(s) \sqsubseteq (\mathsf{1m2}\,F)^{\alpha}(s)$.*

By Lemma 3.2, and a simple cardinality argument, there is an ordinal α such that for every ordinal β such that $\alpha \in \beta$, $(\mathsf{1m2}\,F)^{\beta}(s) = (\mathsf{1m2}\,F)^{\alpha}(s)$. In fact, for every directed-complete subsemilattice $\langle X, \sqsubseteq \rangle$ of $\langle \mathrm{S}, \sqsubseteq \rangle$, there is a least ordinal α such that for every contracting function F on X, any post-fixed point s of F, and every ordinal β such that $\alpha \in \beta$, $(\mathsf{1m2}\,F)^{\beta}(s) = (\mathsf{1m2}\,F)^{\alpha}(s)$.

We write $\mathsf{oh}\,\langle X, \sqsubseteq \rangle$ for the least ordinal α such that there is no function φ from α to X such that for every $\beta, \gamma \in \alpha$, if $\beta \in \gamma$, then $\varphi(\beta) \sqsubset \varphi(\gamma)$.

In other words, $\mathsf{oh}\,\langle X, \sqsubseteq \rangle$ is the least ordinal that cannot be orderly embedded in $\langle X, \sqsubseteq \rangle$, which we may think of as the *ordinal height* of $\langle X, \sqsubseteq \rangle$. Notice that the Hartogs number of X is an ordinal that cannot be orderly embedded in $\langle X, \sqsubseteq \rangle$, and thus, $\mathsf{oh}\,\langle X, \sqsubseteq \rangle$ is well defined, and in particular, smaller than or equal to the Hartogs number of X.

Lemma 4. *If $\langle X, \sqsubseteq \rangle$ is a directed-complete subsemilattice $\langle X, \sqsubseteq \rangle$ of $\langle \mathrm{S}, \sqsubseteq \rangle$, then for every contracting function F on X, any post-fixed point s of F, and every ordinal α, if $(\mathsf{1m2}\,F)^{\alpha}(s)$ is not a fixed point of $\mathsf{1m2}\,F$, then $\alpha + 2 \in \mathsf{oh}\,\langle X, \sqsubseteq \rangle$.*

By Lemma 4, $(\mathsf{1m2}\,F)^{\mathsf{oh}\,\langle X, \sqsubseteq \rangle}(s)$ is a fixed point of $\mathsf{1m2}\,F$. Nevertheless, $(\mathsf{1m2}\,F)^{\mathsf{oh}\,\langle X, \sqsubseteq \rangle}(s)$ need not be a fixed point of F as intended. For example, if F is the function of Example 2, then for every ordinal α, $(\mathsf{1m2}\,F)^{\alpha}(\emptyset) = \emptyset$, even though $\emptyset$ is not a fixed point of F. This rather trivial example demonstrates how the recursion process might start stuttering at points that are not fixed under the function in question. If the function is strictly contracting, however, progress at such points is guaranteed.

Lemma 5. *If $\langle X, \sqsubseteq \rangle$ is a subsemilattice of $\langle \mathrm{S}, \sqsubseteq \rangle$, then for every strictly contracting function F on X, s is a fixed point of F if and only if s is a fixed point of $\mathsf{1m2}\,F$.*

We may at last put all the pieces together to obtain a constructive fixed-point theorem for strictly contracting functions on directed-complete subsemilattices of $\langle \mathrm{S}, \sqsubseteq \rangle$.

Theorem 6. *If $\langle X, \sqsubseteq \rangle$ is a directed-complete subsemilattice of $\langle \mathrm{S}, \sqsubseteq \rangle$, then for every strictly contracting function F on X, and any post-fixed point s of F,*

$$\mathsf{fix}\,F = (\mathsf{1m2}\,F)^{\mathsf{oh}\,\langle X, \sqsubseteq \rangle}(s) \enspace .$$

To be pedantic, Theorem 6 does not directly prove that F has a fixed point; unless there is a post-fixed point of F, the theorem is true vacuously. But by Lemma 1.1, for every $s \in X$, $(\mathsf{1m2}\,F)(s)$ is a post-fixed point of F.

This construction of fixed points as "limits of stationary transfinite iteration sequences" is very similar to the construction of extremal fixed points of monotone operators in [16] and references therein, where the function iterated is not $\mathsf{lm2}\,F$, but F itself. Notice, however, that if F preserves the prefix relation, then for any post-fixed point of F, $(\mathsf{lm2}\,F)(s) = F(s)$.

The astute reader will at this point anticipate the following:

Theorem 7. *If $\langle X, \sqsubseteq \rangle$ is a non-empty, directed-complete subsemilattice of $\langle \mathrm{S}, \sqsubseteq \rangle$, then for every strictly contracting function F on X,*

$$\mathsf{fix}\,F = \bigsqcup\nolimits_X \{s \mid s \in X \text{ and } s \sqsubseteq F(s)\} \ .$$

The construction of Theorem 7 is identical in form to Tarski's well known construction of greatest fixed points of order-preserving functions on complete lattices (see [17, thm. 1]).

We note here that $\mathsf{lm2}\,F$ is not, in general, order-preserving under the above premises (see [2, exam. 5.15]), as might be suspected, and thus, our fixed-point theorem is not a reduction to a standard order-theoretic one.

Now, having used transfinite recursion to construct fixed points, we may use transfinite induction to prove properties of them. And in particular, we may use Theorem 6 to establish a special proof rule.

Assume $P \subseteq \mathrm{S}$.

We say that P is *strictly inductive* if and only if every non-empty chain in $\langle P, \sqsubseteq \rangle$ has a least upper bound in $\langle P, \sqsubseteq \rangle$.

Note that P is strictly inductive if and only if $\langle P, \sqsubseteq \rangle$ is directed-complete (see [18, cor. 2]).

Theorem 8. *If $\langle X, \sqsubseteq \rangle$ is a non-empty, directed-complete subsemilattice of $\langle \mathrm{S}, \sqsubseteq \rangle$, then for every strictly contracting function F on X, and every non-empty, strictly inductive $P \subseteq X$, if for every $s \in P$, $(\mathsf{lm2}\,F)(s) \in P$, then $\mathsf{fix}\,F \in P$.*

Theorem 8 is an induction principle that one may use to prove properties of fixed points of strictly contracting endofunctions. We think of properties extensionally here; that is, a property is a set of signals. And the properties that are admissible for use with this principle are those that are non-empty and strictly inductive.

It is interesting to compare this principle with the fixed-point induction principle for order-preserving functions on complete partial orders (see [19]), which we will here refer to as *Scott-de Bakker induction*, and the fixed-point induction principle for contraction mappings on complete metric spaces (see [20]), which we will here refer to as *Reed-Roscoe induction* (see also [21], [22], [23]).

For a comparison between our principle and Scott-de Bakker induction, let F be a function of the most general kind of function to which both our principle and Scott-de Bakker induction apply, namely an order-preserving, strictly contracting function on a pointed, directed-complete subsemilattice $\langle X, \sqsubseteq \rangle$ of $\langle \mathrm{S}, \sqsubseteq \rangle$. Now assume a property $P \subseteq X$. If P is admissible for use with Scott-de Bakker induction, that is, closed under suprema in $\langle X, \sqsubseteq \rangle$ of arbitrary chains in

$\langle P, \sqsubseteq \rangle$, then $\{s \mid s \in P \text{ and } s \sqsubseteq F(s)\}$ is non-empty and strictly inductive. And if P is closed under F, then $\{s \mid s \in P \text{ and } s \sqsubseteq F(s)\}$ is trivially closed under $\mathsf{1m2}\,F$. Therefore, given any reasonable property-specification logic, our principle is at least as strong a proof rule as Scott-de Bakker induction. At the same time, the often inconvenient requirement that a property P that is admissible for use with Scott-de Bakker induction contain the least upper bound in $\langle X, \sqsubseteq \rangle$ of the empty chain, namely the least element in $\langle X, \sqsubseteq \rangle$, and the insistence that the least upper bound of every non-empty chain in $\langle P, \sqsubseteq \rangle$ be the same as in $\langle X, \sqsubseteq \rangle$ make it less likely that every property true of $\mathsf{fix}\,F$ that can be proved using our principle can also be proved using Scott-de Bakker induction. For this reason, we are inclined to say that, given any reasonable property-specification logic, our principle is a strictly stronger proof rule than Scott-de Bakker induction, in the case, of course, where both apply.

The relationship between our principle and Reed-Roscoe induction is less clear. $\langle \mathrm{S}, \mathscr{L}\langle \mathrm{T}, \preceq \rangle, \supseteq, \mathrm{T}, \mathrm{d} \rangle$ being a generalized ultrametric space rather than a metric one, it might even seem that there can be no common ground for a meaningful comparison between the two. Nevertheless, it is possible to generalize Reed-Roscoe induction in a way that extends its applicability to the present case, while preserving its essence. According to the generalized principle, then, for every strictly contracting function F on any Cauchy complete, non-empty, directed-complete subsemilattice of $\langle \mathrm{S}, \sqsubseteq \rangle$ such that every orbit under F is a Cauchy sequence, every non-empty property closed under limits of Cauchy sequences that is preserved by F is true of $\mathsf{fix}\,F$. One similarity between this principle and our own, and an interesting difference from Scott-de Bakker induction, is the lack of an explicit basis for the induction; as long as the property in question is non-empty, there is some basis available. In terms of closure and preservation of admissible properties, however, the two principles look rather divergent from one another. For example, the property of a signal having only a finite number of events in any finite interval of time is Cauchy complete, but not strictly inductive. On the other hand, by Lemma 1.1 and Theorem 7, our principle is better fit for proving properties that are closed under prefixes, such as, for example, the property of a signal having at most one event in any time interval of a certain fixed size. And for this reason, we suspect that, although complimentary to the generalized Read-Roscoe induction principle in theory, our principle might turn out to be more useful in practice, what can of course only be evaluated empirically.

6 Related Work

Fixed points have been used extensively in the construction of mathematical models in computer science. In most cases, ordered sets and monotone functions have been the more natural choice. But in the case of timed computation, metric spaces and contraction mappings have proved a better fit, and Tarski's fixed-point theorem and its variants have given place to Banach's contraction principle.

Common to all approaches using this kind of modeling framework that we know of is the requirement of a positive lower bound on the reaction time of each component in a system (e.g., see [20], [24], [12], [25], [7], [8], [9]). This constraint is used to guarantee that the functions modelling these components are actually contraction mappings with respect to the defined metrics. The motivation is of course the ability to use Banach's fixed-point theorem in the interpretation of feedback, but a notable consequence is the absence of non-trivial Zeno phenomena, what has always been considered a precondition for realism in the real-time systems community. And yet in modelling and simulation, where time is represented as an ordinary program variable, Zeno behaviours are not only realizable, but occasionally desirable as well. Simulating the dynamics of a bouncing ball, for example, will naturally give rise to a Zeno behaviour, and the mathematical model used to study or even define the semantics of the simulation environment should allow for that behaviour. This is impossible with the kind of metric spaces found in [20], [24], [12], [25], [7], [8], and [9] (see [4, sec. 4.1]).

Another limiting factor in the applicability of the existing approaches based on metric spaces is the choice of tag set. The latter is typically some unbounded subset of the real numbers, excluding other interesting choices, such as, for example, that of superdense time (see [4, sec. 4.3]).

Naundorf was the first to address these issues, abolishing the bounded reaction time constraint, and allowing for arbitrary tag sets (see [3]). He defined strictly causal functions as the functions that we here call strictly contracting, and used an ad hoc, non-constructive argument to prove the existence of a unique fixed point for every such function. Unlike that in [7], [8], and [9], Naundorf's definition of strict causality was at least sound (see Theorem 3), but nevertheless incomplete (e.g., see Example 5). It was rephrased in [4] using the generalized distance function to explicitly identify strictly causal functions with the strictly contracting ones. This provided access to the fixed-point theory of generalized ultrametric spaces, which, however, proved less useful than one might have hoped. The main fixed-point theorem of Priess-Crampe and Ribenboim for strictly contracting endofunctions offered little more than another non-constructive proof of Naundorf's theorem, improving only marginally on the latter by allowing the domain of the function to be any arbitrary spherically complete set of signals, and the few constructive fixed-point theorems that we know to be of any relevance (e.g., see *Proof of Theorem 9 for ordinal distances* in [10], [26, thm. 43]) were of limited applicability.

There have also been a few attempts to use complete partial orders and least fixed points in the study of timed systems. In [11], Yates and Gao reduced the fixed-point problem related to a system of so-called "Δ-causal" components to that of a suitably constructed Scott-continuous function, transferring the Kahn principle to networks of real-time processes, but once more, under the usual bounded reaction-time constraint. A more direct application of the principle in the context of timed systems was put forward in [27]. But the proposed definition of strict causality was still incomplete, unable to accommodate components with more arbitrarily varying reaction-times, such as the one modelled by the function of Example 4 restricted to the set of all discrete-event signals, and ultimately, systems with non-trivial Zeno behaviours. Finally, a more naive approach was proposed in [28], where components were modelled as Scott-continuous functions with respect to the prefix relation on signals, creating, of course, all kinds of causality problems, which, however, seem to have gone largely unnoticed.

7 Conclusion

Strictly contracting functions form the largest class of strictly causal functions for which fixed points exist uniformly and canonically. More importantly, they coincide with strictly causal function in the case of discrete-event computation. The constructive fixed-point theorem for such functions presented in this work is, we believe, a leap forward in our understanding of functional timed systems. But a complete treatment of such systems should allow for arbitrarily complex networks of components. What we need, then, is an abstract characterization of a class of structures that will support the development of the theory, and stay closed under the construction of products and function spaces of interest, enabling the treatment of arbitrary, even higher-order composition in a uniform and canonical way. This is the subject of future work.

References

1. Zeigler, B.P.: Theory of Modeling and Simulation. John Wiley & Sons (1976)
2. Matsikoudis, E., Lee, E.A.: The fixed-point theory of strictly causal functions. Technical Report UCB/EECS-2013-122, EECS Department, University of California, Berkeley (June 2013)
3. Naundorf, H.: Strictly causal functions have a unique fixed point. Theoretical Computer Science 238(1-2), 483–488 (2000)
4. Liu, X., Matsikoudis, E., Lee, E.A.: Modeling timed concurrent systems. In: Baier, C., Hermanns, H. (eds.) CONCUR 2006. LNCS, vol. 4137, pp. 1–15. Springer, Heidelberg (2006)
5. Priess-Crampe, S., Ribenboim, P.: Fixed points, combs and generalized power series. Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg 63(1), 227–244 (1993)
6. Davey, B.A., Priestley, H.A.: Introduction to Lattices and Order, 2nd edn. Cambridge University Press (2002)
7. Lee, E.A., Sangiovanni-Vincentelli, A.: A framework for comparing models of computation. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 17(12), 1217–1229 (1998)
8. Lee, E.A.: Modeling concurrent real-time processes using discrete events. Annals of Software Engineering 7(1), 25–45 (1999)
9. Liu, J., Lee, E.A.: On the causality of mixed-signal and hybrid models. In: Maler, O., Pnueli, A. (eds.) HSCC 2003. LNCS, vol. 2623, pp. 328–342. Springer, Heidelberg (2003)
10. Hitzler, P., Seda, A.K.: Generalized metrics and uniquely determined logic programs. Theoretical Computer Science 305(1-3), 187–219 (2003)
11. Yates, R.K., Gao, G.R.: A Kahn principle for networks of nonmonotonic real-time processes. In: Bode, A., Reeve, M., Wolf, G. (eds.) PARLE 1993. LNCS, vol. 694, pp. 209–227. Springer, Heidelberg (1993)

12. Yates, R.K.: Networks of real-time processes. In: Best, E. (ed.) CONCUR 1993. LNCS, vol. 715, pp. 384–397. Springer, Heidelberg (1993)
13. Broy, M.: Refinement of time. Theoretical Computer Science 253(1), 3–26 (2001)
14. Lee, E.A., Varaiya, P.: Structure and Interpretation of Signals and Systems, 2nd edn. (2011), `http://LeeVariaya.org`
15. Enderton, H.B.: Elements of Set Theory. Academic Press (1977)
16. Cousot, P., Cousot, R.: Constructive versions of Tarski's fixed point theorems. Pacific J. of Math. 82(1), 43–57 (1979)
17. Tarski, A.: A lattice-theoretical fixpoint theorem and its applications. Pacific J. of Math. 5(2), 285–309 (1955)
18. Markowsky, G.: Chain-complete posets and directed sets with applications. Algebra Universalis 6(1), 53–68 (1976)
19. Scott, D.S., de Bakker, J.W.: A theory of programs. Unpublished notes, Seminar on Programming, IBM Research Center, Vienna, Austria (1969)
20. Reed, G.M., Roscoe, A.W.: A timed model for communicating sequential processes. In: Kott, L. (ed.) ICALP 1986. LNCS, vol. 226, pp. 314–323. Springer, Heidelberg (1986)
21. Rounds, W.C.: Applications of topology to semantics of communicating processes. In: Brookes, S., Roscoe, A.W., Winskel, G. (eds.) Seminar on Concurrency. LNCS, vol. 197, pp. 360–372. Springer, Heidelberg (1985)
22. Roscoe, A.W.: Topology, computer science, and the mathematics of convergence. In: Reed, G.M., Roscoe, A.W., Wachter, R.F. (eds.) Topology and Category Theory in Computer Science, pp. 1–27. Oxford University Press, Inc., New York (1991)
23. Kozen, D., Ruozzi, N.: Applications of metric coinduction. In: Mossakowski, T., Montanari, U., Haveraaen, M. (eds.) CALCO 2007. LNCS, vol. 4624, pp. 327–341. Springer, Heidelberg (2007)
24. Reed, G.M., Roscoe, A.W.: Metric spaces as models for real-time concurrency. In: Main, M.G., Mislove, M.W., Melton, A.C., Schmidt, D. (eds.) MFPS 1987. LNCS, vol. 298, pp. 331–343. Springer, Heidelberg (1988)
25. Müller, O., Scholz, P.: Functional specification of real-time and hybrid systems. In: Maler, O. (ed.) HART 1997. LNCS, vol. 1201, pp. 273–285. Springer, Heidelberg (1997)
26. Krötzsch, M.: Generalized ultrametric spaces in quantitative domain theory. Theoretical Computer Science 368(1-2), 30–49 (2006)
27. Liu, X., Lee, E.A.: CPO semantics of timed interactive actor networks. Theoretical Computer Science 409(1), 110–125 (2008)
28. Caspi, P., Benveniste, A., Lublinerman, R., Tripakis, S.: Actors without directors: A Kahnian view of heterogeneous systems. In: Majumdar, R., Tabuada, P. (eds.) HSCC 2009. LNCS, vol. 5469, pp. 46–60. Springer, Heidelberg (2009)

Detecting Quasi-equal Clocks in Timed Automata

Marco Muñiz, Bernd Westphal, and Andreas Podelski

Albert-Ludwigs-Universität Freiburg, 79110 Freiburg, Germany
{muniz,westphal,podelski}@informatik.uni-freiburg.de

Abstract. A recent optimizations technique for timed model checking starts with a given specification of *quasi-equal* clocks. In principle, the zone graph can used to detect which clocks are quasi-equal; the construction of the zone graph would, however, defeat its very purpose (which is the optimization of this construction). In this paper, we present an abstraction that is effective for the goal of the optimization based on quasi-equal clocks: it is coarse enough to yield a drastic reduction of the size of the zone graph. Still, it is precise enough to identify a large class of quasi-equal clocks. The abstraction is motivated by an intuition about the way quasi-equalities can be tracked. We have implemented the corresponding reasoning method in the Jahob framework using an SMT solver. Our experiments indicate that our intuition may lead to a useful abstraction.

1 Introduction

Timed automata and timed model checking [1,12,18] have been very successfully applied for the verification of real-time systems. Still, the number of clocks in a timed automaton will always be an issue for scalability, and the optimization of timed model checking will always be a topic of research. Optimization techniques for timed model checking are often based on some notion of redundancy in the representation of the timed model and its behavior in terms of clock valuations. The detection of the corresponding redundancies is then a prerequisite for applying the optimization.

An example of a redundancy which gives the opportunity of a provably very effective optimization is the notion of *quasi-equal* clocks [11]. Two clocks x and y in a given timed automaton are quasi-equal if the invariant $x = y \vee x = 0 \vee y = 0$ holds. That is, in every step in every transition sequence, the two clocks x and y have equal value except for steps where one of them has been reset but not the other. As a consequence, the invariant $x = y$ can be violated only at single time points (i.e., during time periods of length zero). In a way, the violation of the invariant is an artefact of the model of the behavior of a timed systems by discrete sequences.

The optimization presented in [11] starts with a given specification of *quasi-equal* clocks. In principle, the zone graph can be used to detect which clocks are quasi-equal; the construction of the zone graph would, however, defeat its very purpose (which is the optimization of this construction). In this paper, we present an abstraction that is coarse enough to yield a drastic reduction of the zone graph and precise enough to identify a large class of quasi-clocks.

The abstraction is motivated by an intuition about the way quasi-equalities can be tracked. The intuition is that the behavior of a timed automaton over a non-zero period of time (without resets) will neither introduce new quasi-equalities nor "destroy" a quasi-equality, and hence we can apply the most coarse abstraction there. In contract, when different values for quasi-equal clocks arise in a sequence of configurations where time does not elapse, we must track the constants for the values of the clocks as precisely as possible (i.e., apply no proper abstraction). Thus, as an intermediate step for computing abstract zones, we must apply logical reasoning in order to infer whether the zone accounts for behavior of zero (as opposed to: non-zero) periods of time.

Our abstraction methods amounts to computing an abstraction of the zone graph. We use the abstract zone graph to detect quasi-equal clocks.

We have implemented our method in the Jahob verification framework. This allows us to represent zones (and abstract zones) by linear real arithmetic formulas and to perform the required logical reasoning (on the duration of the corresponding period of time) through calls of an SMT solver.

We have used our implementation to conduct preliminary experiments. The results indicate that the abstraction is effective for the goal of the optimization based on quasi-equal clocks: it is coarse enough to yield a drastic reduction of the size of the zone graph. Still, the abstraction is precise enough to identify a large class of quasi-clocks.

Related work. In [13], a syntactical pattern for timed automata is proposed. Automata constructed under this pattern are called *sequential* timed automata. There, the so-called *master* clocks will be reset at exactly the same point of time. Thus, master clocks are quasi-equal *by construction.* Unfortunately, the class of sequential timed automata is rather restricted. Therefore, one would like to be able to use the general class of timed automata and apply a method for detecting or checking quasi-equal clocks. This motivates the present work.

In [7] an abstraction method is proposed for detecting *equal* clocks (at a particular location). Once equal clocks at a particular location have been detected, a substitution method can be used to reduce the number of clocks in that location. Thus, the method can effectively reduce the number of clocks per location and also in the whole timed automaton. In this sense, the method [7] is similar wrt. its goal to our method. The difference lies in technical details. Computing quasi-equal clocks requires to detect zero time paths. For this, our method tracks the actual valuations of clocks, which the method [7] does not. As a consequence, it will not be able to detect quasi-equal clocks when they are not equal.

Outline of the paper. In Section 2, we use an example to recall and illustrate the notion of quasi-equal clocks. We also investigate the issue of zero time behaviors and we present the abstract transition system for detecting quasi-equal clocks. In Section 3, we present the formal setting. In Section 4, we formalize our

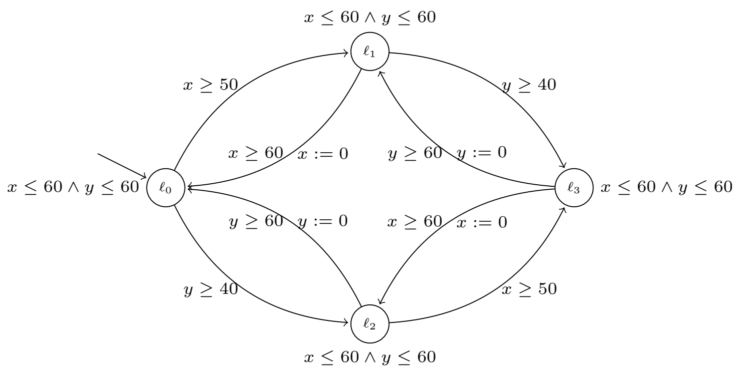


Fig. 1. Automaton with quasi-equal clocks x, y

abstraction by means of an abstract zone graph and a simulation relation. In Section 5, we present an example that illustrates effectiveness of the abstraction, i.e., the reduction of size through the abstraction. In Section 6, we present an algorithm for computing the reachable abstract zone graph (on the fly). The output of the algorithm is a relation that identifies which clocks are quasi-equal. We present the result of our experiments using an implementation of the algorithm.

2 Example

In this section, we illustrate our approach with help of the automaton in Figure 1. First, we refresh the notion of quasi-equal clocks. Next, we show the importance of the zero time behavior for detecting quasi-equal clocks. Finally, we show the corresponding abstract zone graph in which quasi-equal clocks can be efficiently detected.

Quasi-equal clocks. two clocks are quasi-equal if for all computations their values are equal or if one clock was reset then a reset must eventually occur for the other clock in zero time. Quasi-equal clocks are formally defined in Section 3. Consider the timed automata presented in Figure 1 with clocks x and y. Clearly, clocks x and y are not equal since for example in the computation $\langle \ell_0, \{\nu_0\}\rangle \rightarrow^* \langle \ell_1, x = 60 \wedge y = 0\rangle$ the configuration $\langle \ell_1, x = 60 \wedge y = 0\rangle$ yields different values for clocks x and y. However, note that the invariant $I(\ell_1) = x \leq 60 \wedge y \leq 60$ will prevent time to elapse at this configuration and thus the only successor of this configuration is $\langle \ell_0, x = 0 \wedge y = 0 \wedge x \leq 60 \wedge y \leq 60\rangle$ in which the values for clocks x and y are equal. Indeed, it is the case that for all computations from the automaton in Figure 1, the values for clocks x and y are either equal or the value of one clock is zero and a reset in zero time for the other clock occurs. Therefore, clocks x and y are quasi-equal. Clearly, the behavior of timed automata in Figure 1, can be simulated by using one clock, which yields an important speed up in the verification time.

$$\begin{tikzcd}
 & & & \langle \ell_1, x = y \rangle \arrow[dl] \arrow[dddrr] & & \\
 & & \langle \ell_0, P_1 \rangle \arrow[drr] & & & \\
 & & & & \langle \ell_2, P_1 \rangle \arrow[dlll] & \\
 {} \arrow[r] & \langle \ell_0, x = y \rangle \arrow[uuurr, bend left] \arrow[dddrr, bend right] & & & & \langle \ell_3, x = y \rangle \arrow[ul] \arrow[dl] \\
 & & & & \langle \ell_1, P_2 \rangle \arrow[ulll] & \\
 & & \langle \ell_0, P_2 \rangle \arrow[urr] & & & \\
 & & & \langle \ell_2, x = y \rangle \arrow[ul] \arrow[uuurr] & &
\end{tikzcd}$$

Fig. 2. Abstract zone graph corresponding to TA in Figure 1

Zero time behavior. As shown above, the zero time behavior of a timed automaton may cause quasi-equal clocks to arise. Therefore, zero time behavior is important for detecting quasi-equal clocks.

In the computation given above, first a reset for clock x occurs, then in the next transition a reset for clock y occurs. In this case, the length of the zero time computation is just one transition. However, this does not need to be the case. Consider the computation $\langle \ell_0, \{\nu_0\} \rangle \to^* \langle \ell_0, x = 0 \wedge y \leq 60 \wedge y \geq 60 \rangle$, the invariant $I(\ell_0) = x \leq 60 \wedge y \leq 60$ will not let time elapse and the only successor is $\langle \ell_2, x = 0 \wedge y \leq 60 \wedge y \geq 60 \rangle$. The invariant $I(\ell_2) = x \leq 60 \wedge y \leq 60$ will not let time elapse and the only successor is $\langle \ell_0, x = 0 \wedge y = 0 \wedge x \leq 60 \wedge y \leq 60 \rangle$, where time may elapse but the values for clocks x and y are equal. In general, the number of transitions that may occur in zero time might be infinite.

Abstract zone graph. Given a timed automaton, our method will construct an abstract transition system, which preserves as much as possible the zero time behavior of a timed automaton. The configurations of the abstract transition system are pairs consisting of locations and zones. The zones that we compute are of two types. Either a zone for which time is guaranteed not to elapse or a conjunction of equalities for clocks for which time may elapse.

Figure 2 shows the abstract transition system corresponding to the abstraction defined in Section 4 applied to the automaton in Figure 1, where zones P_1, P_2 are $P_1 := x \leq 60 \wedge x \geq 60 \wedge y = 0$ and $P_2 := y \leq 60 \wedge y \geq 60 \wedge x = 0$. Note, that in configurations where time may elapse the corresponding zone is $x = y$. In the transition induced by the edge $(\ell_1, x \geq 60, x := 0, \ell_0)$ from the automaton in Figure 1 and configuration $\langle \ell_1, P_1 \rangle$. The successor configuration $\langle \ell_0, P' \rangle$ with zone $P' = (P_1 \wedge x \geq 60)[\{x\} := 0] \wedge I(\ell_0)$ is not zero time. Therefore, $(P_1 \wedge x \geq 60)[\{x\} := 0] \wedge I(\ell_0)$ is relaxed to $x = y$, leading to configuration $\langle \ell_0, x = y \rangle$. Since, every zone in the set of reachable configurations implies that $x = y \vee x = 0 \vee y = 0$, our abstraction allow us to soundly conclude that x and y are quasi-equal.

3 Preliminaries

The formal basis for our work are timed automata [1]. Let $\mathbb{X}$ be a set of clocks. The set $\Phi(\mathbb{X})$ of simple clock constraints over $\mathbb{X}$ is defined by the grammar $\varphi ::= x \sim y \mid x - y \sim C \mid \varphi_1 \wedge \varphi_2$ where $x, y \in \mathbb{X}$, $C \in \mathbb{Q}_0^+$, and $\sim \in \{<, \leq, =, \geq, >\}$. Constraints of the form $x - y \sim C$ are called difference constraints. We assume the canonical satisfaction relation "$\models$" between *valuations* of the clocks $\nu : \mathbb{X} \to \mathbb{R}_0^+$ and simple clock constraints.

A timed automaton $\mathcal{A}$ is a tuple $(L, \mathbb{X}, I, E, \ell_0)$, which consists of a finite set of *locations* L, with typical element ℓ, a finite set of *clocks* $\mathbb{X}$, a mapping $I : L \to \Phi(\mathbb{X})$, that assigns to each location a *clock constraint*, and a set of *edges*. $E \subseteq L \times \Phi(\mathbb{X}) \times \mathcal{P}(\mathbb{X}) \times L$. An edge $e = (\ell, \varphi, Y, \ell') \in E$ from ℓ to ℓ' involves a *guard* $\varphi \in \Phi(\mathbb{X})$, and a *reset set* $Y \subseteq \mathbb{X}$. For easiness of presentation our definition of timed automaton does not include synchronizations or data variables. However, our idea can be extended to such "richer models" in a straight forward manner. For the rest of the paper, let us fix a timed automaton $\mathcal{A} = (L, \mathbb{X}, I, E, \ell_0)$.

A *zone* is the maximal set of clock valuations satisfying a clock constraint. Given timed automaton $\mathcal{A}$, for a clock constraint $Z \in \Phi(\mathbb{X})$, let $[Z]$ denote the maximal set of valuations satisfying Z. In the following we shall use Z to stand for $[Z]$ as a shorthand. Then, $\Phi(\mathbb{X})$ denotes the set of zones for $\mathcal{A}$. For zone Z, we define $Z^{\uparrow} = \{\nu + d \mid \nu \in Z, d \in \mathbb{R}_+\}$ and $Z[Y := 0] = \{\nu[Y := 0] \mid \nu \in Z\}$ where $\nu[Y := 0]$ denotes the valuation obtained from ν by resetting exactly the clocks in Y.

The symbolic semantics for timed automaton $\mathcal{A}$ is defined by the zone graph $\mathcal{S}(\mathcal{A}) = (\mathit{Conf}(\mathcal{A}), \to, c_0)$ where $\mathit{Conf}(\mathcal{A}) \subseteq L \times \Phi(\mathbb{X})$ is the set of configurations consisting of pairs of a location $\ell \in L$ and a zone $Z \in \Phi(\mathbb{X})$, $c_0 = \langle \ell_0, \{\nu_0\} \rangle$ is the initial configuration where $\nu_0(x) = 0$ for all clocks $x \in \mathbb{X}$ and $\to \subseteq \mathit{Conf}(\mathcal{A}) \times \mathit{Conf}(\mathcal{A})$ is the transition relation with delay transitions $\langle \ell, Z \rangle \to \langle \ell, Z^{\uparrow} \wedge I(\ell) \rangle$, and action transitions $\langle \ell, Z \rangle \to \langle \ell', (Z \wedge \varphi)[Y := 0] \wedge I(\ell') \rangle$ if there exists an edge $(\ell, \varphi, Y, \ell') \in E$.

The *quasi-equal* relation $\equiv$ for timed automaton $\mathcal{A}$ introduced in [11,13] is the relation containing all pairs of clocks for which in all computations of $\mathcal{A}$ their values are equal, except at points of time where they are reset and time is not allowed to elapse. Formally, it is defined as $\equiv \overset{\mathsf{def}}{=} \{(x, y) \mid x, y \in \mathbb{X} \text{ and } \langle \ell_0, \{\nu_0\} \rangle \to^* \langle \ell, Z \rangle \implies \forall \nu \in Z.\ \nu \models x = y \vee x = 0 \vee y = 0\}$. This notion is illustrated by the Example Section 2.

A *normalization operator* norm defined on zones is used to construct a finite representation of the transition relation $\to$. Maximal bound normalization [15,8], lower and upper bound zone based abstractions [3] and normalization using difference constraints [5] are some normalization procedures we may use to present our idea. Since our model for timed automata includes diagonal constraints we will define norm to be the normalization operator presented in [5]. We now formally define the norm operator. For timed automaton $\mathcal{A}$, let $\mathcal{G}$ be a finite set of difference constraints, and $k : \mathbb{X} \to \mathbb{Q}_0^+$ be a function mapping each clock x to the maximal constant $k(x)$ appearing in the guards or invariants in $\mathcal{A}$ containing x. For a real d let $\{d\}$ denote the fractional part of d and $\lfloor d \rfloor$ denote its integer part. Two valuations ν, ν' are equivalent, denoted $\nu \sim \nu'$ iff (1) for all x, either $\lfloor \nu(x) \rfloor = \lfloor \nu'(x) \rfloor$ or both $\nu(x) > k(x)$ and $\nu'(x) > k(x)$,(2) for all x, if $\nu(x) \leq k(x)$ then $\{\nu(x)\} = 0$ iff $\{\nu(x)\} = 0$ and (3) for all x, y if $\nu(x) \leq k(x)$ and $\nu(y) \leq k(y)$ then $\{\nu(x)\} \leq \{\nu(y)\}$ iff $\{\nu(x)\} \leq \{\nu'(x)\} \leq \{\nu'(y)\}$ (4) for all $\varphi \in \mathcal{G}$, $\nu \in \varphi$ iff $\nu' \in \varphi$. Then $\mathsf{norm}(Z) \overset{\mathsf{def}}{=} \{\nu \mid \nu \sim \nu', \nu' \in Z\}$.

4 Zero Time Behavior Abstraction

In this section, we present our method for detecting quasi-equal clocks. The main observation is that if two clocks x and y are quasi-equal then for all computations if one clock, say x is reset then a reset for the other clock y must appear in some future configuration in the computation path. In particular, for all the configurations in the computation fragment between the resets of x and y time cannot elapse (i.e. all the delay successors of a configuration have a delay of zero). Another key observation is that if in a configuration time is allowed to elapse, clocks x and y must be strictly equal. Therefore, to detect quasi-equal clocks. We do not only need to consider resets of clocks but also configurations in which time is allowed to elapse and configurations in which time is not allowed to elapse. The following definition formalizes our notion of zero time configurations, i.e. configurations for which time cannot elapse.

Definition 1 (Zero time configuration). *A configuration $\langle \ell, Z \rangle$ is* zero time *if the invariant of location ℓ precludes time to elapse. Formally,*

$$\mathsf{zt}(\ell, Z) \overset{\mathsf{def}}{=} \forall \nu \in Z, d \in \mathbb{R}_0^+.\ \nu + d \models I(\ell) \implies d = 0.$$

Our method preserves as much as possible the information corresponding to zero time configurations and abstracts away much information from the non-zero time configurations. If a configuration $\langle \ell, Z \rangle$ is zero time, our method will preserve all the information in Z. However, if the configuration $\langle \ell, Z \rangle$ is non-zero time, our method will abstract Z by means of the relax operator rlx to a much bigger zone $\mathsf{rlx}(Z)$, which preservers the strict equalities entailed by the zone Z. The following definition formalizes the relax operator.

Definition 2 (Relax operator). *Given a zone $Z \in \Phi(\mathbb{X})$. The* relax operator rlx *applied to the zone Z over-approximates Z by a conjunction of the clock equalities it entails. Formally,*

$$\mathsf{rlx}(Z) \overset{\mathsf{def}}{=} \bigwedge \{x = y \mid x, y \in \mathbb{X} \textit{ and } \forall \nu \in Z.\ \nu \models x = y\}.$$

As an example of the relax operator, consider the zone Z in Figure 3 left. It is the case that all the valuations in Z satisfy the constraint $x = y$. However, there are valuations in Z which satisfy $x \neq z$ and also valuations which satisfy $y \neq z$. Therefore, the result of applying the relax operator to Z yields the zone $x = y$ as shown in Figure 3 right. Note, that the zone $\mathsf{rlx}(Z)$ has no constraints on the unequal clocks. Another important property of $\mathsf{rlx}(Z)$ is that it contains only

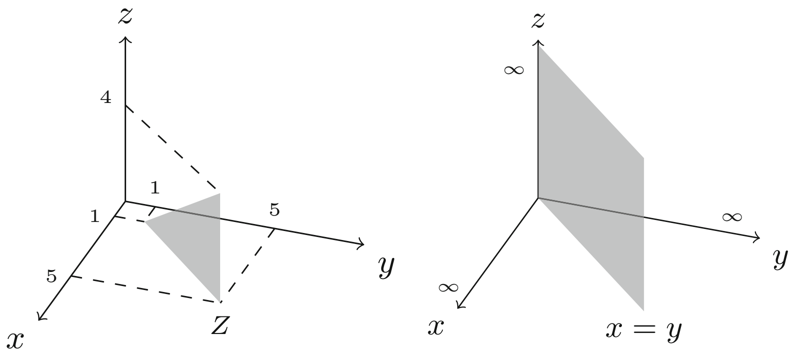


Fig. 3. The relax operator on zone $Z := x - y \leq 0 \wedge x - y \geq 0 \wedge x \geq 1 \wedge y \leq 5 \wedge y - z \leq 1$.

positive equalities and thus will remain unaffected by passage of time. That is $\mathsf{rlx}(Z) = \mathsf{rlx}(Z)^{\uparrow}$. Note, that $\mathsf{rlx}(Z)$ is much bigger than Z and it is closed with respect to delays. If a configuration is zero time our method will abstract the configuration by means of the normalization norm operator. On the contrary, if a configuration is non-zero time, our method will abstract the configuration by means of the relax rlx operator. The normalization operator norm is increasing, idempotent and yields a finite number of zones. Since our method relies on both the normalization operator norm and the relax operator rlx it is important that the relax operator exhibits the above mentioned properties.

Lemma 1. *The relax operator is increasing, idempotent and yields a finite abstraction. Formally, the following hold:*

– $Z \subseteq \mathsf{rlx}(Z)$ *for any* $Z \in \Phi(\mathbb{X})$
– $\mathsf{rlx}(\mathsf{rlx}(Z)) = \mathsf{rlx}(Z)$ *for any* $Z \in \Phi(\mathbb{X})$
– *the set* $\{\mathsf{rlx}(Z) \mid Z \in \Phi(\mathbb{X})\}$ *is finite.*

Our goal is to construct an abstract zone graph in which quasi-equalities can be soundly and efficiently computed. We now continue with the formal definition of our method.

Definition 3 (Abstract zone graph). *Timed automaton* $\mathcal{A}$ *induces the abstract zone graph* $\mathcal{S}^{\#}(\mathcal{A}) = (\mathit{Conf}(\mathcal{A}), \rightsquigarrow, c_0^{\#})$ *where:*

– $\mathit{Conf}(\mathcal{A}) \subseteq L \times \Phi(\mathbb{X})$ *is the set of configurations*
– $c_0^{\#} = (\ell_0, \bigwedge_{x,y \in \mathbb{X}} x = y)$ *is the initial configuration*
– $\rightsquigarrow \subseteq \mathit{Conf}(\mathcal{A}) \times \mathit{Conf}(\mathcal{A})$ *is the transition relation defined as:*

$$\langle \ell, P \rangle \rightsquigarrow \begin{cases} \langle \ell', \mathsf{norm}(F) \rangle & \text{if } \mathsf{zt}(\ell', F) \\ \langle \ell', \mathsf{rlx}(F) \rangle & \text{otherwise} \end{cases}$$

if there is an edge $(\ell, \varphi, Y, \ell') \in E$*, where* $F = (P \wedge \varphi \wedge I(\ell))[Y := 0] \wedge I(\ell')$.

The initial abstract configuration is a conjunction asserting all clocks to be equal. Given a configuration and an edge, the successor zone F is computed. If the destination configuration is zero time, the zone F will be normalized to $\mathsf{norm}(F)$. This is because in zero time it is important to preserve as much information as possible. This information is useful for detecting zero time paths in which quasi-equal clocks may arise. If the destination configuration is not zero time, the zone F will be relaxed to $\mathsf{rlx}(F)$. The zone $\mathsf{rlx}(F)$ consist of conjuncts asserting all strict equalities of clocks in zone F. In addition, the relax operator will remove inequalities introduced by clock resets. Note that the abstract zone graph only performs discrete transitions. This is because if a configuration is "detected" as non-zero time, the relax operator will be applied and the corresponding zone is closed with respect to delays. For easiness of presentation we use $\mathsf{norm}(F)$ to ensure the relation $\rightsquigarrow$ to be finite. We remind the reader that any normalization operator norm' such that $\mathsf{norm}'(Z) \supseteq Z$, $\mathsf{norm}'(\mathsf{norm}'(Z)) = \mathsf{norm}'(Z)$ and the set $\{\mathsf{norm}'(Z) \mid Z \in \Phi(\mathbb{X})\}$ is finite, will be suitable for our method.

For a timed automaton $\mathcal{A}$, we formalize the behavior of the corresponding abstract zone graph $\mathcal{S}^{\#}(\mathcal{A})$ with respect to the zone graph $\mathcal{S}(\mathcal{A})$ via a simulation relation.

Definition 4 (Simulation relation). *Given a timed automaton $\mathcal{A}$, a simulation relation for the zone graph $\mathcal{S}(\mathcal{A}) = (\mathit{Conf}(\mathcal{A}), \rightarrow, c_0)$ and the abstract zone graph $\mathcal{S}^{\#}(\mathcal{A}) = (\mathit{Conf}(\mathcal{A}), \rightsquigarrow, c_0^{\#})$, is a binary relation $\preccurlyeq$ on $\mathit{Conf}(\mathcal{A})$ such that:*

1. $c_0 \preccurlyeq c_0^{\#}$
2. *if $\langle \ell, Z \rangle \preccurlyeq \langle \ell_1, P \rangle$ then:*
   (a) *$\ell = \ell_1$ and $Z \subseteq P$*
   (b) *if $\langle \ell, Z \rangle \rightarrow \langle \ell', Z' \rangle$ with edge $(\ell, \varphi, \gamma, \ell') \in E$, then there exists $\langle \ell', P' \rangle$ such that $\langle \ell, P \rangle \rightsquigarrow \langle \ell', P' \rangle$ with edge $(\ell, \varphi, \gamma, \ell')$ and $\langle \ell', Z' \rangle \preccurlyeq \langle \ell', P' \rangle$*
   (c) *if $\langle \ell, Z \rangle \rightarrow \langle \ell, Z^{\uparrow} \wedge I(\ell) \rangle$, then $\langle \ell, Z^{\uparrow} \wedge I(\ell) \rangle \preccurlyeq \langle \ell, P \rangle$.*

If a simulation relation $\preccurlyeq$ exists, we say that the abstract zone graph $\mathcal{S}^{\#}(\mathcal{A})$ simulates the zone graph $\mathcal{S}(\mathcal{A})$.

In the rest of the paper we will consistently use Z and P to refer to zones in the zone and abstract zone graph respectively. The definition of simulation relation given above is quite specific. This allow us to better explain the relation between the zone graph and the abstract zone graph. If two configurations are in relation, the zone in the abstract zone graph is always bigger or equal than the corresponding zone in the zone graph. If there is a discrete transition in the zone graph then there is a discrete transition in the abstract zone graph as well. If there is a delay transition in the zone graph, meaning that the configuration is non-zero time, the abstract zone graph "remains" at its current configuration. In the latter case, since zones in the abstract zone graph are bigger or equal than zones in the zone graph, the corresponding abstract configuration is also non-zero time and closed with respect to delays. The following lemma guarantees the existence of a simulation relation.

Lemma 2. *For a timed automaton $\mathcal{A}$, the abstract zone graph $\mathcal{S}^{\#}(\mathcal{A})$ simulates the zone graph $\mathcal{S}(\mathcal{A})$.*

Our goal is to find the set of quasi-equal clocks for a given timed automaton $\mathcal{A}$. We now define the quasi-equal relation induced by the abstract zone graph $\mathcal{S}^{\#}(\mathcal{A})$ as the set of quasi-equalities implied by all the zones in the set of its reachable configurations.

Definition 5. *Given timed automaton $\mathcal{A}$. The abstract quasi-equal relation $\equiv^{\#}$ induced by the abstract zone graph $\mathcal{S}^{\#}(\mathcal{A})$, is the set of pairs of clocks such that for all pairs, their values in the reachable configurations are equal or one clock is equal to zero. Formally,*

$$\equiv^{\#} \overset{\mathsf{def}}{=} \{(x,y) \mid x,y \in \mathbb{X} \text{ and } c_0^{\#} \rightsquigarrow^{*} \langle \ell, P \rangle \implies P \subseteq \{\nu \mid \nu \models x = y \vee x = 0 \vee y = 0\}\}.$$

Our method is sound in the sense that if two clocks are quasi-equal in the abstract zone graph, then they are quasi-equal in the concrete zone graph. Our method is not complete in the sense that if two clocks are quasi-equal in the zone graph, then they might not be quasi-equal in the abstract zone graph. As an example consider the timed automaton $\mathcal{A}' = (L, \mathbb{X}, I, E, \ell_0)$ with $L = \{\ell_0, \ell_1, \ell_2\}$, $\mathbb{X} = \{x, y\}$, $I(\ell) = \top$ for all $\ell \in L$ and $E = \{(\ell_0, x \geq 5, \{\}, \ell_1), (\ell_1, x \leq 2, \{x\}, \ell_2)\}$. Then $x \equiv y$ but $x \not\equiv^{\#} y$. This is because the edge $e = (\ell_1, x \leq 2, \{x\}, \ell_2)$ is an unfeasible edge, i.e. it does not induce a reachable transition in the zone graph. Edge e will cause clocks x, y to be unequal in the abstract zone graph. Surprisingly, we have not been able to find an example for which the abstraction is not complete given that the considered timed automaton contains only feasible edges. The following theorem states formally the soundness of the abstraction.

Theorem 1 (Zero time abstraction is sound). *Given timed automaton $\mathcal{A}$. If two clocks are quasi-equal in the abstract zone graph $\mathcal{S}^{\#}(\mathcal{A})$, then they are quasi-equal in the zone graph $\mathcal{S}(\mathcal{A})$. Formally,*

$$\forall x, y \in \mathbb{X}.\ x \equiv^{\#} y \implies x \equiv y.$$

For implementation purposes, it is important that the relation $\rightsquigarrow$ is finite. Given a timed automaton $\mathcal{A}$ and by definition of the transition relation $\rightsquigarrow$. A configuration has two possible successors. Either a successor corresponding to the application of the norm operator, or a successor corresponding to the application of the rlx operator. The set of zones generated by norm is finite. Further, the set of zones generated by rlx is in $\mathcal{O}(2^{|\mathbb{X}|})$, since it contains only positive equalities for the clocks in $\mathbb{X}$. Thus, we obtain the following theorem.

Theorem 2. *The transition relation $\rightsquigarrow$ is finite.*

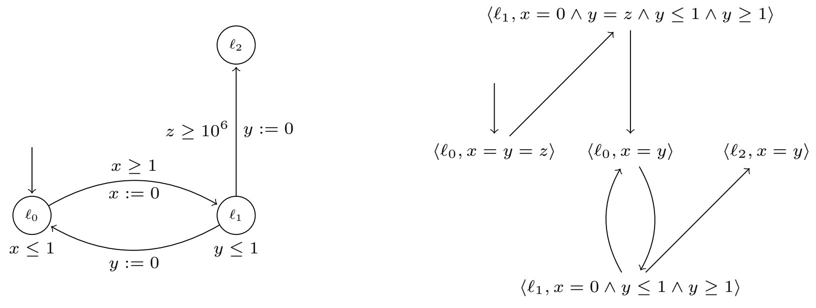

ℓ_2
$z \geq 10^6$
$y := 0$
$x \geq 1$
$x := 0$
ℓ_0
ℓ_1
$x \leq 1$
$y \leq 1$
$y := 0$
$\langle \ell_1, x = 0 \wedge y = z \wedge y \leq 1 \wedge y \geq 1 \rangle$
$\langle \ell_0, x = y = z \rangle$
$\langle \ell_0, x = y \rangle$
$\langle \ell_2, x = y \rangle$
$\langle \ell_1, x = 0 \wedge y \leq 1 \wedge y \geq 1 \rangle$

Fig. 4. Left: a timed automaton with clocks, x, y and z. Clocks x, y are quasi-equal. Right: the corresponding abstract zone graph.

5 Size of the Abstract Zone Graph

In this section we characterize the size of the abstract zone graph based on the number of reachable configurations. We show that when an automaton $\mathcal{A}$ contains only feasible edges i.e. for each edge e in the timed automaton there exists a computation path in which the edge e induces a transition, then the size of the zone graph $\mathcal{S}(\mathcal{A})$ is an upper bound for the size of the abstract zone graph $\mathcal{S}^{\#}(\mathcal{A})$.

Definition 6 (Size). *For timed automaton $\mathcal{A}$ we defined the size of the zone graph $\mathcal{S}(\mathcal{A}) = (Conf(\mathcal{A}), \rightarrow, c_0)$ and the size of the abstract zone graph $\mathcal{S}^{\#}(\mathcal{A}) = (Conf(\mathcal{A}), \rightsquigarrow, c_0^{\#})$ to be the number of reachable configurations. Formally, $|\mathcal{S}(\mathcal{A})| = |\{c \mid c_0 \rightarrow^* c\}|$ and $|\mathcal{S}^{\#}(\mathcal{A})| = |\{c \mid c_0^{\#} \rightsquigarrow^* c\}|$ respectively.*

The following theorem shows that when in the input automaton all edges induce a reachable configuration in the zone graph $\mathcal{S}(\mathcal{A})$. Then the number of configurations from the corresponding abstract zone graph $\mathcal{S}^{\#}(\mathcal{A})$ is smaller or equal than the number of configurations in the zone graph $\mathcal{S}(\mathcal{A})$. The theorem follows from the following facts. First, if the abstract zone graph $\mathcal{S}^{\#}(\mathcal{A})$ performs an action transition with edge e then the zone graph $\mathcal{S}(\mathcal{A})$ also performs a transition with edge e. Second, by Lemma 2 it is the case that $\mathcal{S}^{\#}(\mathcal{A})$ simulates $\mathcal{S}(\mathcal{A})$ and all zones in the configurations of $\mathcal{S}^{\#}(\mathcal{A})$ are bigger than the corresponding ones in $\mathcal{S}(\mathcal{A})$.

Theorem 3 ($|\mathcal{S}^{\#}(\mathcal{A})| \leq |\mathcal{S}(\mathcal{A})|$). *Given a timed automaton $\mathcal{A}$, the zone graph $\mathcal{S}(\mathcal{A}) = (Conf(\mathcal{A}), \rightarrow, c_0)$ and the abstract zone graph $\mathcal{S}^{\#}(\mathcal{A}) = (Conf(\mathcal{A}), \rightsquigarrow, c_0^{\#})$. If $\mathcal{A}$ is such that for all $e = (\ell, \varphi, Y, \ell') \in E$ there is a transition $\langle \ell, Z \rangle \rightarrow \langle \ell', Z' \rangle$ with edge e and $c_0 \rightarrow^* \langle \ell, Z \rangle$ then $|\mathcal{S}^{\#}(\mathcal{A})| \leq |\mathcal{S}(\mathcal{A})|$.*

The feasibility of edges is not a strong condition, since in practice unfeasible edges will not occur intentionally, except in the cases when the modeler makes a mistake. Theorem 3 gives an upper bound on the size of the abstract zone graph. In practice for a timed automaton the difference on the size of the abstract and

Algorithm 1. High level algorithm for detecting quasi-equal clocks

```
Input: timed automaton $\mathcal{A} = (L, \mathbb{X}, I, E, \ell_0)$
Output: a binary relation $QE \subseteq \mathbb{X} \times \mathbb{X}$ containing quasi-equal clocks in $\mathcal{A}$
 1: $W := \{c_0^\#\}$, $V := \emptyset$
 2: $QE := \{(x, y) \mid x, y \in \mathbb{X}$ and $x$ is different than $y\}$
 3: while $W \neq \emptyset$ and $QE \neq \emptyset$ do
 4:     take $\langle \ell, P \rangle$ from $W$
 5:     for all $(x, y) \in QE$ do
 6:         if $P \not\subseteq \{\nu \mid \nu \models x = y \vee x = 0 \vee y = 0\}$ then
 7:             $QE := QE \setminus \{(x, y)\}$
 8:         end if
 9:     end for
10:     if $P \not\subseteq P'$ for all $\langle \ell, P' \rangle \in V$ then
11:         add $\langle \ell, P \rangle$ to $V$
12:         for all $\langle \ell', P' \rangle$ with $\langle \ell, P \rangle \rightsquigarrow \langle \ell', P' \rangle$ do
13:             add $\langle \ell', P' \rangle$ to $W$
14:         end for
15:     end if
16: end while
17: return $QE$
```

the size of the concrete system can be exponential. To illustrate this, consider the automaton $\mathcal{A}$ in Figure 4 left. We observe that the zero time behavior occurs at points of time where $x = 1 \wedge y = 1 \wedge z = n$ with $n \in \{1, 2, \ldots, 10^6\}$. The normalized zone graph using maximal constant over approximation will contain at least 10^6 configurations. In Figure 4 right the complete full abstract zone graph $\mathcal{S}^\#(\mathcal{A})$ is illustrated. At point of time $x = 1 \wedge y = 1 \wedge z = 1$ our method detects a zero time configuration and computes the exact successor zone $Z_1 := x = 0 \wedge y = z \wedge y \leq 1 \wedge y \geq 1$. Note that $\mathsf{zt}(\ell_1, Z_1)$ is valid, then there is a transition with edge $(\ell_1, \top, \{y\}, \ell_0)$. Our method computes $F := x = 0 \wedge y = 0 \wedge z = 1$ and the formula $\mathsf{zt}(\ell_0, F)$ is not valid, thus F is relax to $\mathsf{rlx}(F) := x = y$ leading to configuration $\langle \ell_0, x = y \rangle$. Since F does not imply a quasi-equality for clock z, the clock z is abstracted away. The resulting abstraction has only 5 configurations.

6 Algorithm and Experiments

In this section, first we present a high level algorithm for performing a reachability analysis on the abstract zone graph induced by a given timed automaton. Next, we give some details on our implementation and compare the results obtained by our implementation to the ones obtained by using a model checker.

6.1 Algorithm for Detecting Quasi-equal Clocks

Algorithm 1 lists a high level algorithm for finding quasi-equal clocks in a given timed automaton. The idea of Algorithm 1 is to traverse the reachable state space of $\mathcal{S}^{\#}(\mathcal{A})$ while maintaining a relation QE containing quasi-equal clocks. The reachable state space is computed on the fly.

We continue by describing Algorithm 1 in detail. At line 2 the set QE is initialized to have all non reflexive quasi-equalities. At line 3 the while condition ensures that the algorithm will terminate either when we have visited the whole reachable space of $\mathcal{S}^{\#}(\mathcal{A})$ or when there are no quasi-equal clocks in QE. At lines 4 to 8 the algorithm picks a configuration $\langle \ell, P \rangle$ to be explored and checks that all the quasi-equalities (x, y) in QE are implied by P. If this is not the case, then it will remove (x, y) from QE. Note that in the algorithm the size of QE only decreases. Finally, at lines 10 to 14 the algorithm computes the successor of $\langle \ell, P \rangle$ using the transition relation $\rightsquigarrow$.

Note, that all the operations needed in Algorithm 1 can be implemented using difference bound matrices [4,10] and thus our approach can be implemented in tools like Kronos [18] or Uppaal [12]. The check in line 6 for two clocks can be implemented by checking independently whether P satisfies any disjunct in $x = y \vee x = 0 \vee y = 0$. For computing the successor in line 12 by definition of $\rightsquigarrow$ it is necessary to compute $\mathsf{zt}(\ell', F)$ where F is a convex zone. This can be computed by checking the inclusion $F \wedge I(\ell') \supseteq F^{\uparrow} \wedge I(\ell')$.

If the set QE is never empty, Algorithm 1 computes the reachable space of $\mathcal{S}^{\#}(\mathcal{A})$ and if a pair of clocks (x, y) are in QE by Theorem 1 it follows that they are quasi-equal in $\mathcal{A}$.

Theorem 4 (Partial correctness). *For any timed automaton $\mathcal{A}$, if two clocks are in relation QE from Algorithm 1, then they are quasi-equal in the corresponding zone graph. Formally,*

$$\forall x, y \in \mathbb{X}. (x, y) \in QE \implies x \equiv y.$$

Algorithm 1 will terminate whenever the relation QE is empty or whenever the wait list W is empty. By Theorem 2, the relation $\rightsquigarrow$ is finite, meaning that the list W will be eventually empty. Thus we obtain the following theorem.

Theorem 5 (Termination). *Algorithm 1 terminates for all inputs.*

6.2 Experiments

As a proof of concept we have implemented our approach in our prototype tool *Saset*. Saset is implemented in the Jahob [19,16] verification system. Our implementation represents zones as linear real arithmetic formulae and uses logical implications to ensure a finite number of representatives. As Saset constructs the abstract zone graph a number of constraints will arise, Saset will use an SMT solver to solve them. In our experiments we used the solver Z3 [9].

In general our results show that the size of the transition system computed using our abstraction is very small in comparison to the one computed by Uppaal. Therefore, the verification times for Saset are fast in spite of the number of SMT calls which are time costly.

Table 1. Results for detecting quasi-equal clocks using tools Saset and Uppaal. Saset returns the set of quasi-equal clocks whereas Uppaal performs a single query asserting clocks to be quasi-equal. Note, that for detecting quasi-equal clocks multiple queries are needed. The zero time behavior for automata in classA remains constant, in classB grows linearly and in classC grows exponentially.

Automaton	clocks	qe-clocks	max k	Saset			Uppaal		
				SMT-calls	states	t (s)	Q	states	t (s)
classA1	3	2	10^4	26	5	0.3	φ_1	20k	7.3
classA2	3	2	10^5	26	5	0.3	φ_1	200k	1200
classA3	3	2	10^6	26	5	0.3	φ_1	t.o.	t.o.
classB2	4	2	10^4	72	8	0.8	φ_1	20k	7.4
classB3	5	3	10^4	105	10	1.3	φ_2	30k	12.5
classB4	6	4	10^4	144	12	2.2	φ_3	40k	21.0
classB5	7	5	10^4	193	14	3.5	φ_4	50k	34.8
classB6	8	6	10^4	248	16	5.16	φ_5	60k	45.2
classC2	3	2	5000	63	7	1.2	φ_1	5k	5.2
classC3	4	3	5000	237	14	8.3	φ_2	35k	31.8
classC4	5	4	5000	809	26	44.05	φ_3	75k	202.3
classC5	6	5	5000	2389	47	195.3	φ_3	150k	1007.3
classC6	7	6	5000	8515	85	844	φ_4	t.o.	t.o.

In Table 1, we present a number of results obtained by using our tool and the model checker Uppaal [12]. Our intention is not to outperform Uppaal in terms of time but to show that the abstraction method that we propose for detecting quasi-equalities is a good abstraction. Thus, we encourage the reader to focus on the number of states generated by the tools for each automaton. Note, that for Uppaal to detect n clocks to be quasi-equal it would need to perform 2^n queries whereas our tool compute them directly. In Table 1, max k is the number of the maximal constant appearing in the corresponding timed automaton and the queries φ are TCTL formulae asserting a number of clocks to be quasi-equal, e.g. $\varphi_1 := \mathsf{AG}\ x_0 = x_1 \vee x_0 = 0 \vee x_1 = 0$, $\varphi_2 := \mathsf{AG}\ (x_0 = x_1 \vee x_0 = 0 \vee x_1 = 0) \wedge (x_0 = x_2 \vee x_0 = 0 \vee x_2 = 0)$. The experiments were executed in a AMD Phenom II X6 3.2Ghz Processor with 8GB RAM running Linux 3.2.

We use three classes of timed automata which are relevant for our abstraction. For classA the zero time behavior is constant and the non zero time behavior grows. For classB the zero time behavior grows linear and the non-zero time behavior remains constant. For classC, the zero time behavior grows exponentially.

The automata in classA correspond to the automaton in Figure 4. We only change the maximal constant appearing in the automaton and observe that the size of the abstraction remains the same. For the automata in classB the number of quasi-equal clocks is increased but there is an order on the reset of the quasi-equal clocks. We observe a linear increase in the number of states for both tools. For the automata in classC the number of quasi-equal clocks is increased

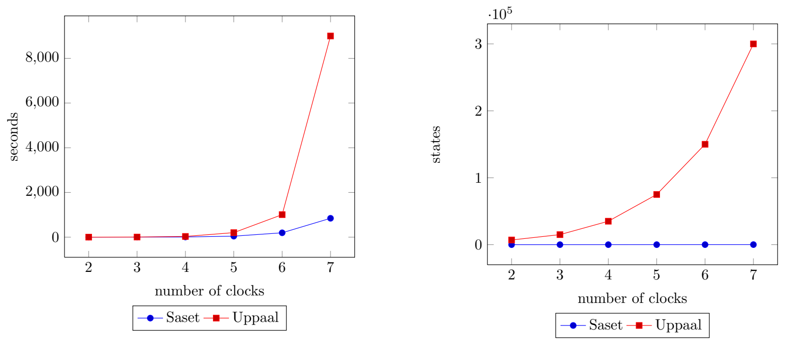


Fig. 5. Verification times and number of states explored for automata in classC

but there is no order in the reset of the clocks. We observe an exponential increase in the number of states for both tools. In Table 1, the construction of the abstraction for automaton classC6 required 8515 SMT calls, according to our tool these SMT calls took 695.2 seconds, which is 82% of the time cost. Thus, a more efficient implementation is desirable. Since our algorithm can be implemented using difference bound matrices a much efficient implementation is possible. Figure 5 illustrates the results for automata in classC.

Once the quasi-equal clocks in a timed automaton have been detected. A reduction on the number of clocks might take place, leading to a major speed up. As an example we have reduced all the quasi-equal clocks for the automaton classC6 by replacing them with a representative clock. The resulting zone graph computed by Uppaal consists of 5003 states and invariant properties can be verified in few seconds, which is an exponential gain.

7 Conclusion

In this paper, we have presented an abstraction that is effective for the goal of an already established optimization technique which is based on quasi-equal clocks. The abstraction is motivated by an intuition about the way quasi-equalities can be tracked. We have implemented the corresponding reasoning method in the Jahob framework using an SMT solver. Our experiments indicate that our intuition may lead to a useful abstraction. I.e., the is coarse enough to yield a drastic reduction of the size of the zone graph. Still, it is precise enough to identify a large class of quasi-clocks.

The goal of our prototypical implementation is to be able to evaluate the effectiveness of our abstraction for the goal of the optimization. An orthogonal issue is the optimization of the execution time of the abstraction method itself. In our experiments, the execution time is acceptable. Possibly the execution time can be improved by exchanging the general-purpose SMT solver with a specialized machinery, i.e., difference bound matrices [4,10]. We leave this aspect to future work.

References

1. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science 126(2), 183–235 (1994)
2. Behrmann, G., Bouyer, P., Larsen, K.G., Pelánek, R.: Lower and upper bounds in zone based abstractions of timed automata. In: Jensen, K., Podelski, A. (eds.) TACAS 2004. LNCS, vol. 2988, pp. 312–326. Springer, Heidelberg (2004)
3. Behrmann, G., Bouyer, P., Larsen, K.G., Pelánek, R.: Lower and upper bounds in zone-based abstractions of timed automata. International Journal on Software Tools for Technology Transfer 8(3), 204–215 (2006)
4. Bellman, R., Kalaba, R.E.: Dynamic programming and modern control theory. Academic Press, New York (1965)
5. Bengtsson, J., Yi, W.: On clock difference constraints and termination in reachability analysis of timed automata. In: Dong, J.S., Woodcock, J. (eds.) ICFEM 2003. LNCS, vol. 2885, pp. 491–503. Springer, Heidelberg (2003)
6. Bengtsson, J., Yi, W.: Timed automata: Semantics, algorithms and tools. In: Desel, J., Reisig, W., Rozenberg, G. (eds.) ACPN 2003. LNCS, vol. 3098, pp. 87–124. Springer, Heidelberg (2004)
7. Daws, C., Yovine, S.: Reducing the number of clock variables of timed automata. In: Proc. RTSS 1996, pp. 73–81. IEEE Computer Society Press (1996)
8. Daws, C., Tripakis, S.: Model checking of real-time reachability properties using abstractions. In: Steffen, B. (ed.) TACAS 1998. LNCS, vol. 1384, pp. 313–329. Springer, Heidelberg (1998)
9. de Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008)
10. Dill, D.L.: Timing assumptions and verification of finite-state concurrent systems. In: Sifakis, J. (ed.) CAV 1989. LNCS, vol. 407, pp. 197–212. Springer, Heidelberg (1990)
11. Herrera, C., Westphal, B., Feo-Arenis, S., Muñiz, M., Podelski, A.: Reducing quasi-equal clocks in networks of timed automata. In: Jurdziński, M., Ničković, D. (eds.) FORMATS 2012. LNCS, vol. 7595, pp. 155–170. Springer, Heidelberg (2012)
12. Larsen, K.G., Pettersson, P., Yi, W.: Uppaal in a nutshell. International Journal on Software Tools for Technology Transfer (STTT) 1(1), 134–152 (1997)
13. Muñiz, M., Westphal, B., Podelski, A.: Timed automata with disjoint activity. In: Jurdziński, M., Ničković, D. (eds.) FORMATS 2012. LNCS, vol. 7595, pp. 188–203. Springer, Heidelberg (2012)
14. Olderog, E.R., Dierks, H.: Real-Time Systems - Formal Specification and Automatic Verification. Cambridge University Press (2008)
15. Pettersson, P.: Modelling and verification of real-time systems using timed automata: theory and practice. Ph.D. thesis, Citeseer (1999)
16. Podelski, A., Wies, T.: Counterexample-guided focus. In: Proceedings of the 37th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2010, pp. 249–260. ACM, New York (2010), `http://doi.acm.org/10.1145/1706299.1706330`
17. Wies, T., Muñiz, M., Kuncak, V.: An efficient decision procedure for imperative tree data structures. In: Bjørner, N., Sofronie-Stokkermans, V. (eds.) CADE 2011. LNCS, vol. 6803, pp. 476–491. Springer, Heidelberg (2011)
18. Yovine, S.: Kronos: A verification tool for real-time systems. International Journal on Software Tools for Technology Transfer (STTT) 1(1), 123–133 (1997)
19. Zee, K., Kuncak, V., Rinard, M.: Full functional verification of linked data structures. In: Proceedings of the 2008 ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2008, pp. 349–361. ACM, New York (2008), `http://doi.acm.org/10.1145/1375581.1375624`

On the Verification of Timed Discrete-Event Models[⋆]

Christos Stergiou, Stavros Tripakis, Eleftherios Matsikoudis, and Edward A. Lee

University of California, Berkeley

Abstract. Timed discrete-event (DE) is an actor-oriented formalism for modeling timed systems. A DE model is a network of actors consuming/producing timed events from/to a set of input/output channels. In this paper we study a basic DE model, called deterministic DE (DDE), where actors are simple constant-delay components, and two extensions of DDE: NDE, where actors are non-deterministic delays, and DETA, where actors are either deterministic delays or timed automata. We investigate verification questions on DE models and examine expressiveness relationships between the DE models and timed automata.

1 Introduction

Timed automata, introduced by Alur and Dill [2,7] are one of the prominent formalisms for timed systems. In this paper we study another formalism for modeling dense-time systems, which we call *timed discrete-event* (DE). DE is inspired by the DE domain of the tool Ptolemy [9], which is itself inspired by established fields such as discrete-event simulation and queueing models. More recently, DE has been proposed as a programming abstraction for cyber-physical systems and used in the Ptides framework [8] for the design of distributed real-time systems, e.g. control systems for electric power stations and correct shutdown mechanisms for industrial controllers.

DE is an *actor-oriented* formalism, in the sense of Agha [1], where a model consists of a collection of *actors*, each consuming and producing messages from input and to output channels. In the case of DE a model is a network of actors. Ptolemy DE models can be hierarchical, but for the purposes of this paper we only consider flat models. We also abstract away message values, thereby leaving actors to communicate only via *timed events*. A timed event is characterized by a single value, its timestamp.

⋆ This work was supported in part by TerraSwarm, one of six centers of STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA. It was also supported in part by the Center for Hybrid and Embedded Software Systems (CHESS) at UC Berkeley (supported by NSF awards #0720882 (CSR-EHS: PRET) and #0931843 (ActionWebs), the Naval Research Laboratory (NRL #N0013-12-1-G015), and the companies: Bosch, National Instruments, and Toyota), and the NSF Expeditions in Computing project *ExCAPE: Expeditions in Computer Augmented Program Engineering.*

Even though discrete-event simulation is widespread in system design, including, for instance, hardware system simulation methods based on languages such as Verilog, VHDL, and SystemC, the exhaustive verification of DE models has received little attention, to our knowledge. In this paper we take a step towards remedying this, by studying three versions of DE models in terms of expressiveness and model-checking.

First, we introduce a basic, *deterministic DE* (DDE) model, where actors are simple constant (and known) delays. An actor in DDE delays every input event by a constant delay Δ, which means that if the input event has timestamp τ then the actor produces a corresponding output event with timestamp $\tau + \Delta$. A constant delay actor cannot be represented by an equivalent timed automaton, as the latter would need an unbounded number of clocks, one for every input event that may arrive within an interval Δ.

Nevertheless, we can show that the strong deterministic properties of DDE allow its state space to be reduced to a finite *lasso*. The latter can be used for exhaustive model-checking of both *signal* and *state* queries. An example of a signal query is "*is there an execution where an event with timestamp* > 10 *occurs in channel c.*" An example of a state query is "*is there a reachable state where channels* c_1 *and* c_2 *contain two events with timestamps* τ_1, τ_2*, such that* $|\tau_1 - \tau_2| \leq 2$." The lasso can also be used to show that every DDE model can be transformed to an equivalent timed automaton (TA) model.

We also introduce two extensions to the basic DDE model: *non-deterministic DE* (NDE) and *DE with timed automata* (DETA). In NDE, actors are non-deterministic delays, specified by an interval, say, $[l, u]$, so that an input event is delayed by some arbitrary $\delta \in [l, u]$. In DETA, actors are either constant delays, or timed automata. A timed automaton M can be viewed as an actor which reacts to input events arriving on a given channel c by taking a discrete transition labeled with input c (we require that M be *receptive*, that is, always be able to accept any input). M can spontaneously choose to generate an output event on a given channel c' by taking a discrete transition labeled with c'.

Finally, we discuss expressiveness of the above models. We show that DDE $\subset$ NDE and DDE $\subset$ TA $\subset$ DETA, where all inclusions are strict. We also show that NDE $\not\subseteq$ TA, and conjecture that TA $\not\subseteq$ NDE and NDE $\not\subseteq$ DETA.

2 Deterministic Timed Discrete-Event Models

We abstract away event values, so events are only timestamped tokens. Formally, an event is represented by a timestamp $\tau \in \mathbb{R}_{\geq 0}$, where $\mathbb{R}_{\geq 0}$ is the set of non-negative reals. The set of naturals is denoted $\mathbb{N} = \{0, 1, 2, ...\}$.

2.1 Syntax

A DDE model is a finite labeled directed graph $G = (A, C, D)$ such that

- A is the set of nodes of G. Each node is called a *DE actor* or *actor* in short.
- $C \subseteq A \times A$ is the set of edges of G. Each edge $c \in C$ is called a *channel*.

– $D : A \to \mathbb{N}$ is a (total) function mapping each actor $a \in A$ to a non-negative integer number called the *delay* of a. Note that $D(a)$ may be 0.

Let $c = (a, b) \in C$. Then c is an *output* channel of a and an *input* channel of b. We use $C^{in}(a)$ and $C^{out}(a)$ to denote the sets of input and output channels of an actor a, respectively. Let $C(a) = C^{in}(a) \cup C^{out}(a)$. By definition, G is a *closed* model, in the sense that all input channels are connected. In fact, every channel has a unique writer and a unique reader. An actor without input (respectively, output) channels is called a *source* (respectively, *sink*).

An example of a DDE model is given Figure 1. The model has three actors, a_1, a_2, a_3, with delays $1, 1, 0$, respectively, and four channels (the four arrows).

A *channel state* for a DE model G is a total function $r : C \to 2^{\mathbb{R}_{\geq 0}}$ which maps every channel $c \in C$ to a finite set of events initially pending on c. In Figure 1, the bullets annotating channels c_1, c_2 specify an *initial channel state.* In this case there are two initial events, both with timestamp 3.

Partial order: Our model allows cyclic graphs and zero-delay actors. However, we require that every cycle visits at least one actor a such that $D(a) > 0$. This condition effectively allows to "break" zero-delay loops, and to define a partial order $\prec$ on the set of actors A, so that $a \prec a'$ iff there exists a path from a to a' such that for any actor a'' in the path (including a but excluding a') we have $D(a'') = 0$. The order $\prec$ is essential for ensuring that actors are fired in timestamp order (Lemma 1) which in turn yields important deterministic properties of the DDE model. For the example of Figure 1, the order $\prec$ is $a_3 \prec a_2$.

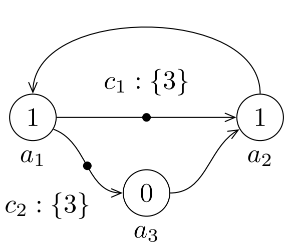


Fig. 1. A DDE model

2.2 Operational Semantics

To a given DDE model G and initial channel state r_0, we will associate a *timed transition system* $TTS(G, r_0) = (S, s_0, \to)$, where S is its set of states, $s_0 = (r_0, 0)$ is its (unique) initial state, and $\to \subseteq S \times A \times S$ is its transition relation, defined below. A state $s \in S$ is a pair (r, t), where r is a channel state and $t \in \mathbb{R}_{\geq 0}$ is a *global timestamp.* The initial global timestamp is 0.

Given a channel state $r : C \to 2^{\mathbb{R}_{\geq 0}}$, let $\tau_{\min}(r) = \min \bigcup_{c \in C} r(c)$. That is, $\tau_{\min}(r)$ is the minimum timestamp among all currently pending events in r. Given actor $a \in A$ and r, we denote by $\tau_{\min}(a, r)$ the minimum timestamp among all currently pending events in the input channel(s) of a at r. That is, $\tau_{\min}(a, r) = \min \bigcup_{c \in C^{in}(a)} r(c)$. Note that $\tau_{\min}(a, r) \geq \tau_{\min}(r)$ for any a, r. By convention we set $\min \emptyset = \infty$. This implies that for an empty channel state r, we have $\tau_{\min}(r) = \infty$. Also, if $C^{in}(a) = \emptyset$, that is, if a has no input channels, then $\tau_{\min}(a, r) = \infty$ for all r.

We say that an actor $a \in A$ is *enabled* at state $s = (r, t)$, denoted *enabled*(a, s), if $\tau_{\min}(a, r) = \tau_{\min}(r) = t$. That is, a is enabled at s if there is at least one event pending in one of the inputs of a which has timestamp τ no greater than the smallest timestamp in r and τ agrees with the global time t. We say that a is *strongly enabled* at s if $\mathit{enabled}(a,s)$ and there is no actor $b \neq a$ such that $\mathit{enabled}(b,s)$ and $b \prec a$. That is, a is strongly enabled at s if it is enabled and there is no actor b which is also enabled at s and which comes before a according to $\prec$.

We next define the operation of *firing* an actor, and the effect that this has on the state. Intuitively, firing an actor a at state $s = (r,t)$ consists in removing the event $\tau_{\min}(a,r)$ from all input channels of a that contain this event, and adding the event $\tau_{\min}(a,r) + D(a)$ to each output channel of a. Formally, we define an auxiliary function $f(a,r,d)$ which, given actor $a \in A$, channel state r, and delay $d \in \mathbb{R}_{\geq 0}$, returns a channel state r' defined as follows:

$$r'(c) = \begin{cases} r(c) - \{\tau_{\min}(a,r)\} & \text{if } c \in C^{in}(a) \\ r(c) \cup \{\tau_{\min}(a,r) + d\} & \text{if } c \in C^{out}(a) \\ r(c) & \text{otherwise.} \end{cases} \tag{1}$$

We are now ready to define the transition relation $\rightarrow$ of $\mathit{TTS}(G, r_0)$. $\rightarrow$ has two types of transitions: *discrete transitions* of the form $s \xrightarrow{a} s'$, such that a is strongly-enabled in state $s = (r,t)$ and $s' = (r',t)$ with $r' = f(a,r,D(a))$, and *timed transitions* of the form $s \xrightarrow{\delta} s'$, where $s, s' \in S$, $a \in A$, and $\delta \in \mathbb{R}_{\geq 0}$. Timed transitions are enabled at a state $s = (r,t)$ when no discrete transition is enabled at s and when r is not empty. In that case, it must be $t < \tau_{\min}(r)$ and $\tau_{\min}(r) \neq \infty$. Then, a timed transition $s \xrightarrow{\delta} (r,t')$ occurs, with $\delta = \tau_{\min}(r) - t$ and $t' = t + \delta = \tau_{\min}(r)$.

Remarks: (1) Global time is not affected by a discrete transition and channel state is not affected by a timed transition. (2) In $\mathit{TTS}(G, r_0)$ there cannot be two timed transitions in a row. (3) According to the rule $\tau_{\min}(a,r) = \infty$, source actors are never enabled, and therefore never fire. We use source actors simply to allow for initial events at the inputs of some actors.

A state $s = (r,t)$ is a *deadlock*, that is, has no outgoing transitions, iff r is empty, that is, for every $c \in C$, $r(c) = \emptyset$.

An *execution* of $\mathit{TTS}(G, r_0)$ is a sequence of states $\rho = s_0, s_1, ...$ such that there is a (discrete or timed) transition from every s_i to s_{i+1}. We require ρ to be *maximal*, that is, either infinite or ending in a deadlock state. Note that if the DDE model contains a loop that is reachable from some initial event, there will not be a deadlock state.

Lemma 1. *Let $\rho = s_0, s_1, ...$ be an execution of $\mathit{TTS}(G, r_0)$ and let a be an actor of G. For any transitions $s_i \xrightarrow{a} s_{i+1}$ and $s_j \xrightarrow{a} s_{j+1}$ in ρ such that $s_i = (r_i, t_i)$, $s_j = (r_j, t_j)$, and $i < j$, we have $t_i < t_j$.*

Lemma 1 states that every actor is fired in timestamp order, and in particular, that it cannot be fired more than once before time elapses.

$\mathit{TTS}(G, r_0)$ has several deterministic properties. First, by definition, if $s \xrightarrow{a} s'$ then there is no $s'' \neq s'$ such that $s \xrightarrow{a} s''$. Second, $\mathit{TTS}(G, r_0)$ has the so-called "diamond property":

Lemma 2. *If $s \xrightarrow{a} s_1$ and $s \xrightarrow{b} s_2$, then there is a unique s' such that $s_1 \xrightarrow{b} s'$ and $s_2 \xrightarrow{a} s'$.*

Let $\rho = s_0, s_1, \ldots$ be an execution of $TTS(G, r_0)$ where $G = (A, C, D)$ and $s_i = (r_i, t_i)$. The *signal* of a channel $c \in C$ under execution ρ, denoted σ_c^ρ, is defined to be the set of all events occurring in c along the entire execution: $\sigma_c^\rho = \bigcup_{i \in \mathbb{N}} r_i(c)$.

Lemma 3 (Kahn property [12]). *For any $c \in C$ and any two executions ρ_1 and ρ_2, $\sigma_c^{\rho_1} = \sigma_c^{\rho_2}$.*

Because of the Kahn property, we can write σ_c for the unique signal of a channel c. This can be viewed as the denotational semantics of DDE models.

3 Boundedness of DDE

In this section we study boundedness of the state-space of DDE models. Let us begin with an illustrative example. Figure 2 shows a DDE model $G = (A, C, D)$, with $A = \{a_1, a_2\}$, $C = \{c_1 : (a_1, a_1), c_2 : (a_1, a_2)\}$, $D(a_1) = P$, and $D(a_2) = 0$. This model captures a periodic source with period P, generating events at times $P, 2P, \cdots$. The model includes actor a_1 which delays its input by P and a sink actor a_2. If the delay of an actor is non-zero, it is drawn inside the actor. Zero delays are not drawn. The model has two channels, c_1, c_2. Channel c_1, a self-loop of a_1, contains an initial event with timestamp 0. This is the only initial event in the system (empty initial event sets on channels are not drawn). The initial event models the seed of the periodic source. Actor a_1 adds a delay of P to the event's timestamp, outputs the event to c_2, which represents the source's output, and starts anew a cycle where the initial event is replaced with an event with timestamp P.

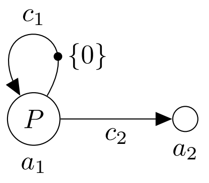


Fig. 2. Periodic clock

The initial channel state is $r_0 = \{(c_1, \{0\}), (c_2, \{\})\}$. A prefix of a path in the transition system $TTS(G, r_0)$ is the following:

$$
\begin{array}{llll}
s_0 : (c_1 : \{0\}, c_2 : \{\}, t = 0) & \xrightarrow{a_1} s_1 : (c_1 : \{P\}, c_2 : \{P\}, t = 0) & \xrightarrow{P} \\
s_2 : (c_1 : \{P\}, c_2 : \{P\}, t = P) & \xrightarrow{a_2} s_3 : (c_1 : \{P\}, c_2 : \{\}, t = P) & \xrightarrow{a_1} \\
s_4 : (c_1 : \{2P\}, c_2 : \{2P\}, t = P) & \xrightarrow{P} s_5 : (c_1 : \{2P\}, c_2 : \{2P\}, t = 2P) & \xrightarrow{a_1} \\
s_6 : (c_1 : \{3P\}, c_2 : \{2P, 3P\}, t = 2P) & \xrightarrow{a_2} s_7 : (c_1 : \{3P\}, c_2, \{3P\}, t = 2P) & \xrightarrow{P} \cdots
\end{array}
$$

Note that in state s_2 there are two events with timestamps equal to $\tau_{\min} = P$ but they are both strongly enabled since it is neither the case that $a_1 \prec a_2$ nor $a_2 \prec a_1$. Furthermore, it is easy to see that the signal in c_1 is $\sigma_{c_1} = \{i \cdot P \mid i \in \mathbb{N}\}$ and the signal in c_2 is $\sigma_{c_2} = \{i \cdot P \mid i \in \mathbb{N}_{>0}\}$.

As it can be seen from the above example, $TTS(G, r_0)$ is generally infinite-state. There are two potential sources of infinity of state-space in DE models.

First, the timestamps may grow unbounded, as is the case with the above example. Second, it is unclear whether the set of events on each channel remain bounded. This is true in the above example, but is it generally true? In Section 3.1 we show that this is true for all DDE models. Then in Section 3.2 we show how timestamps can also be bounded.

3.1 Bounding the Number of Events in the Channels

Let us begin by providing some intuition about why the number of events in an execution of $TTS(G, r_0)$ remains bounded.

Consider the example in Figure 3. The set of events produced by "loop1" in channel c_1 is $\{i \cdot P \mid i \in \mathbb{N}\}$. Each new event with timestamp t that enters "loop2" from "loop1", will result in an infinite set of events $\{t + j \cdot D \mid j \in \mathbb{N}_{>0}\}$ in channel c_2. Therefore the set of all events in channel c_2 will be $\{i \cdot P + j \cdot D \mid i, j \in \mathbb{N}_{>0}\}$.

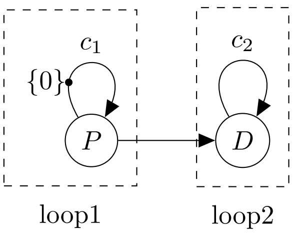


Fig. 3. Loop example

Because $P, D \in \mathbb{N}$, the timestamp of any event that appears in c_2 can be written as $k \cdot \mathsf{gcd}(P, D)$ for some k. In fact, there exists n, such that for all $k > n$, there exist positive i and j such that $k \cdot \mathsf{gcd}(P, D) = i \cdot P + j \cdot D$. So eventually all multiples of $\mathsf{gcd}(P, D)$ appear as timestamps of events in c_2.

Note that in every reachable state $s = (r, t)$ of $TTS(G, r_0)$, for $G = (A, C, D)$, an upper bound on the timestamp of any event is $\tau_{\min}(r) + \max\{D(a) \mid a \in A\}$, and a lower bound is $\tau_{\min}(r)$. Hence, because event timestamps in c_2 are separated by at least $\mathsf{gcd}(D, P)$, the number of events in c_2 satisfies

$$|r(c_2)| \leq \left\lceil \frac{\max\{D(a) \mid a \in A\}}{\mathsf{gcd}(D, P)} \right\rceil \text{ in any state } s = (r, t).$$

In general, let $G = (A, C, D)$ be a DE model and let r_0 be an initial channel state for G. Let $TTS(G, r_0) = (S, s_0, \rightarrow)$. Consider a state $s = (r, t) \in S$. Recall that r is a function $r : C \rightarrow 2^{\mathbb{R}_{\geq 0}}$. The *size* of r, denoted $|r|$, is defined to be

$$|r| := \sum_{c \in C} |r(c)|$$

Theorem 1 (Boundedness of channels). *There exists $K \in \mathbb{N}$ such that for every reachable state $s = (r, t)$ of $TTS(G, r_0)$, $|r| \leq K$.*

3.2 Bounding Timestamps

In $TTS(G, r_0)$ timestamps of events can still grow unbounded. Moreover, there is the additional global timestamp which grows unbounded too. Nevertheless, it is easy to see how to transform $TTS(G, r_0)$ in order to obtain an equivalent *bounded timed transition system*, which we will denote $BTS(G, r_0)$. To define

$BTS(G, r_0)$, we introduce some notation. Let $s = (r, t)$ be a state of $TTS(G, r_0)$. Let $\delta \in \mathbb{R}_{\geq 0}$ be such that $\delta \leq \tau_{\min}(r)$. Then we denote by $r - \delta$ the new channel state r' obtained from r by decrementing all timestamps in r by δ.

We are now ready to define $BTS(G, s_0)$. Its states are channel states, that is, the global timestamp is dropped. On the other hand, $BTS(G, r_0)$ has both discrete and timed transitions, like $TTS(G, r_0)$. A timed transition in $BTS(G, r_0)$ has the form $r \xrightarrow{\delta}_b r'$ where $\delta = \tau_{\min}(r)$ and $r' = r - \tau_{\min}(r)$. A discrete transition in $BTS(G, s_0)$ has the form $r \xrightarrow{a}_b r'$ with $r' = f(a, r, D(a))$, such that $\tau_{\min}(a, r) = \tau_{\min}(r) = 0$. That is, in $BTS(G, r_0)$, we keep track of time elapsing by appropriately decrementing the timestamps of pending events.

Theorem 2. *The set of reachable states of $BTS(G, r_0)$ is finite.*

Is it easy to show that a bisimulation exists between TTS and BTS. In particular, let $s = (r, t)$ be a reachable state of $TTS(G, r_0)$. It can be easily shown, by induction on the transition relation of $TTS(G, r_0)$, that s satisfies $t \leq \tau_{\min}(r)$. We define the relation R between states of $TTS(G, r_0)$ and states of $BTS(G, r_0)$, so that R contains all pairs $((r, t), r - t)$. It can be checked that R is a bisimulation relation.

4 Extended Discrete-Event Models

In this section we introduce extensions to the DDE model.

4.1 Non-deterministic DE

The *non-deterministic DE model* (NDE) extends DDE by allowing actors with variable delays, specifically intervals.

The syntax of an NDE model is almost the same as that of a DDE model. It is a labeled graph $G = (A, C, D)$, with A and C being as in a DDE model, and D associating an interval instead of a fixed value to each actor. Intervals must be non-empty, and can be of the form $[l, u]$, (l, u), (l, ∞), and so on, for $l, u \in \mathbb{N}$. When the interval is $[l, l]$ we simply write $D(a) = l$. We allow loops, but require that every loop visits at least one actor a such that $l(a) \geq 1$, where $l(a)$ is the lower bound of $D(a)$. The partial order $\prec$ is also defined in NDE, so that $a \prec a'$ iff there exists a path from a to a' such that for any actor a'' in the path (including a but excluding a') we have $l(a'') = 0$.

The semantics of NDE is defined as a timed transition system, as with DDE. Given an NDE graph G, and an initial channel state r_0, $TTS(G, r_0)$ is defined to be the tuple $(S, s_0, \rightarrow)$ where S and s_0 are as in DDE, and the transition relation $\rightarrow$ contains both discrete and timed transitions. A discrete transition of $TTS(G, r_0)$ is of the form $(r, t) \xrightarrow{a} (r', t)$ where a is strongly enabled in (r, t) and $r' = f(a, r, d)$, for some $d \in D(a)$. The definition of strongly enabled for NDE is the same as in DDE and uses the partial order $\prec$ as defined above. The timed transitions of $TTS(G, r_0)$ are defined in the same way as in DDE.

We point out that the above semantics allows to "reorder" events, in the sense that an event produced in a channel could have timestamp smaller than the events already in the channel. However, execution of actors is still guaranteed to happen in timestamp order. Also, since we are not currently using multisets, if an event is added to a channel which already has an event with the same timestamp then the two events are merged into one.

4.2 DE with Timed Automata

The *DE with timed automata model* (DETA) extends DDE by allowing actors to be modeled as timed automata. This extension allows actors in a discrete-event program to model environment behavior as well as have more elaborate internal behavior than DDE and NDE.

Like DDE and NDE, a DETA model is represented by a labeled graph. In the case of DETA, a label is either a fixed delay or a timed automaton (TA). Formally, a DETA model is a graph $G = (A, C, L)$, with A and C being as in a DDE model, and L being a labeling function which maps every actor $a \in A$ to either a delay $d \in \mathbb{N}$, or a TA $M = (Q, q_0, X, I, E)$, where:

- Q is the set of *locations* of M, and $q_0 \in Q$ is its *initial location.*
- X is the set of *clocks* of M. Both Q and X are finite sets.
- I is the *invariant function* which maps every $q \in Q$ to a simple convex constraint of the form $\bigwedge_i x_i \le k_i$, where $x_i \in X$ are clocks and $k_i \in \mathbb{N}$ are constants.
- E is the set of *transitions* of M. A transition is a tuple $h = (q, c, q', \phi, X')$ where:
  - $q, q' \in Q$: q is the source and q' the destination location of h.
  - $c \in C(a)$, i.e., c is either an input or an output channel of actor a.
  - ϕ is a simple constraint on clocks, called the *guard* of h.
  - $X' \subseteq X$ is a subset of clocks to be *reset* by h.

We require that every TA M in a DETA model be *receptive*, that is, able to accept any input event at any state. Formally, for every location q of M, and for every input channel c of a, the union of all guards of all outgoing transitions from q labeled with c must cover the whole space of clock valuations, that is, must be equivalent to the guard *true*.

We allow loops in DETA models, but we assume conservatively that the delay introduced by TA actors could be zero. Therefore we require that every loop visits at least one constant delay actor with delay ≥ 1. The partial order $\prec$ is defined for DETA in the same way as for DDE, by treating TA actors like zero-delay actors.

Before defining the semantics of DETA models we briefly recall the semantics of TA. A state of a TA M is a pair (q, v) where $q \in Q$ is a location and v is a *clock valuation*, that is, a function $v : X \to \mathbb{R}_{\ge 0}$ mapping every clock of M to a non-negative real value. We will use the term *TA state* for a pair (q, v), to avoid confusion with states of DE models, which we sometimes for clarity call *channel states.* The initial TA state of M is defined to be $(q_0, \mathbf{0})$, where $\mathbf{0}$ is the valuation assigning 0 to all clocks. M defines two types of transitions on this state-space: discrete and timed transitions. A discrete transition is possible from TA state (q, v) to TA state (q', v'), denoted $(q, v) \xrightarrow{c}_M (q', v')$, if M has a transition $h = (q, c, q', \phi, X')$

such that: (1) v satisfies the guard ϕ, denoted $v \models \phi$; (2) $v' = v[X' := 0]$, which means that $v'(x) = 0$ if $x \in X'$ and $v'(x) = v(x)$ otherwise; and (3) v' satisfies the invariant of the destination location q', denoted $v' \models I(q')$. A timed transition of delay $\delta \in \mathbb{R}_{\geq 0}$ is possible from TA state (q, v) to TA state (q, v'), denoted $(q, v) \xrightarrow{\delta}_M (q, v')$, if: (1) $v' = v + \delta$, which means that $v'(x) = v(x) + \delta$ for all $x \in X$; and (2) $v' \models I(q)$. The latter condition, together with our assumption on the form of invariants, ensures that the progress of time from v to v' does not violate any urgency constraints at location q. Note that, since $I(q)$ is downwards-closed, $v + \delta \models I(q)$ implies that for any $\delta' \leq \delta$, we also have $v + \delta' \models I(q)$.

We are now ready to define the semantics of DETA models. Consider a DETA model $G = (A, C, L)$ and an initial channel state r_0. Let A_{TA} be the subset of A such that $a \in A_{TA}$ iff $L(a)$ is a TA. For $a \in A_{TA}$, we denote the TA $L(a)$ by M_a. Then, G and r_0 define the timed transition system $TTS(G, r_0)$. A state of $TTS(G, r_0)$ is a triple (r, w, t) where r is a channel state, w is a total function mapping actors in A_{TA} to TA states, and $t \in \mathbb{R}_{\geq 0}$ is a global timestamp. For given $a \in A_{TA}$, $w(a)$ represents the TA state which M_a is currently at.

Like the other timed transition systems defined earlier, $TTS(G, r_0)$ has two types of transitions: discrete and timed. A discrete transition has the form $(r, w, t) \xrightarrow{a} (r', w', t)$, for $a \in A$, and is possible if:

- either $a \notin A_{TA}$, that is, $L(a) \in \mathbb{N}$, in which case $r' = f(a, r, L(a))$ and $w' = w$;
- or $a \in A_{TA}$, in which case
  1. either a has an input channel c such that $t \in r(c)$, in which case:
     (a) r' is obtained from r by removing the event with timestamp t from c, that is, $r'(c) = r(c) - \{t\}$ and $r'(c') = r(c')$ for all $c' \neq c$.
     (b) w' is obtained from w by having M_a take the discrete transition $w(a) \xrightarrow{c}_{M_a} w'(a)$ in reaction to the event in c, and having all other TA retain their state, that is, $w'(a') = w(a)$ for all $a' \in A_{TA}, a' \neq a$.
  2. or a has an output channel c such that $w(a) \xrightarrow{c}_{M_a} w'(a)$ and for all $a' \in A_{TA}$ s.t. $a' \neq a$, we have $w'(a') = w(a)$. In this case, $r'(c) = r(c) \cup \{t\}$ and $r'(c') = r(c')$ for all $c' \neq c$.

The case $a \notin A_{TA}$ corresponds to the case where a standard DDE actor fires, that is, an actor introducing a deterministic delay. The case $a \in A_{TA}$ corresponds to the case where a TA actor fires, that is, makes a discrete transition. In this case, the following subcases are possible:

- Either a consumes an event from an input channel and reacts to it (Case 1). Note that since M_a is assumed to be receptive, the transition $w(a) \xrightarrow{c}_{M_a} w'(a)$ is guaranteed to exist. Also note that it is by definition impossible for a TA actor to consume multiple events from multiple input channels in a single transition. This is true even when all these events may have the same timestamp. On the other hand, in that case the TA actor will consume all these events in a series of discrete *simultaneous* transitions, that is, without time passing in-between these transitions.
- Or a "spontaneously" produces an event to an output channel (Case 2).

A timed transition in $TTS(G, r_0)$ has the form $(r, w, t) \xrightarrow{\delta} (r, w', t + \delta)$, for $\delta \in \mathbb{R}_{\geq 0}$, and is possible if:

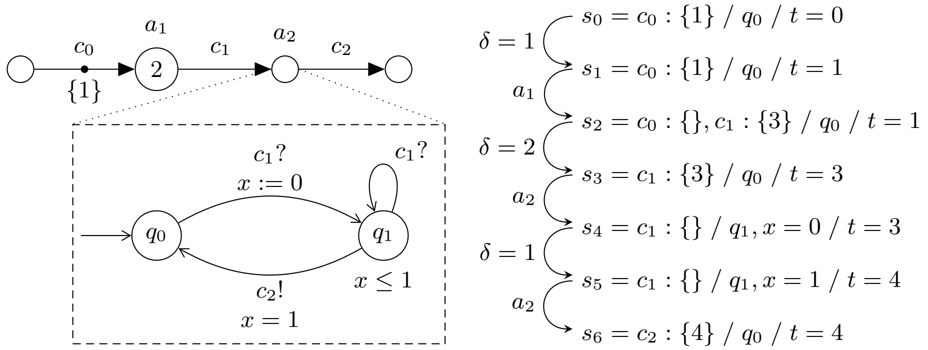
1. $t + \delta \leq \tau_{\min}(r)$; and
2. for all $a \in A_{TA}$, M_a has a timed transition by δ, that is, $w(a) \xrightarrow{\delta}_{M_a} w'(a)$ is a valid transition.

That is, a timed transition by δ is possible if it is possible for every TA in the system to let time elapse by δ, and also if this does not violate the urgency of any pending event in the system. Note that it is possible in DETA to have several timed transitions in a row.

An example of a DETA model is provided in the left part of Figure 4. There are four actors in this model, one of which, a_2, is a TA actor. The automaton for a_2 has two locations, q_0, q_1, and a single clock x. The invariant at q_1 is $x \leq 1$, whereas the invariant at q_0 is *true* and therefore not shown. The guard in the transition from q_0 to q_1 is also *true*, and not shown either. The label $x := 0$ means that x is reset on the corresponding transition (absence of such a label means that the clock is not reset). In the transitions of the automaton, we use the label $c?$ instead of c when c is an input channel, to emphasize the fact that the actor consumes an event from c. Similarly, we use $c!$ when c is an output channel, to emphasize the fact that the actor produces an event in c. A sample execution of this DETA model is provided in the right part of Figure 4.

Executions and signals in NDE and DETA: The notions of executions and signals can be easily extended from DDE to NDE and DETA models. Because of non-determinism in both NDE and DETA, Lemmas 2 and 3 do not hold in neither NDE nor DETA. This means in particular that the signal σ_c^ρ of a given channel c in these models generally depends on the execution ρ. For NDE and DETA models, we define σ_c to be the union of σ_c^ρ over all executions ρ.

Unboundedness of NDE and DETA: Boundedness does not hold for neither NDE nor DETA models. We can show that if we feed a variable delay with a periodic stream of events we can construct a sporadic stream which, in turn, if fed into



a_1
a_2
c_0
c_1
c_2
$\{1\}$
2
$c_1?$
$x := 0$
$c_1?$
q_0
q_1
$c_2!$
$x = 1$
$x \leq 1$
$\delta = 1$
$s_0 = c_0 : \{1\} \ / \ q_0 \ / \ t = 0$
$s_1 = c_0 : \{1\} \ / \ q_0 \ / \ t = 1$
a_1
$s_2 = c_0 : \{\}, c_1 : \{3\} \ / \ q_0 \ / \ t = 1$
$\delta = 2$
$s_3 = c_1 : \{3\} \ / \ q_0 \ / \ t = 3$
a_2
$s_4 = c_1 : \{\} \ / \ q_1, x = 0 \ / \ t = 3$
$\delta = 1$
$s_5 = c_1 : \{\} \ / \ q_1, x = 1 \ / \ t = 4$
a_2
$s_6 = c_2 : \{4\} \ / \ q_0 \ / \ t = 4$

Fig. 4. A DETA model (left) and a sample execution (right)

a periodic loop causes Theorem 1 to fail. Figure 5 illustrates the idea. In this model, the variable delay is implemented as a TA, resulting in a DETA model. The variable delay can also be implemented as an NDE actor a with $D(a) = [l, u]$, resulting in an NDE model.

More precisely, assume that the input stream of a has period P, as shown in Figure 5, and that $P > u$. Then the TA shown in the figure correctly implements a variable delay and every event coming out of the loop with period P will be given a variable delay $[l, u]$. Let a add delay $l + \frac{u-l}{i}$ to its i^{th} input event. This will result in an output stream of events $\{l+u-l, P+l+\frac{u-l}{2}, 2\cdot P+l+\frac{u-l}{4}, \ldots\}$.

In general, if a signal of the form $\{i \cdot P + \frac{x}{2^i} \mid i \in \mathbb{N}\}$ is fed into a loop with delay D, then the signal in the loop will contain events $\{i\cdot D+j\cdot P+\frac{x}{2^j} \mid i,j \in \mathbb{N}\}$. This set of events has the property that there is no bound $K \in \mathbb{N}$ such that for every n the number of events in window $[n, n+1]$ is less than K. Intuitively the reason is that for any K, an n can be found such that the equation $i \cdot D + j \cdot P = n$ has more than K solutions, and since, for large enough j, $x/2^j < 1$, the window $[n, n+1]$ will contain more than K events.

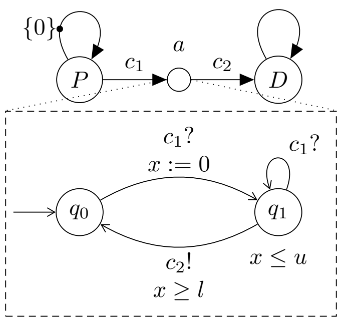


Fig. 5. Unbounded DETA model

5 Verification

We begin by defining the types of verification queries that we are interested in.

Let G be a DE model (i.e., a DDE, NDE, or DETA model) with set of channels C and let r_0 be an initial channel state. Let ρ be an execution of $TTS(G, r_0)$. Recall that σ_c^ρ, for channel $c \in C$, denotes the set of all events (timestamps) that occur in c along execution ρ in $TTS(G, r_0)$. A *signal query* is a query of the form "does σ_c^ρ satisfy some property ϕ?", where ϕ is a property written in (some subclass of) first-order logic. For instance, the property "an event occurs in c" can be written as $\phi := \exists\tau : \tau \geq 0$. The property "two events occur in c at two distinct times in the interval $[1, 2]$" can be written as $\exists\tau_1, \tau_2 : \tau_1 \neq \tau_2 \wedge 1 \leq \tau_1, \tau_2, \leq 2$. The property "two events occur in c separated by at most 1 time unit" can be written as $\exists\tau_1, \tau_2 : |\tau_1 - \tau_2| \leq 1$.

We are also interested in queries which involve states of $TTS(G, r_0)$. A *state query* asks whether there exists a reachable state $s = (r, t)$ such that r satisfies some property ψ. Again, we can imagine various types of properties ψ. For example, given constant $k \in \mathbb{N}$, ψ could be the expression $|r| > k$, which states that there are more than k events pending in the system, or the expression $|r(c)| > k$, for given $c \in C$, which states that there are more than k events pending on channel c. ψ could also be an expression such as those mentioned for signal queries above, stating, for example, that r contains an event with a certain timestamp or timestamp bounds, two events with a certain time difference, etc.

Channel signals are denotational semantics of DE models and signal queries allow to express natural properties on these. State queries are also important, as they refer to system snapshots as well as to implementation properties such as buffer space requirements. In the rest of this section we discuss how signal and state queries can be automatically checked on DDE models.

First, consider signal queries. They can be checked with the help of a *lasso* derived from BTS. This lasso is a finite and deterministic transition system (deterministic in the sense that every state has at most one successor), derived from BTS by merging all enabled discrete transitions from a given state s into a single *supertransition* where all corresponding actors fire. The diamond property (which by the bisimulation property also holds on BTS) ensures that this transformation is valid. We can analyze this lasso and compute, for every channel c, an affine expression that describes σ_c. Then we can reduce the problem of checking whether σ_c satisfies a signal query ϕ to an SMT (satisfiability modulo theory) problem. For example, the affine expression for channel c_2 for the example of Figure 3 is $i \cdot P + j \cdot D$, with $i, j \in \mathbb{N}_{>0}$. Checking whether there exist two events τ_1, τ_2 in σ_{c_2} such that $\tau_1 - \tau_2 = 5$, can then be reduced to checking satisfiability of the expression $\tau_1 = i_1 \cdot P + j_1 \cdot D \wedge \tau_2 = i_2 \cdot P + j_2 \cdot D \wedge \tau_1 - \tau_2 = 5$.

Second, for state queries, we can again use the lasso and the bisimulation of BTS and TTS to compute an affine expression characterizing the set of timestamps on a *per state* basis. We can then reduce the problem of whether there exists a reachable state satisfying a property ψ to a series of SMT problems, one for every affine expression computed for every state in the lasso.

6 Expressiveness

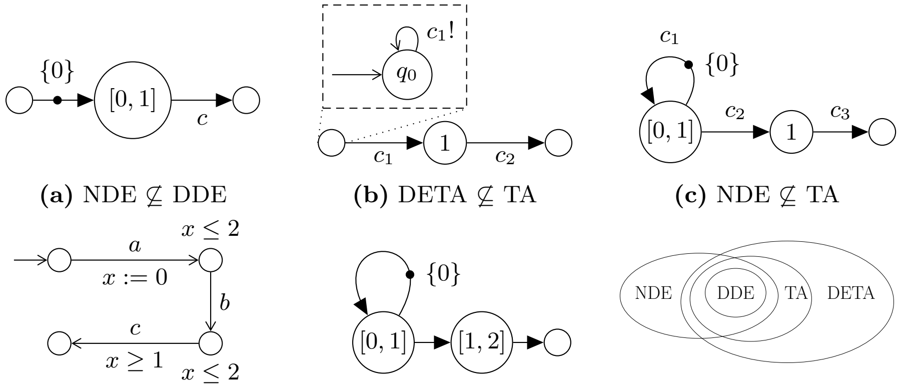
In this section we discuss how the various DE models introduced above are related to each other, as well as to timed automata, in terms of expressiveness. We write $\mathcal{A} \subseteq \mathcal{B}$ if for every model G in formalism $\mathcal{A}$ there exists a model G' in formalism $\mathcal{B}$ such that G and G' are equivalent in terms of denotational semantics, i.e., channel signals. More precisely, G and G' are equivalent if they refer to the same set of channels C, and for every $c \in C$, $\sigma_c^G = \sigma_c^{G'}$, where $\sigma_c^G, \sigma_c^{G'}$ are the signals of c in G, G', respectively.

To be able to compare the DE models with timed automata, we view TA as a subclass of DETA models. Concretely, suppose M is a TA whose transitions are labeled with $c_1, c_2, ..., c_n$. Then M can be seen as a DETA model with $n+1$ actors, $a, a_1, ..., a_n$, where a is a source actor labeled by M, and $a_1, ..., a_n$ are sink actors connected to a. In this interpretation, every label c_i of M is seen as a distinct output channel of a.

The expressiveness results, summarized in Figure 6(f), are as follows:

DDE $\subsetneq$ NDE: DDE $\subseteq$ NDE because the fixed delay d can be expressed as the interval $[d, d]$. NDE $\not\subseteq$ DDE because NDE allows non-deterministic behavior but DDE does not. Indeed, the NDE model of Figure 6(a) produces a single event on channel c at time $t \in [0, 1]$, but this is impossible to express in DDE.

DDE $\subsetneq$ TA: TA $\not\subseteq$ DDE because TA allows non-deterministic behavior but DDE does not. The example of Figure 6(a) can be easily constructed with TA. To see why DDE $\subseteq$ TA, consider the lasso defining the signals of a DDE model,



(a) NDE $\not\subseteq$ DDE **(b)** DETA $\not\subseteq$ TA **(c)** NDE $\not\subseteq$ TA

(d) TA $\not\subseteq$ NDE **(e)** NDE $\not\subseteq$ DETA **(f)** Model expressiveness

Fig. 6. Models used in expressiveness discussion

discussed in Section 5. The affine expressions describing the channel signals can be directly transformed into parallel compositions of simple TA with periodic self-loops. For example, $(2 + 3 \cdot i) \cup (3 + 7 \cdot j)$ can be trivially transformed into the parallel composition of two TA.

DDE $\subsetneq$ DETA: DDE $\subseteq$ DETA because every DDE model is by definition a DETA model (one that has no TA actors). DETA $\not\subseteq$ DDE again because of non-determinism.

TA $\subsetneq$ DETA: As defined above, TA is by definition a subclass of DETA. To see that DETA $\not\subseteq$ TA, consider the DETA model shown in Figure 6(b). In this model, every event produced by the TA on channel c_1 is delayed by the constant delay actor by exactly 1 time unit. Since there is no bound on the number of events that can be produced on c_1 in a time interval of size 1, an equivalent TA model would require an unbounded number of clocks.

NDE $\not\subseteq$ TA: To see this, consider the example of Figure 6(c). Similarly to the model of Figure 6(b), in this model the number of events that can be produced in an interval of size 1 on channel c_2 is unbounded, and for every such event a TA implementation would require a separate clock to produce the corresponding event on channel c_3.

We also conjecture that the TA of Figure 6(d) cannot be implemented in NDE. This TA produces three events, a, b, c, in that order, with the constraint that the distance between a and c is in the interval $[1, 2]$. This behavior requires both non-deterministic delays and some form of synchronization to ensure that b occurs before c, which does not appear to be implementable in NDE.

We finally conjecture that the NDE model of Figure 6(e) cannot be implemented in DETA. The loop in the model can produce unbounded bursts of events in a finite window. A variable delay needs to be introduced for every such event. In DETA, variable delays can only be implemented using timed automata, which only have a finite number of clocks.

7 Related Work

The term *discrete-event systems* (DES) is often used to denote untimed models based on automata, Petri nets, and related formalisms, with a focus on controller synthesis and similar problems [17,4]. In this paper we study timed DE models.

Timed and actor-oriented DE models have been considered previously in a number of works, but with a different focus than that of our paper. [20,14,16,5,15] focus on the semantics of timed systems. [10] focuses on compositionality and preservation of properties such as worst-case throughput or latency. [3] presents a translation of Ptolemy DE models to Real-Time Maude for the purposes of verification, but does not study the verification problem *per se*. Rebeca is an actor-oriented language where actors can specify timed behavior using statements such as *delay* [18]. However, decidability of model-checking for timed Rebeca has not been investigated. [6] extends finite-state automata with delay blocks and examines the expressiveness of the resulting timed languages.

A large body of research exists on the real time calculus (RTC) [19]. RTC focuses on performance properties for which interesting analytic and modular techniques can be derived. On the other hand, RTC models have relatively limited expressiveness compared to state-based models such as TA and the analysis results are generally approximations. Recent works on RTC include techniques for models combining analytic components as used in standard RTC with TA [13] and techniques to handle networks with cyclic dependencies [11].

DE models have a significant difference from time(d) Petri nets, even though at first sight they may appear similar. In Petri nets, a transition fires only if enough tokens are available in *every* input place, whereas in DE, an actor fires in timestamp order, and may consume tokens only from *some* input channels.

8 Perspectives

The verification problems for NDE and DETA remain open. We are currently exploring ideas to constrain the model to regain, or statically check for, boundedness, which would enable transformation of bounded DETA models to timed automata. We are also currently working on extracting affine expressions directly from DDE models (without the use of lassos) and then extending this technique to NDE, which would allow verification of signal queries on NDE despite unboundedness. TA are another possible symbolic representation of signals, natural in DETA models. It is easy to see how to transform TA signal representations by fork, join, constant- and variable-delay actors, but not how to compute fixpoints which seems needed for general cyclic networks.

Another direction for future work is investigating model-checking of general temporal logics against DE models, or coming up with new logics especially designed for DE models.

Another direction is to enrich expressiveness of DDE and NDE models, for instance, by adding control-expressive actors such as synchronizers, which from the comparison between NDE and TA appear important. Adding values to events is another possibility for extending DE models in general, including DETA models.

References

1. Agha, G.: ACTORS: A Model of Concurrent Computation in Distributed Systems. The MIT Press Series in Artificial Intelligence. MIT Press, Cambridge (1986)
2. Alur, R., Dill, D.: A theory of timed automata. Theoretical Computer Science 126, 183–235 (1994)
3. Bae, K., Csaba Olveczky, P., Feng, T.H., Lee, E.A., Tripakis, S.: Verifying Hierarchical Ptolemy II Discrete-Event Models using Real-Time Maude. Science of Computer Programming 77(12), 1235–1271 (2012)
4. Cassandras, C., Lafortune, S.: Introduction to Discrete Event Systems. Kluwer Academic Publishers (1999)
5. Cataldo, A., Lee, E., Liu, X., Matsikoudis, E., Zheng, H.: A constructive fixed-point theorem and the feedback semantics of timed sy stems. In: WODES (2006)
6. Chatterjee, K., Henzinger, T.A., Prabhu, V.S.: Finite automata with time-delay blocks. In: EMSOFT 2012, pp. 43–52. ACM (2012)
7. Dill, D.L.: Timing assumptions and verification of finite-state concurrent systems. In: Sifakis, J. (ed.) CAV 1989. LNCS, vol. 407, pp. 197–212. Springer, Heidelberg (1990)
8. Eidson, J.C., Lee, E.A., Matic, S., Seshia, S.A., Zou, J.: Distributed real-time software for cyber-physical systems. Proceedings of the IEEE 100(1), 45–59 (2012)
9. Eker, J., Janneck, J., Lee, E.A., et al.: Taming heterogeneity – the Ptolemy approach. Proc. IEEE 91(1) (2003)
10. Geilen, M., Tripakis, S., Wiggers, M.: The Earlier the Better: A Theory of Timed Actor Interfaces. In: HSCC. ACM (2011)
11. Jonsson, B., Perathoner, S., Thiele, L., Yi, W.: Cyclic dependencies in modular performance analysis. In: EMSOFT 2008, pp. 179–188. ACM (2008)
12. Kahn, G.: The semantics of a simple language for parallel programming (1974)
13. Lampka, K., Perathoner, S., Thiele, L.: Analytic real-time analysis and timed automata: a hybrid method for analyzing embedded real-time systems. In: EMSOFT, pp. 107–116. ACM (2009)
14. Lee, E.A.: Modeling concurrent real-time processes using discrete events. Annals of Software Engineering 7(1), 25–45 (1999)
15. Liu, X., Lee, E.A.: CPO semantics of timed interactive actor networks. Theoretical Computer Science 409(1), 110–125 (2008)
16. Liu, X., Matsikoudis, E., Lee, E.A.: Modeling timed concurrent systems. In: Baier, C., Hermanns, H. (eds.) CONCUR 2006. LNCS, vol. 4137, pp. 1–15. Springer, Heidelberg (2006)
17. Ramadge, P., Wonham, W.: The control of discrete event systems. Proceedings of the IEEE (January 1989)
18. Sirjani, M., Jaghoori, M.M.: Ten Years of Analyzing Actors: Rebeca Experience. In: Agha, G., Danvy, O., Meseguer, J. (eds.) Formal Modeling: Actors, Open Systems, Biological Systems. LNCS, vol. 7000, pp. 20–56. Springer, Heidelberg (2011)
19. Thiele, L., Chakraborty, S., Naedele, M.: Real-time calculus for scheduling hard real-time systems. In: ISCAS, pp. 101–104 (2000)
20. Yates, R.K.: Networks of real-time processes. In: Best, E. (ed.) CONCUR 1993. LNCS, vol. 715, pp. 384–397. Springer, Heidelberg (1993)

Symmetry Breaking for Multi-criteria Mapping and Scheduling on Multicores

Pranav Tendulkar, Peter Poplavko*, and Oded Maler

Verimag (CNRS and University of Grenoble), France

Abstract. Multiprocessor mapping and scheduling is a long-old difficult problem. In this work we propose a new methodology to perform mapping and scheduling along with buffer memory optimization using an SMT solver. We target split-join graphs, a formalism inspired by synchronous data-flow (SDF) which provides a compact symbolic representation of data-parallelism. Unlike the traditional design flow for SDF which involves splitting of a big problem into smaller heuristic sub-problems, we deal with this problem as a whole and try to compute exact Pareto-optimal solutions for it. We introduce symmetry breaking constraints in order to reduce the run-times of the solver. We have tested our work on a number of SDF graphs and demonstrated the practicality of our method. We validate our models by running an image decoding application on the Tilera multicore platform.

Keywords: synchronous data-flow, multiprocessor, multicore, mapping, scheduling, SMT, SAT solver.

1 Introduction

This work is motivated by a key important problem in contemporary computing: *how to exploit efficiently the resources provided by a multicore platform while executing application programs*. The problem has many variants depending on the intended use of the platform (general-purpose server or a dedicated accelerator), the specifics of the architecture (memory hierarchy, interconnect), the granularity of parallelism (instruction level, task level), the class of applications and the programming model. We focus on applications such as video, audio and other forms of signal processing which are naturally structured in a *data-flow* style as a network of interconnected software components (actors, filters, tasks). Such a description already exposes the precedence constraints among tasks and hence the task-level parallelism inherent in the application. More specifically we address applications written as *split-join graphs*, which can be viewed as a variant of the Synchronous Data-Flow (SDF) formalism [13,20], or an abstract semantic model of a subset of streaming languages such as StreaMIT [24]. Such formalisms, in addition to precedence constraints, also provide a compact *symbolic representation* of data-parallelism, namely, the presence of numerous tasks which have identical function and can be executed in parallel for different data. Once the split-join graph is annotated

* The research work of this author has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under grant agreement no. 288175 [Certainty].

with execution time figures and the data-parallel tasks have been explicitly expanded we obtain a task graph [3] whose *deployment* on the execution platform is the subject of optimization.

The deployment decisions that we consider and which may affect cost and performance are the following. First we can vary the number of processors used which gives a rough estimation of the cost of the platform (and its static power consumption). On a given configuration it remains to *map* tasks to processors, and to *schedule* the execution order on each processor. The performance measures to evaluate such a deployment are the total execution time (*latency*) and the size of the communication *buffers* which depend on the execution order. This is a multi-criteria (cost, latency and buffer size) optimization problem whose single-criterium version is already intractable. We take advantage of recent progress in SMT (SAT modulo theory) solvers [17,7] to provide a good approximation of the Pareto front of the problem. We encode the precedence and resource constraints of the problem in the theory of linear arithmetic and, following [15,14], we submit queries to the solver concerning the existence of solutions whose costs reside in various parts of the multi-dimensional cost space. Based on the answers to these queries we obtain a good approximation of the optimal trade-off between these criteria. The major computational obstacle is the intractability of the mapping and scheduling problems aggravated by the exponential blow-up while expanding the graph from symbolic to explicit form. We tackle this problem by introducing "symmetry breaking" constraints among identical processors and identical tasks. For the latter we prove a theorem concerning the optimality of schedules where instances of the same actor are executed according to a fixed *lexicographical* order.

The rest of the paper is organized as follows. In Section 2 we give some background on split-join graphs and their transformation into task graphs and prove a useful property of their optimal schedules. In Section 3 we write down in more detail the constraint-based formulation of deployment and present our multi-criteria cost-space exploration procedure. An experimental evaluation of our approach appears in Section 4, including a validation on the Tilera multicore platform. We conclude by discussing related and future work.

2 Split-Join Graphs

A parallelization factor is any number of the form α (split) or $1/\alpha$ (join) for $\alpha \in \mathbb{N}$. We use Σ^* to denote the set of sequences over a set Σ and use $\sqsubset$ for the prefix relation with $\xi \sqsubset \xi \cdot \xi'$, where $\xi \cdot \xi'$ denotes concatenation.

Definition 1 (Split-Join and Task Graphs). *A split-join graph is $S = (V, E, d, r)$ where (V, E) is a directed acyclic graph (DAG), that is, a set V of nodes, a set $E \subseteq V \times V$ of edges and no cyclic paths. The function $d : V \to \mathcal{R}_+$ defines the execution times of the nodes and $r : E \to \mathcal{Q}$ assigns a parallelization factor to every edge. An edge e is a* split, join *or* neutral *edge depending on whether $r(e) > 1$, < 1 or $= 1$. A split-join graph with $r(e) = 1$ for every e is called a task-graph and is denoted by $T = (U, \mathcal{E}, \delta)$, where the three elements in the tuple correspond to V, E and d.*

The decomposability of a task into parallelizable sub-tasks is expressed as a numerical label (parallelization factor) on a precedence edge leading to it. A label α on the edge from A to B means that every executed instance of task A spawns α instances of task B. Likewise, a $1/\alpha$ label on the edge from B to C means that all those instances of B should terminate and their outputs be joined before executing C (see Fig. 1). A task graph can thus be viewed as obtained from the split-join graph by making data parallelism *explicit* . To distinguish between these two types of graphs we call the nodes of the split-join graphs *actors* (task types) and those of the task graph *tasks*.

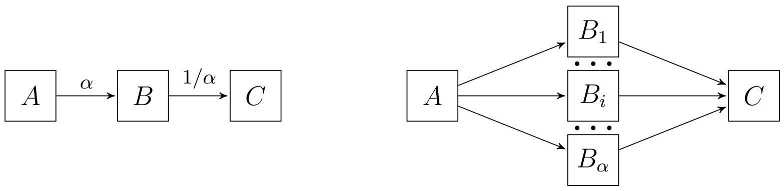


Fig. 1. A simple split-join graph and its expanded task graph. Actor B has α instances.

The DAG structure naturally induces a partial-order relation $\angle$ over the actors such that $v \angle v'$ if there is a path form v to v'. The set of *minimal* elements with respect to $\angle$ is $V_\bullet \subseteq V$ consisting of nodes with no incoming edges. Likewise, the *maximal* elements $V^\bullet$ are those without outgoing edges. An *initialized path* in a DAG is directed path $\pi = v_1 \cdot v_2 \cdots v_k$ starting from some $v_1 \in V_\bullet$. Such a path is *complete* if $v_k \in V^\bullet$. With any such path we associate the *multiplicity signature*

$$\xi(\pi) = (v_1, \alpha_1) \cdot (v_2, \alpha_2) \cdots (v_{k-1}, \alpha_{k-1})$$

where $\alpha_i = r((v_i, v_{i+1}))$. We will also abuse ξ to denote the projection of the signature on the multiplication factors, that is $\xi(\pi) = \alpha_1 \cdot \alpha_2 \cdots \alpha_{k-1}$.

To ensure that different instances of the same actor communicate with the matching instances of other actors and that such instances are joined together properly, we need an *indexing scheme* similar to indices of multi-dimensional arrays accessed inside nested loops. Because an actor may have several ancestral paths, we need to ensure that its indices via different paths agree. This will be guaranteed by a well-formedness condition that we impose on the multiplicity signatures along paths.

Definition 2 (Parenthesis Alphabet). *Let $\Sigma = \{1\} \cup \Sigma_\{ \cup \Sigma_\}$ be any set of symbols consisting of a special symbol 1 and two finite sets $\Sigma_\{$ and $\Sigma_\}$ admitting a bijection which maps every $\alpha \in \Sigma_\{$ to $\alpha' \in \Sigma_\}$.*

Intuitively α and α' correspond to a matching pair consisting of a split α and its inverse join $1/\alpha$. These can be viewed also as a pair of (typed) left and right *parentheses*.

Definition 3 (Canonical Form). *The canonical form of a sequence ξ over a parentheses alphabet Σ is the sequence $\bar{\xi}$ obtained from ξ by erasing occurrences of the neutral element 1 as well as matching pairs of the form $\alpha \cdot \alpha'$.*

For example, the canonical form of $\xi = 5 \cdot 1 \cdot 3 \cdot 1 \cdot 1 \cdot 1/3 \cdot 1 \cdot 2$ is $\bar{\xi} = 5 \cdot 2$. Note that the (arithmetic) products of the factors of ξ and of $\bar{\xi}$ are equal and we denote this value by $c(\xi)$ and let $c(\epsilon) = 1$. A sequence ξ is *well-parenthesized* if $\bar{\xi} \in \Sigma_{\{}^{*}$, namely its canonical form consists only of left parentheses. Note that this notion refers also to signature *prefixes* that can be *extended* to well-balanced sequences, namely, sequences with no violation of being well-parenthesized by a join not compatible with the *last* open split.

Definition 4 (Well Formedness). *A split-join graph is well formed if:*

1. *Any complete path π satisfies $c(\xi(\pi)) = 1$;*
2. *The signatures of all initialized paths are well parenthesized.*

The first condition ensures that the graph is meaningful (all splits are joined) and that the multiplicity signatures of any two paths leading to the same actor v satisfy $c(\xi) = c(\xi')$. We can thus associate unambiguously this number with the actor itself and denote it by $c(v)$. This *execution count* is the number of instances of actor v that should be executed.

The second condition forbids, for example, sequences of the form $2 \cdot 3 \cdot 1/2 \cdot 1/3$. It implies an additional property: every two initialized paths π and π' leading to the same actor satisfy $\bar{\xi}(\pi) = \bar{\xi}(\pi')$. Otherwise, if two paths would reach the same actor with different canonical signatures, there will be no way to close their parentheses by the same path suffix. Although split-join graphs *not* satisfying Condition 2 can make sense for certain computations, they require more complicated mappings between tasks and they will not be considered here, but see a brief discussion in Section 5. For well-formed graphs, a *unique canonical signature*, denoted by $\bar{\xi}(v)$, is associated with every actor.

Definition 5 (Indexing Alphabet and Order). *An actor v with $\bar{\xi}(v) = \alpha_1 \cdots \alpha_k$ defines an indexing alphabet A_v consisting of all k-digit sequences $h = a_1 \cdots a_k$ such that $0 \leq a_i \leq \alpha_i - 1$. This alphabet can be mapped into $\{0, \ldots, c(v) - 1\}$ via the following recursive rule:*

$$\mathcal{N}(\varepsilon) = 0 \quad \textit{and} \quad \mathcal{N}(h \cdot a_j) = \alpha_j \cdot \mathcal{N}(h) + a_j$$

We use $\ll_v$ to denote the lexicographic total order over A_v which coincides with the numerical order over $\mathcal{N}(A_v)$.

Every instance of actor v will be indexed by some $h \in A_v$ and will be denoted as v_h. We use notation h and A_v to refer both to strings and to their numerical interpretation via $\mathcal{N}$. In the latter case v_h will refer to the task in position h according to the lexicographic order $\ll_v$. See for example, tasks $B_0, B_1, \ldots$ in Figure 1.

Definition 6 (Derived Task Graph). *From a well-formed split-join graph $S = (V, E, d, r)$ we derive the task graph $T = (U, \mathcal{E}, \delta)$ as follows: $U = \{v_h | v \in V, h \in A_v\}$, $\mathcal{E} = \{(v_h, v'_{h'}) \mid (v, v') \in E, (h \sqsubseteq h' \vee h' \sqsubseteq h)\}$ and $\forall v, \forall h \in A_v,\ \delta(v_h) = d(v)$.*

Notation $h \sqsubseteq h'$ indicates that string h' is a prefix of h. To take an example, according to the definition, a split edge (v, v') is expanded to a set of edges $\{(v_h, v'_{h \cdot a}) \mid a = 0 \ldots \alpha - 1\}$, where $\alpha = r((v, v'))$. The tasks can be partitioned naturally according to their actors, letting $U = \bigcup_{v \in V} U_v$ and $U_v = \{v_h : h \in A_v\}$. A permutation $\omega : U \to U$ is *actor-preserving* if it can be written as $\omega = \bigcup_{v \in V} \omega_v$ and each ω_v is a permutation on U_v.

Definition 7 (Deployment). *A deployment for a task graph $T = (U, \mathcal{E}, \delta)$ on an execution platform with a finite set M of processors consists of a mapping function $\mu : U \rightarrow M$ and a scheduling function $s : U \rightarrow \mathcal{R}_+$ indicating the start time of each task. A deployment is called feasible if it satisfies precedence and mutual exclusion constraints, namely, for each pair of tasks we have:*

Precedence: $(u, u') \in \mathcal{E} \Rightarrow s(u') - s(u) \geq \delta(u)$

Mutual exclusion: $\mu(u) = \mu(u') \Rightarrow [(s(u') - s(u) \geq \delta(u)) \vee (s(u) - s(u') \geq \delta(u'))]$

Note that $\mu(u)$ and $s(u)$ are decision variables while $\delta(u)$ is a constant. The *latency* of the deployment is the termination time of the last task, $\max_{u \in U}(s(u) + \delta(u))$. The problem of optimal scheduling of a task-graph is already NP-hard due to the non-convex mutual exclusion constraints. This situation is aggravated by the fact that the task-graph will typically be exponential in the size of the split-join graph. On the other hand, it admits many tasks which are identical in their duration and isomorphic in their precedence constraints. In the sequel we exploit this symmetry by showing that all tasks that correspond to the same actor can be executed according to a *lexicographic order* without compromising latency.

Definition 8 (Ordering Scheme). *An ordering scheme for a task-graph $T = (U, \mathcal{E}, \delta)$ derived from a split-join graph $G = (V, E, r, d)$ is a relation $\prec = \bigcup_{v \in V} \prec_v$ where each $\prec_v$ is a total order relation on U_v.*

In the lexicographic ordering scheme $\lll$, the tasks $v_h \in U_v$ are ordered in the lexicographic order $\lll_v$ of their indices 'h'. We say that a schedule s is *compatible* with an ordering scheme $\prec$ if $v_h \prec v_{h'}$ implies $s(v_h) \leq s(v_{h'})$. We denote such an ordering scheme by $\prec^s$ and use notation $v[h]$ for the task occupying position h in $\prec^s_v$.

Lemma 1. *Let s be a feasible schedule and let v and v' be two actors such that $(v, v') \in E$. Then*

1. *If $r(v, v') = \alpha \geq 1$, then for every $h \in [0, c(v) - 1]$ and every $a \in [0, \alpha - 1]$ we have*
$$s(v'[\alpha h + a]) - s(v[h]) \geq d(v).$$
2. *If $r(v, v') = 1/\alpha$ then for every $h \in [0, c(v) - 1]$ and every $a \in [0, \alpha - 1]$ we have*
$$s(v'[h]) - s(v[\alpha h + a]) \geq d(v).$$

Proof. The precedence constraints for Case 1 are in fact $s(v'_{\alpha h + a}) - s(v_h) \geq d(v)$, and we have to prove that in this expression the lexicographic index v_h can be replaced by schedule-compatible index $v[h]$. Let $j = h\alpha + a$ and $j' = j + 1$. Since each instance of v is a predecessor of exactly α instances of v', the execution of $v'[j]$ must occur after the completion of *at least* $\lceil j'/\alpha \rceil$ instances of v. By construction, this is not earlier than the termination of the *first* $\lceil j'/\alpha \rceil$ instances of v to occur in schedule s. In our notation this can be written as:

$$s(v'[j]) \geq s(v[\lceil j'/\alpha \rceil - 1]) + d(v)$$

Substituting j and j' into the above formula we obtain our thesis. A similar argument holds for Case 2. □

Theorem 1 (Lexicographic Ordering). *Every feasible schedule s can be transformed into a latency-equivalent schedule s' compatible with the lexicographic order $\ll$.*

Proof. Let ω_s be an actor-preserving permutation on U defined as $\omega_s(v_h) = v[h]$. In other words, ω_s maps the task in position h according to $\ll$ to the task occupying that position according to $\prec^s$. The new deployment is defined as

$$\mu'(v_h) = \mu(\omega_s(v_h)) \quad \text{and} \quad s'(v_h) = s(\omega_s(v_h)).$$

Permuting tasks of the same duration does not influence latency nor the satisfaction of resource constraints. All that remains to show is that s' satisfies precedence constraints. Each v_h is mapped into $v[h]$ and each of its v' sons (resp. parents) is mapped into $v'[\alpha h + a]$, $0 \leq a \leq \alpha - 1$. Hence a precedence constraint for s' of the form

$$s'(v_{h \cdot a}) - s'(v_h) \geq d(v)$$

is equivalent to

$$s(v[\alpha h + a]) - s(v[h]) \geq d(v)$$

which holds by virtue of Lemma 1 and the feasibility of s. □

For example, in Figure 2 we illustrate a task graph, a feasible schedule and the same schedule transformed into a lexicographic-compatible schedule by a permutation of the task indices.

The implication of this result is that we can introduce additional lexicographic constraints to the formulation of the scheduling problem without losing optimality and thus significantly reduce the search space, *i.e.,* we can do symmetry breaking.

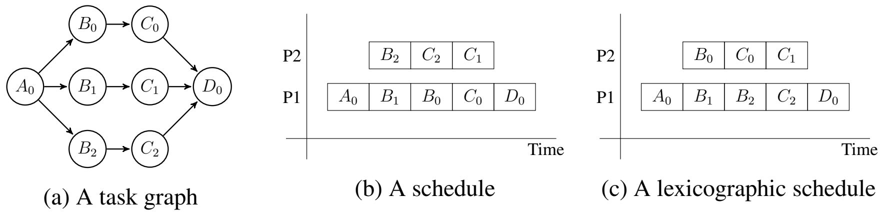


Fig. 2. Illustration of the lexicographic ordering theorem

3 Constraint-Based Feasible Cost-Space Exploration

In this section, to illustrate the effectiveness of the proposed symmetry breaking result, we encode the multicore deployment for split-join graphs as a quantifier-free SMT problem, defined by a set of constraints in the theory of linear arithmetics. Expressing scheduling problems using constraint solvers is fairly standard [1,2,27,15] and various formulations may differ in the assumptions they make about the application and the architecture and the aspects of the problem they choose to capture. For clarity and page limitation reasons, we present only the non-pipelined scheduling case.

To take advantage of symmetry breaking, we assume a multicore architecture where all cores are symmetric (homogeneous) both in terms of the computation times and the memory access times to the task communication data located in a shared memory. Fortunately, many advanced multicore architectures [16,25,11] either have a global symmetric shared memory for all processors or contain large groups of processors – so-called *clusters* – inside which this assumption holds. The access to the shared memory (including contentions and cache misses) is taken into account in the task execution times δ. In accordance with a common practice in SDF literature, we assume that a separate communication buffer is assigned to each edge (channel) (v, v') of the split-join graph so that tasks associated with the same actor read from and write to the same buffer.

To take buffer storage into account, we enrich the split-join graph model to become $G = (V, E, d, w, r)$ with $w(v, v')$ assigning to any edge in E the amount of data (in bytes) communicated from an instance of v to an instance of v' (this is called *token size* in the SDF literature). The corresponding task graph is $T = (U, \mathcal{E}, \delta, w^{\uparrow}, w^{\downarrow})$ where $w^{\uparrow}_{v,v'}$ is the amount of data *written* on the channel (v, v') by a task in U_v and $w^{\downarrow}_{v,v'}$ is the amount *read* by a task in $U_{v'}$. We assume that u allocates this memory space while starting and that u' releases it upon termination. The relation between w, $w^{\uparrow}$ and $w^{\downarrow}$ depends on the type of the edge: for a split edge $w^{\uparrow}_{v,v'} = \alpha w(v, v')$ and $w^{\downarrow}_{v,v'} = w(v, v')$ while for join edges we have $w^{\uparrow}_{v,v'} = w(v, v')$ and $w^{\downarrow}_{v,v'} = \alpha w(v, v')$.

In the following we write down the constraints that define a feasible schedule and its cost in terms of latency, number of processors and buffer size.

- **Completion time and precedence:** $e(u)$ is the time when task u terminates and a task cannot start before the termination of its predecessors.

$$\bigwedge_{u \in U} e(u) = s(u) + \delta(u) \;\wedge \bigwedge_{(u,u') \in \mathcal{E}} e(u) \leq s(u')$$

- **Mutual exclusion:** tasks running on same processor should not overlap in time.

$$\bigwedge_{u \neq u' \in U} (\mu(u) = \mu(u')) \Rightarrow (e(u) \leq s(u') \vee (e(u') \leq s(u))$$

- **Buffer:** these constraints compute the buffer size of every channel $(v, v') \in E$. They are based on the observation that buffer utilization is piecewise-constant over time, with jumps occurring upon initiation of writers and termination of readers. Hence the peak value of memory utilization can be found among one out of finitely-many starting points.
The first constraint defines $W^{\uparrow}_{v,v'}(u, u_*)$, the contribution of writer $u \in U_v$ to the filling of buffer (v, v') observed at the start of a writer $u_* \in U_v$:

$$\bigwedge_{(v,v') \in E} \bigwedge_{u \in U_v} \bigwedge_{u_* \in U_v} \begin{array}{l} (s(u) > s(u_*)) \wedge (W^{\uparrow}_{v,v'}(u, u_*) = 0) \vee \\ (s(u) \leq s(u_*)) \wedge (W^{\uparrow}_{v,v'}(u, u_*) = w^{\uparrow}_{v,v'}) \end{array}$$

Likewise the value $W^{\downarrow}_{v,v'}(u', u_*)$ is the (negative) contribution of reader u' to buffer (v, v') observed at the start of writer u_*:

$$\bigwedge_{(v,v')\in E}\ \bigwedge_{u'\in U_{v'}}\ \bigwedge_{u_*\in U_v} \begin{array}{l}((e(u') > s(u_*)) \wedge W^{\downarrow}_{v,v'}(u', u_*) = 0) \vee \\ (e(u') \le s(u_*)) \wedge (W^{\downarrow}_{v,v'}(u', u_*) = w^{\downarrow}_{v,v'})\end{array}$$

The total amount of data in buffer (v, v') at the start of task $u_* \in U_v$, denoted by $R_{v,v'}(u_*)$, is the sum of contributions of all readers and writers already executed:

$$\bigwedge_{(v,v')\in E}\ \bigwedge_{u_*\in U_v} R_{v,v'}(u_*) = \sum_{u\in U_v} W^{\uparrow}_{v,v'}(u, u_*) - \sum_{u'\in U_{v'}} W^{\downarrow}_{v,v'}(u', u_*)$$

The buffer size for (v, v'), denoted by $B_{v,v'}$ is the maximum over all the start times of tasks in U_v:

$$\bigwedge_{(v,v')\in E}\ \bigwedge_{v_*\in U_v} R_{v,v'}(u_*) \le B_{v,v'}$$

– **Costs**: The following constraints define the cost vector associated with a given deployment, which is $C = (C_l, C_n, C_b)$, where the costs indicate, respectively, *latency* (termination of last task), number of *processors* used and total *buffer size*.

$$\bigwedge_{u\in U} e(u) \le C_l \ \wedge\ \bigwedge_{u\in U} \mu(u) \le C_n \ \wedge\ \sum_{(v,v')\in E} B_{v,v'} \le C_b$$

We refer to the totality of these constraints as $\varphi(\mu, s, C)$ which are satisfied by any feasible deployment (μ, s) whose cost is C.

– **Symmetry breaking:** We add two kinds of symmetry-breaking constraints, which do not change optimal costs. Firstly, we add the lexicographic task ordering constraints as implied from Theorem 1 – henceforth: *task symmetry*

$$\bigwedge_{v\in V}\ \bigwedge_{v_h, v_{h+1}\in U_v} s(v_h) \le s(v_{h+1})$$

where v_h denotes the instance of v at the h-th position in the order $\ll_v$.
Secondly we add fairly standard constraints to exploit *processor symmetry*: processor 1 runs task 1, processor 2 runs the lowest index task not running on processor 2, *etc.*. Therefore, let us number all tasks arbitrarily with a unique index: u^1, u^2, *etc.* The processor symmetry breaking is defined by the following constraint:

$$\mu(u^1) = 1 \ \wedge \bigwedge_{2\le i\le |U|} \mu(u^i) \le \max_{1\le j<i} \mu(u^j) + 1$$

More details on how all constraints were encoded in Z3 solver can be found in [22].

SAT and SMT solvers were designed for deciding satisfiability, not for optimization. However, such solvers can be used to find optimal costs by submitting queries concerning the existence of solutions with specific costs, which can be viewed as a binary search in the cost space with the solver acting as an oracle. We focus on *multi-criteria* optimization problems where we seek to find optimal trade-offs between latency C_l, number of processors C_n and buffer storage C_b. Such problems [8] do not admit a unique optimal solution but rather a set of *efficient* Pareto solutions [18] that cannot be improved in one cost dimension without being worsened in others. The set of such solutions, known as the *Pareto front*, represents the optimal trade-offs between the conflicting criteria. Following [14] we use queries to an SMT solver to find an approximation of the Pareto front. We summarize below the essence of the exploration methodology of [14], which can be viewed as a multi-dimensional generalization of binary search. Other approaches for multi-criteria optimization can be found in [8,28,6].

Let $Q(c)$ be a shorthand for the satisfiability query $\exists\mu\exists s$ s.t. $\varphi(\mu, s, c)$ which asks whether there is a feasible deployment whose cost vector is equal to c. If the solver answers affirmatively with some cost c we have a solution and may also conclude any cost in *forward cone* of c set $B^+(c) = \{c' \mid c' \geq c\}$ is feasible, which follows directly from the cost constraints. If the answer is negative we can conclude that any cost in the *backward cone* $B^-(c) = \{c' \mid c' \leq c\}$ is infeasible. After submitting any number of queries with different values of c we face a situation illustrated in Fig. 3. The sets $\overline{\mathcal{C}}_0$ and $\underline{\mathcal{C}}_1$ are, respectively, the maximal costs known to be infeasible (**unsat**) and minimal costs found (**sat**). Sets $\mathcal{C}_0$ and $\mathcal{C}_1$ are defined as the sets of all points known to be **unsat** and **sat**, they are equal to the forward/backward cone of the extremal points. The feasibility of costs which are outside $\mathcal{C}_0 \cup \mathcal{C}_1$ is unknown. The set $\underline{\mathcal{C}}_1$ constitutes an approximation of the Pareto front and its quality, defined as a kind of Hausdorff distance to the actual front, is bounded by its distance to the boundary of the backward cone of $\overline{\mathcal{C}}_0$.

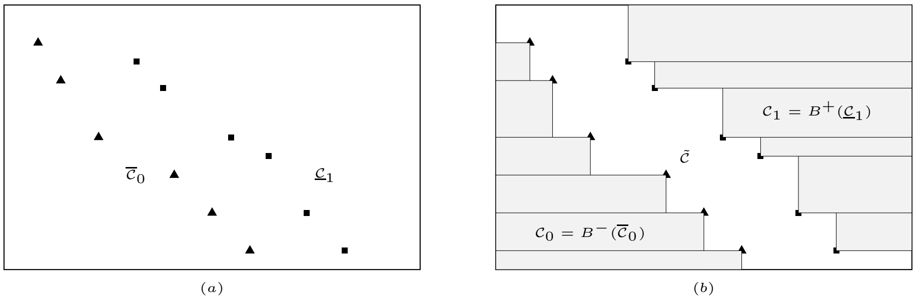

$\overline{\mathcal{C}}_0$
$\underline{\mathcal{C}}_1$
(a)
$\mathcal{C}_1 = B^+(\underline{\mathcal{C}}_1)$
$\tilde{\mathcal{C}}$
$\mathcal{C}_0 = B^-(\overline{\mathcal{C}}_0)$
(b)

Fig. 3. (a) Sets $\mathcal{C}_0$ (**unsat**) and $\mathcal{C}_1$ (**sat**) represented by their extremal points $\overline{\mathcal{C}}_0$ and $\underline{\mathcal{C}}_1$; (b) The state of our knowledge at this point as captured by $\mathcal{C}_0$ (infeasible costs) $\mathcal{C}_1$ (feasible costs) and $\tilde{\mathcal{C}}$ (unknown). The actual Pareto front is contained in the closure of $\tilde{\mathcal{C}}$.

Before we apply the exploration procedure we need to bound the cost space. For latency C_l, a lower bound is the size of the the longest path (in terms of δ) through the task graph. The upper bound is the total amount of work (sum of δ over all tasks). The bounds on buffers size are obtained by the two extreme scenarios. The lower bound is when each buffer is filled by the writer(s) to the minimal level required by the reader(s) to execute, that is, $B_{v,v'} = \alpha w(v, v')$ for an edge with multiplicity α or $1/\alpha$.

The upper bound should cover the execution of all instances of v before any instance of v', $B_{v,v'} = w(v,v') \cdot \max(c(v), c(v'))$. The number of processors ranges trivially between 1 and the maximal number of processors on the platform. The width of the task-graph, when smaller than the number of processors, can serve as a tighter upper-bound as it limits the number of tasks that can execute in parallel.

Unlike the distance-oriented algorithm of [14], we use here a simpler exploration algorithm based on grid refinement. At every stage of the algorithm we refine the grid defined on the cost space and ask $Q(c)$-queries with c ranging over those newly-added grid points which are outside $\mathcal{C}_0 \cup \mathcal{C}_1$. Note that not all these new points will necessarily be queried because each query increases the size of $\mathcal{C}_0 \cup \mathcal{C}_1$ so as to include some of these points. The description so far was based on the assumption that all queries terminate. However it is well-known that as c gets closer to the boundary between **sat** and **unsat**, the computation time may grow prohibitively and the solver can get stuck. To tackle this problem we bound the time budget per query and when this bound is reached we abort the query and interpret the result as **unsat**. Choosing the appropriate value for this time-out bound is a matter of trial and error.

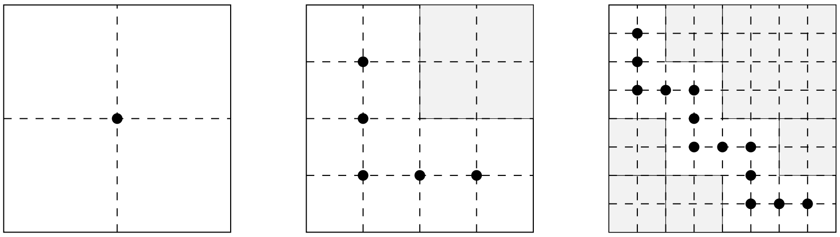
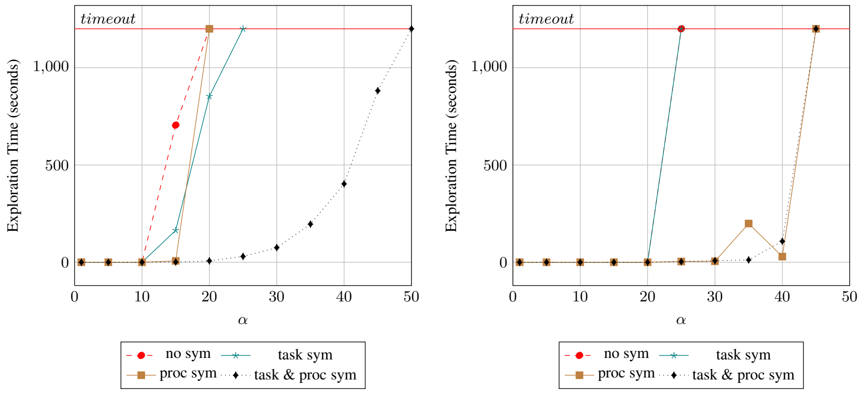


Fig. 4. Exploring the cost space via grid refinement. The dark points indicate the new candidates for exploration after each refinement.

4 Experiments

In this section we investigate the performance of the cost-space exploration algorithm. First, we assess the contribution of the symmetry reduction constraints on the execution time and the quality of solutions for a synthetic example. Then we explore the cost space for a split-join graph derived from a real video application. These experiments use version 4.1 of the Z3 Solver [17] running on a Linux machine with *Intel Core i7* processor at 1.73 GHz with 4 GB of memory. Finally, we validate the model used to derive the solution by deploying a JPEG decoder on the Tilera platform [25] according to the derived schedule. The measured performance is very close to the predicted model.

Finding Optimal Latency: We use the split-join graph of Fig. 1 with various values of α to explore the effect of the symmetry reduction constraints on the performance of the solver. We start with a single cost version of the problem and perform binary search to find the minimum latency that can be achieved for a fixed number of processors. We solve the same problem using four variations of the constraints: without symmetry reduction, with processor symmetry, with task symmetry and with both. Figure 5 depicts the computation times for finding the optimal latency as a function of α on platforms with 5 and 20 processors. We use time-out per query of 20 minutes, which is much larger than the one minute we typically use because we want to find the *exact* optimum in order to compare the effects of different symmetry constraints. Scheduling problems are known to be easy when the number of processors approaches the number of tasks. For the difficult case of 5 processors, task symmetry starts dominating beyond 10 tasks and the combination of both gives the best results. It increases the size of graphs whose optimal latency can be found (with no query executing more than 20 minutes) from $\alpha = 12$ to $\alpha = 48$. Not surprisingly, for 20 processors, the relative importance of processor symmetry grows. In Figure 5(b) we see no advantage from the task symmetry presumably because we could not try large values of α.

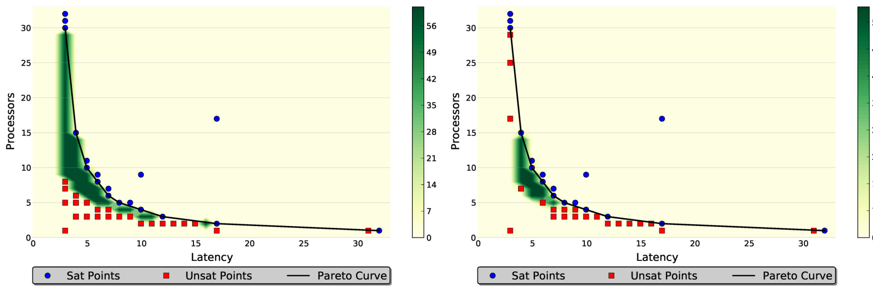


(a) 5-processor deployments (b) 20-processor deployments

Fig. 5. Time to find optimal latency as a function of the number of tasks for 5 and 20 processors

Processor-Latency Trade-Offs: To demonstrate the effect of symmetry reductions on the Pareto front exploration we fix $\alpha = 30$ and seek trade-offs between latency and the number of processors. We use a time budget of one minute per query. Fig. 6 depicts the results obtained with and without symmetry breaking constraints. The square points show the **unsat** points whereas the circle are the **sat** points. The black curve is the approximation of the Pareto front, connecting all the minimal **sat** points. Points whose queries took long time to answer are surrounded by a dark halo whose intensity is proportional to the time (the darkest areas are around the **timeout** points). As one can see from the figure, symmetry constraints reduce significantly the number of time-outs with processor symmetry doing the job on the upper-left part of the curve while task symmetry is useful around the middle. The total time to find the minimal latency for each and every value of C_n is 42 minutes without symmetry, and 16 minutes with both types of symmetry constraints.

Video Decoder: Next we perform a 3-dimensional cost exploration for a model of a video decoder taken from [10] and described in more detail in [22]. The application admits 11 actors expanding to 122 tasks. Without any symmetry constraints the solver quickly times out for most queries of interest. Symmetry constraints do not completely eliminate time-outs but reduce them significantly and therefore the quality of the Pareto



(a) Without Symmetry Constraints (b) With Task and Proc Symmetry

Fig. 6. Pareto Exploration Result

front approximation is much better, as shown in Fig. 7. Note that for a sequential implementation ($C_n = 1$) the constraints improve the buffer size from 276 to 182 and for the most parallel deployment ($C_n = 122$) they reduce the latency from 10 K to 7 K and the buffer size from 333 to 229. The Pareto point $(14, 333, 62)$ found without symmetry reduction is strongly dominated by the point $(10, 229, 31)$ found with symmetry breaking. This solution improves the latency and buffer usage by roughly a third while using half of the processors. We believe it is a promising indication of the applicability of our approach and of the potential performance gains in treating the optimization problem globally.

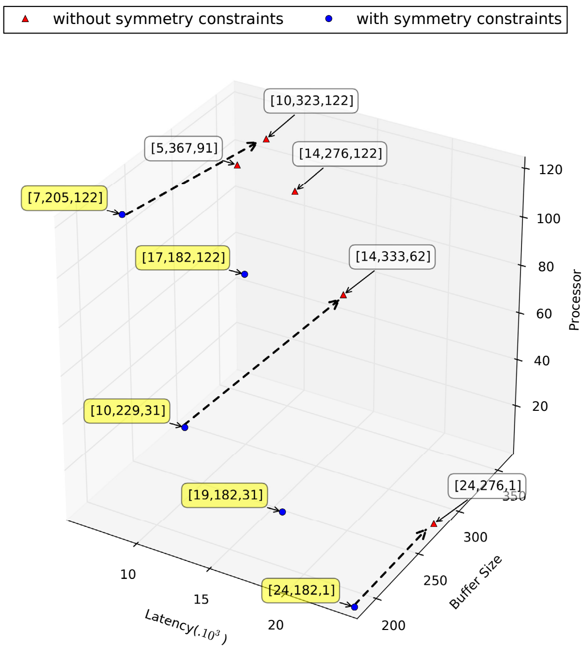


Fig. 7. Video Decoder Pareto Points

Jpeg Decoder: Finally we validate our model by deploying a JPEG decoder on the Tilera platform [25] which is a 64-core symmetric multicore platform running at 862.5 MHz. The theoretical scheduling problem that we solve is *deterministic* where task durations are assumed to be precisely known. The obtained schedule is time-triggered, given in terms of the *exact* start time function s. In reality, there are variations in execution times and in our implementation we use static order schedules, preserving only the *order* of task execution on each processor. This is a common way to generate schedules for task graphs and SDF, see for example [20]. When task durations agree with the nominal values used in the optimization problem, this scheduling policy coincides with s. Unlike the traditional work on dataflow mapping, we support mappings where the writers and readers of the same buffer storage can be spread over more than two different processors. Our experience confirms that this dynamic scheduling policy can be implemented with a reasonable amount of additional synchronization between the cores. Note also that when the schedule is compatible with lexicographical task order (justified by Theorem 1), the accesses to the channels automatically become FIFO and this facilitates the implementation of cyclic buffers.

The split-join graph of the decoder can be found in in [22]. It has three main actors: variable length decoding (vld), inverse quantization and inverse discrete cosine transform ($iq/idct$) combined and color conversion ($color$). To measure execution times we run the decoder several times on a single processor and measure the execution time of each actor. To mitigate cache effects, we consider the average execution time rather than worst case, which occurs only in the first execution due to cache misses. We use these average execution times in the model we submit to the solver. We then deploy the decoder on the platform and run it 100 times (again to dampen cache effects). The relation between the average latency (in μs) observed experimentally and the Pareto points computed by the SMT solver is depicted below and the deviation is typically smaller than 15%.

no. proc	1	3	4	6	8	12
predicted	506	314	278	261	243	226
measured	461	309	299	307	300	351

5 Discussion

The deployment of programs on parallel machines is a very old problem whose parameters change with the evolution of computer architecture. The problem exists in both software [12] and hardware [5] and in the latter it is viewed as an instance of *high-level synthesis* . Due to problem complexity the problem is often solved using heuristics such as list scheduling and/or decomposed into separate phases, for example, optimizing latency and buffers separately [21]. Recent advances in SAT and SMT solvers and other constraint propagation techniques suggest an opportunity to formulate and solve the problem in a monolithic way, avoiding the sub-optimality of decomposed solutions. For example, [15] exploits SMT solvers to combine multiple deployment sub-problems: the task-to-processor assignment, the ordering of tasks on each processor and the assignment of scalable voltage per processor. For SDF graphs, [2] and [27] combine multiple phases using a *constraint programming* (CP) engine. In the context of high-level synthesis, the tool FACTS (see [9] for references) uses branch-and-bound approach combined with constraint analysis, whereas [5] discusses various ILP formulations. In [26] a quantitative model checking engine is developed using a variant of timed automata for combined scheduling and buffer storage optimization of SDF graphs.

Various approaches to facilitate the task of the solver by additional symmetry breaking constraints have been tried, for example [19] for graph coloring or an automated method for discovering graph automorphism [4] which can lead to significant improvements [9]. However, our deployment problem does not require complex detection of isomorphic subgraphs. Instead we exploit the knowledge about the structure of the task graphs coming from the original split-join graph and not relying in any way on the graph automorphism. In fact, our approach leads to stronger symmetry reduction than could be obtained by exploiting the automorphism in the task graph as done in [9]. Theorem 1 provides the necessary compact symmetry breaking constraints that do the job. As for the restrictions that we imposed on the split-join graph compared to more general SDF graphs admitting non-divisible token production and consumption rate, let us first remark that Theorem 1 can be extended, somewhat less elegantly, to this more general case. Moreover, the extensive study of StreaMIT benchmarks found in [23] reports that most actors in most applications, fall into the category of well-formed split-join graphs that we treat.

The contribution of the paper can be summarized as follows. We provide a framework for multi-criteria optimization and cost-space exploration, not based on heuristic sub-optimal decomposition. Using symmetry reduction justified by Theorem 1, we could conduct a 3-dimensional cost-space exploration for a non-trivial problem with 122 tasks. The theorem itself generalizes the result of [9] which proves optimality of lexicographic order for one level of nesting. We prove the result for arbitrary nesting depth and give a simpler proof. In the future it might be interesting to apply this result in various alternative solution space exploration methods for the scheduling problems, *e.g.,* ILP, model checking or genetic programming.

In future, we plan to extend this work in several directions. First we will employ more refined models of data communication where different mappings imply different data transfer costs. Secondly we will consider *pipelined* executions as was done in [15,2,27,26], using *e.g.,* a finite unfolding. This will increase the number of tasks but will reduce the effect of precedence constraints. Thirdly we should adapt the methodology to a more significant variability in task duration and this will require an implementation of scheduling under uncertainty that can deviate from the task execution order provided by s. Finally we will seek ways for a more direct exploitation of the symbolic representation of data-parallel tasks and a tighter interaction between the cost exploration algorithm and the solver.

References

1. Baptiste, P., Le Pape, C., Nuijten, W.: Constraint-Based Scheduling. Kluwer international series in engineering and computer science. Kluwer (2001)
2. Bonfietti, A., Benini, L., Lombardi, M., Milano, M.: An efficient and complete approach for throughput-maximal SDF allocation and scheduling on multi-core platforms. In: DATE, pp. 897–902. IEEE (2010)
3. Coffman, E.G.: Computer and job-shop scheduling theory. Wiley (1976)

4. Darga, P.T., Sakallah, K.A., Markov, I.L.: Faster symmetry discovery using sparsity of symmetries. DAC, pp. 149–154. ACM, New York (2008)
5. De Micheli, G.: Synthesis and optimization of digital circuits. Electrical and Computer Engineering Series. McGraw-Hill Higher Education (1994)
6. Deb, K.: Multi-Objective Optimization Using Evolutionary Algorithms. Wiley paperback series. Wiley (2009)
7. Dutertre, B., de Moura, L.: A fast linear-arithmetic solver for DPLL(T). In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 81–94. Springer, Heidelberg (2006)
8. Ehrgott, M.: Multicriteria Optimization. Springer, Heidelberg (2005)
9. van Eijk, C.A.J., Jacobs, E.T.A.F., Mesman, B., Timmer, A.H.: Identification and exploitation of symmetries in DSP algorithms. In: DATE. IEEE, New York (1999)
10. Fradet, P., Girault, A., Poplavko, P.: SPDF: A schedulable parametric data-flow MoC. In: DATE, pp. 769–774. IEEE (2012)
11. Kalray: Kalray MPPA 256, `http://www.kalray.eu/`
12. Kwok, Y.K., Ahmad, I.: Static scheduling algorithms for allocating directed task graphs to multiprocessors. ACM Comput. Surv. 31(4), 406–471 (1999)
13. Lee, E., Messerschmitt, D.: Synchronous data flow. IEEE 75(9), 1235–1245 (1987)
14. Legriel, J., Le Guernic, C., Cotton, S., Maler, O.: Approximating the Pareto front of multicriteria optimization problems. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 69–83. Springer, Heidelberg (2010)
15. Legriel, J., Maler, O.: Meeting deadlines cheaply. In: ECRTS, pp. 185–194. IEEE (2011)
16. Melpignano, D., Benini, L., Flamand, E., Jego, B., Lepley, T., Haugou, G., Clermidy, F., Dutoit, D.: Platform 2012, a many-core computing accelerator for embedded SoCs: performance evaluation of visual analytics applications. DAC, pp. 1137–1142. ACM, USA (2012)
17. de Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008)
18. Pareto, V.: Manuel d'économie politique. Bull. Amer. Math. Soc. 18, 462–474 (1912)
19. Ramani, A., Aloul, F., Markov, I., Sakallah, K.: Breaking instance-independent symmetries in exact graph coloring. In: DATE, vol. 1, pp. 324–329. IEEE (2004)
20. Sriram, S., Bhattacharyya, S.: Embedded Multiprocessors: Scheduling and Synchronization, 2nd edn. Signal Processing and Communications. Taylor & Francis (2009)
21. Stuijk, S., Geilen, M., Basten, T.: Exploring trade-offs in buffer requirements and throughput constraints for synchronous dataflow graphs. DAC, pp. 899–904. ACM, USA (2006)
22. Tendulkar, P., Poplavko, P., Maler, O.: Symmetry breaking for multi-criteria mapping and scheduling on multicores. technical report TR-2013-3, Verimag (2013)
23. Thies, W., Amarasinghe, S.: An empirical characterization of stream programs and its implications for language and compiler design. PACT, pp. 365–376. ACM, USA (2010)
24. Thies, W., Karczmarek, M., Amarasinghe, S.: StreamIt: A language for streaming applications. In: Nigel Horspool, R. (ed.) CC 2002. LNCS, vol. 2304, pp. 179–196. Springer, Heidelberg (2002)
25. Tilera, LTD: Tilera TILE64 processor, `http://www.tilera.com/`
26. Yang, Y., Geilen, M., Basten, T., Stuijk, S., Corporaal, H.: Exploring trade-offs between performance and resource requirements for synchronous dataflow graphs. In: ESTImedia, pp. 96–105 (2009)
27. Zhu, J., Sander, I., Jantsch, A.: Buffer minimization of real-time streaming applications scheduling on hybrid CPU/FPGA architectures. DATE, pp. 1506–1511. IEEE, Belgium (2009)
28. Zitzler, E., Thiele, L.: Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach. IEEE Transactions on Evolutionary Computation 3(4), 257–271 (1999)

Confluence Reduction for Markov Automata

Mark Timmer, Jaco van de Pol, and Mariëlle I.A. Stoelinga*

Formal Methods and Tools, Faculty of EEMCS
University of Twente, The Netherlands
{timmer,vdpol,marielle}@cs.utwente.nl

Abstract. Markov automata are a novel formalism for specifying systems exhibiting nondeterminism, probabilistic choices and Markovian rates. Recently, the process algebra MAPA was introduced to efficiently model such systems. As always, the state space explosion threatens the analysability of the models generated by such specifications. We therefore introduce confluence reduction for Markov automata, a powerful reduction technique to keep these models small. We define the notion of confluence directly on Markov automata, and discuss how to syntactically detect confluence on the MAPA language as well. That way, Markov automata generated by MAPA specifications can be reduced on-the-fly while preserving divergence-sensitive branching bisimulation. Three case studies demonstrate the significance of our approach, with reductions in analysis time up to an order of magnitude.

1 Introduction

Over the past two decades, model checking algorithms were generalised to handle more and more expressive models. This now allows us to verify probabilistic as well as hard and soft real-time systems, modelled by timed automata, Markov decision processes, probabilistic automata, continuous-time Markov chains, interactive Markov chains, and Markov automata. Except for timed automata—which incorporate real-time deadlines—all other models are subsumed by the Markov automaton (MA) [14, 13, 12]. MAs can therefore be used as a semantic model for a wide range of formalisms, such as generalised stochastic Petri nets (GSPNs) [2], dynamic fault trees [9], Arcade [8] and the domain-specific language AADL [10].

Before the introduction of MAs, the above models could not be analysed to their full extent. For instance, the semantics of a (potentially nondeterministic) GSPN were given as a fully probabilistic CTMC. To this end, weights had to be assigned to resolve the nondeterminism between immediate transitions. As argued in [20], it is often much more natural to omit most of these weights, retaining rates and probability as well as nondeterminism, and thus obtaining an MA. For example, consider the GSPN in Figure 1(a), taken from [13]. Immediate

* This research has been partially funded by NWO under grants 612.063.817 (SYRUP), 12238 (ArRangeer) and Dn 63-257 (ROCKS), and the EU under grants 318490 (SENSATION) and 318003 (TREsPASS).

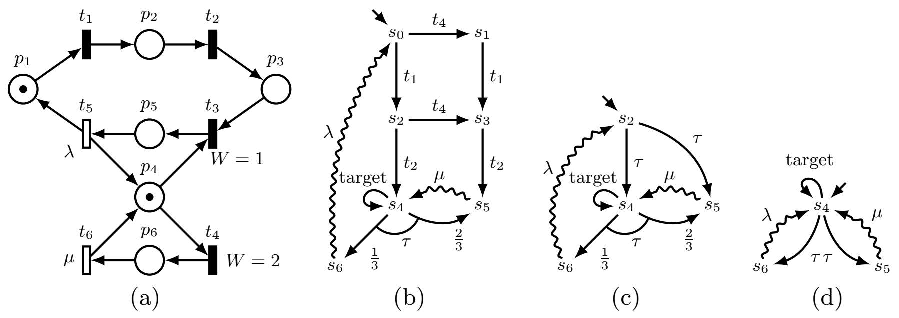


Fig. 1. A GSPN and the corresponding unreduced and reduced state spaces. For the reduced model in (d) the weights of transitions t_3 and t_4 are assumed to be absent.

transitions are indicated in black, Markovian transitions in white, and we assume a partial weight assignment. The underlying MA is given in Figure 1(b), where s_0 corresponds to the initial situation with one token in p_1 and p_4. We added a selfloop labelled *target* to indicate a possible state of interest s_4 (having one token in p_3 and p_4), and for convenience labelled the interactive transitions of the MA by the immediate transition of the GSPN they resulted from (except for the probabilistic transition, which is the result of t_3 and t_4 together).

Recently, the data-rich process-algebraic language MAPA was introduced to efficiently specify MAs in a compositional manner [23]. As always, though, the state space explosion threatens the feasibility of model checking, especially in the presence of data and interleaving. Therefore, reduction techniques for MAs are vital to keep the state spaces of these models manageable. In this paper we introduce such a technique, generalising *confluence reduction* to MAs. It is a powerful state space reduction technique based on commutativity of transitions, removing spurious nondeterminism often arising from the parallel composition of largely independent components. Basically, *confluent transitions* never disable behaviour, since all transitions enabled from their source states can be *mimicked* from their target states. To the best of our knowledge, it is the first technique of this kind for MAs. We give heuristics to apply confluence reduction directly on specifications in the MAPA language, reducing them on-the-fly while preserving divergence-sensitive branching bisimulation.

To illustrate confluence reduction, reconsider the MA in Figure 1(b) and assume that $t_1 = t_2 = t_4 = \tau$, i.e., all action-labelled transitions, except for the *target*-transition, are invisible. We are able to detect automatically that the t_1-transitions are confluent; they can thus safely be given priority over t_4, without losing any behaviour. Figure 1(c) shows the reduced state space, generated on-the-fly using confluence reduction. If all weights are omitted from the specification, an even smaller reduced state space is obtained (Figure 1(d)), while the only change in the unreduced state space is the substitution of the probabilistic choice by a nondeterministic choice.

Outline of the Approach. First, we introduce the technical background of our work (Section 2). Then, we define our novel notion of confluence for MAs (Section 3). It specifies sufficient conditions for invisible transitions to not alter the behaviour of an MA; i.e., if a transition is confluent, it could be given priority over all other transitions with the same source state.

We formally show that confluent transitions connect divergence-sensitive branching bisimilar states, and present a mapping of states to representatives to efficiently generate a reduced MA based on confluence (Section 4). We discuss how confluence can be detected symbolically on specifications in the MAPA language (Section 5) and illustrate the significance of our technique using three case studies (Section 6). We show state spaces shrinking by more than 80%, making the entire process from MAPA specification to results more than ten times as fast for some models.[1]

Related Work. Confluence reduction for process algebras was first introduced for non-probabilistic systems [7], and later for probabilistic automata [24]. Also, several types of *partial order reduction* (POR) have been defined, both for non-probabilistic [26, 21, 16] and probabilistic systems [11, 4, 3]. These techniques are based on ideas similar to confluence, and have been compared to confluence recently, both in a theoretical [18] and in a practical manner [19]. The results showed that branching-time POR is strictly subsumed by confluence, and that the additional advantages of confluence can be employed nicely in the context of statistical model checking.

Compared to the earlier approaches to confluence reduction for process algebras [7, 24], our novel notion of confluence is different in three important ways:

- It can handle MAs, and hence is applicable to a larger class of systems.
- It fixes a subtle flaw in the earlier papers, which did not guarantee closure under unions. We solve this by introducing an underlying classification of the interactive transitions. This way we do guarantee closure under unions, a key requirement for the way we detect confluence on MAPA specifications.
- It preserves divergences and hence minimal reachability probabilities, incorporating a technique used earlier in [18].

Since none of the existing techniques is able to deal with MAs, we believe that our generalisation—the first reduction technique for MAs abstracting from internal transitions—is a major step forward in efficient quantitative verification.

2 Preliminaries

Definition 1 (Basics). *A* probability distribution *over a countable set S is a function $\mu\colon S \to [0,1]$ such that $\sum_{s \in S} \mu(s) = 1$. For $S' \subseteq S$, let $\mu(S') = \sum_{s \in S'} \mu(s)$. We define* $\mathrm{spt}(\mu) = \{s \in S \mid \mu(s) > 0\}$ *to be the* support *of μ, and write $\mathbb{1}_s$ for the* Dirac distribution *for s, determined by $\mathbb{1}_s(s) = 1$.*

[1] Due to space limitations, we discuss the notion of divergence-sensitive branching bisimulation only on an intuitive level, deferring the formal definitions and proofs of all our results to a technical report [25].

We use Distr(S) *to denote the set of all probability distributions over* S, *and* SDistr(S) *for the set of all* substochastic *probability distributions over* S, *i.e., where* $0 \leq \sum_{s \in S} \mu(s) \leq 1$. *Given a function* f, *we denote by* $f(\mu)$ *the* lifting *of* μ *over* f, *i.e.,* $f(\mu)(s) = \mu(f^{-1}(s))$, *with* $f^{-1}(s)$ *the inverse image of* s *under* f.

Given an equivalence relation $R \subseteq S \times S$, *we write* $[s]_R$ *for the equivalence class of* s *induced by* R, *i.e.,* $[s]_R = \{s' \in S \mid (s, s') \in R\}$. *Given two probability distributions* $\mu, \mu' \in$ Distr(S) *and an equivalence relation* R, *we write* $\mu \equiv_R \mu'$ *to denote that* $\mu([s]_R) = \mu'([s]_R)$ *for every* $s \in S$.

An MA is a transition system in which the set of transitions is partitioned into probabilistic action-labelled interactive transitions (equivalent to the transitions of a PA), and Markovian transitions labelled by the rate of an exponential distribution (equivalent to the transitions of a CTMC). We assume a countable universe of actions *Act*, with $\tau \in Act$ the invisible internal action.

Definition 2 (Markov automata). *A* Markov automaton (MA) *is a tuple* $\mathcal{M} = \langle S, s^0, A, \hookrightarrow, \rightsquigarrow \rangle$, *where*

- S *is a countable set of* states, *of which* $s^0 \in S$ *is the* initial state;
- $A \subseteq Act$ *is a countable set of* actions;
- $\hookrightarrow \subseteq S \times A \times \text{Distr}(S)$ *is the* interactive transition relation;
- $\rightsquigarrow \subseteq S \times \mathbb{R}_{>0} \times S$ *is the* Markovian transition relation.

If $(s, a, \mu) \in \hookrightarrow$, *we write* $s \overset{a}{\hookrightarrow} \mu$ *and say that the action* a *can be* executed *from state* s, *after which the probability to go to each* $s' \in S$ *is* $\mu(s')$. *If* $(s, \lambda, s') \in \rightsquigarrow$, *we write* $s \overset{\lambda}{\rightsquigarrow} s'$ *and say that* s *moves to* s' *with rate* λ.

The *rate* between two states $s, s' \in S$ is $\mathit{rate}(s, s') = \sum_{(s,\lambda,s') \in \rightsquigarrow} \lambda$, and the *outgoing rate of* s is $\mathit{rate}(s) = \sum_{s' \in S} \mathit{rate}(s, s')$. We require $\mathit{rate}(s) < \infty$ for every state $s \in S$. If $\mathit{rate}(s) > 0$, the *branching probability distribution* after this delay is denoted by $\mathbb{P}_s$ and defined by $\mathbb{P}_s(s') = \frac{\mathit{rate}(s,s')}{\mathit{rate}(s)}$ for every $s' \in S$.

By definition of the exponential distribution, the probability of leaving a state s within t time units is given by $1 - e^{-\mathit{rate}(s) \cdot t}$ (given $\mathit{rate}(s) > 0$), after which the next state is chosen according to $\mathbb{P}_s$.

MAs adhere to the *maximal progress assumption*, prescribing τ-transitions to never be delayed. Hence, a state that has at least one outgoing τ-transition can never take a Markovian transition. This fact is captured below in the definition of extended transitions, which is used to provide a uniform manner for dealing with both interactive and Markovian transitions.

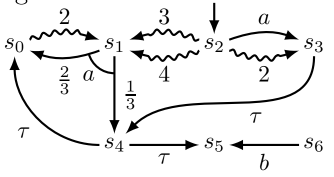
Definition 3 (Extended action set). *Let* $\mathcal{M} = \langle S, s^0, A, \hookrightarrow, \rightsquigarrow \rangle$ *be an MA, then the* extended action set of $\mathcal{M}$ *is given by* $A^\chi = A \cup \{\chi(r) \mid r \in \mathbb{R}_{>0}\}$. *Given a state* $s \in S$ *and an action* $\alpha \in A^\chi$, *we write* $s \xrightarrow{\alpha} \mu$ *if either*

- $\alpha \in A$ *and* $s \overset{\alpha}{\hookrightarrow} \mu$, *or*
- $\alpha = \chi(\mathit{rate}(s))$, $\mathit{rate}(s) > 0$, $\mu = \mathbb{P}_s$ *and there is no* μ' *such that* $s \overset{\tau}{\hookrightarrow} \mu'$.

A transition $s \xrightarrow{\alpha} \mu$ *is called an* extended transition. *We use* $s \xrightarrow{\alpha} t$ *to denote* $s \xrightarrow{\alpha} \mathbb{1}_t$, *and write* $s \rightarrow t$ *if there is at least one action* α *such that* $s \xrightarrow{\alpha} t$. *We write* $s \xrightarrow{\alpha,\mu} s'$ *if there is an extended transition* $s \xrightarrow{\alpha} \mu$ *such that* $\mu(s') > 0$.

Note that each state has an extended transition per interactive transition, while it has only one extended transition for all its Markovian transitions together (if there are any).

Example 4. Consider the MA $\mathcal{M}$ shown on the right. For this system, $rate(s_2, s_1) = 3 + 4 = 7$, $rate(s_2) = 7 + 2 = 9$, and $\mathbb{P}_{s_2} = \mu$ such that $\mu(s_1) = \frac{7}{9}$ and $\mu(s_3) = \frac{2}{9}$. There are two extended transitions from s_2: $s_2 \xrightarrow{a} \mathbb{1}_{s_3}$ (also written as $s_2 \xrightarrow{a} s_3$) and $s_2 \xrightarrow{\chi(9)} \mathbb{P}_{s_2}$.



□

We define several notions for paths and connectivity. These are based on extended transitions, and thus may contain interactive as well as Markovian steps.

Definition 5 (Paths). *Given an MA* $\mathcal{M} = \langle S, s^0, A, \hookrightarrow, \rightsquigarrow \rangle$,

- *A* path *in* $\mathcal{M}$ *is a finite sequence* $\pi^{\text{fin}} = s_0 \xrightarrow{a_1,\mu_1} s_1 \xrightarrow{a_2,\mu_2} \dots \xrightarrow{a_n,\mu_n} s_n$ *from some state* s_0 *to a state* s_n *(*$n \geq 0$*), or an infinite sequence* $\pi^{\text{inf}} = s_0 \xrightarrow{a_1,\mu_1} s_1 \xrightarrow{a_2,\mu_2} s_2 \xrightarrow{a_3,\mu_3} \dots$*, with* $s_i \in S$ *for all* $0 \leq i \leq n$ *and all* $0 \leq i$*, respectively. We use* $\mathit{prefix}(\pi, i)$ *to denote* $s_0 \xrightarrow{a_1,\mu_1} \dots \xrightarrow{a_i,\mu_i} s_i$*, and* $\mathit{step}(\pi, i)$ *for the transition* $s_{i-1} \xrightarrow{a_i} \mu_i$*. When* π *is finite we define* $|\pi| = n$ *and* $\mathit{last}(\pi) = s_n$*. We use* $\mathit{finpaths}_{\mathcal{M}}$ *for the set of all* finite paths *in* $\mathcal{M}$ *(not necessarily starting in the initial state* s^0*), and* $\mathit{finpaths}_{\mathcal{M}}(s)$ *for all such paths with* $s_0 = s$*.*
- *We denote by* $\mathit{trace}(\pi)$ *the sequence of actions of* π *while omitting all* τ*-actions, and use* ϵ *to denote the empty sequence.*

Definition 6 (Connectivity). *Let* $\mathcal{M} = \langle S, s^0, A, \hookrightarrow, \rightsquigarrow \rangle$ *be an MA,* $s, t \in S$*, and consider again the binary relation* $\rightarrow \subseteq S \times S$ *from Definition 3 that relates states* $s, t \in S$ *if there is a transition* $s \xrightarrow{\alpha} \mathbb{1}_t$ *for some* α*.*

We let $\twoheadrightarrow$ (reachability) *be the reflexive and transitive closure of* $\rightarrow$*, and we let* $\twoheadleftarrow\!\!\twoheadrightarrow$ (convertibility) *be its reflexive, transitive and symmetric closure. We write* $s \twoheadrightarrow\!\!\twoheadleftarrow t$ (joinability) *if there is a state* u *such that* $s \twoheadrightarrow u$ *and* $t \twoheadrightarrow u$*.*

Note that the relation $\twoheadrightarrow\!\!\twoheadleftarrow$ is symmetric, but not necessarily transitive. Also note that, intuitively, $s \twoheadleftarrow\!\!\twoheadrightarrow t$ means that s is connected by extended transitions to t—disregarding the orientation of these transitions, but requiring them all to have a Dirac distribution.

Clearly, $s \twoheadrightarrow t$ implies $s \twoheadrightarrow\!\!\twoheadleftarrow t$, and $s \twoheadrightarrow\!\!\twoheadleftarrow t$ implies $s \twoheadleftarrow\!\!\twoheadrightarrow t$. These implications do not hold the other way.

Example 7. The system in Example 4 has infinitely many paths, for example

$$\pi = s_2 \xrightarrow{\chi(9),\mu_1} s_1 \xrightarrow{a,\mu_2} s_0 \xrightarrow{\chi(2),\mathbb{1}_{s_1}} s_1 \xrightarrow{a,\mu_2} s_4 \xrightarrow{\tau,\mathbb{1}_{s_5}} s_5$$

with $\mu_1(s_1) = \frac{7}{9}$ and $\mu_1(s_3) = \frac{2}{9}$, and $\mu_2(s_0) = \frac{2}{3}$ and $\mu_2(s_4) = \frac{1}{3}$. We have $\mathit{prefix}(\pi, 2) = s_2 \xrightarrow{\chi(9),\mu_1} s_1 \xrightarrow{a,\mu_2} s_0$, and $\mathit{step}(\pi, 2) = s_1 \xrightarrow{a} \mu_2$. Also, $\mathit{trace}(\pi) = \chi(9)\, a\, \chi(2)\, a$. It is easy to see that $s_2 \twoheadrightarrow s_5$ (via s_3), as well as $s_3 \twoheadrightarrow\!\!\twoheadleftarrow s_6$ (at s_5) and $s_0 \twoheadleftarrow\!\!\twoheadrightarrow s_5$. However, $s_0 \twoheadrightarrow s_5$ and $s_0 \twoheadrightarrow\!\!\twoheadleftarrow s_5$ do not hold. □

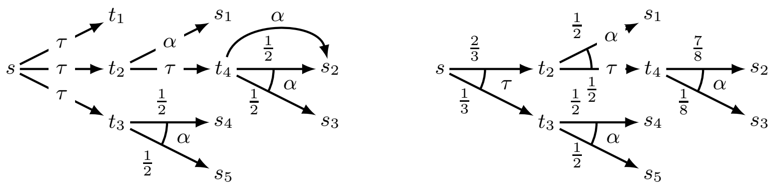


Fig. 2. An MA (left), and a tree demonstrating branching transition $s \overset{\alpha}{\Longrightarrow}_R \mu$ (right)

2.1 Divergence-Sensitive Branching Bisimulation

To prove our confluence reduction technique correct, we show that it preserves divergence-sensitive branching bisimulation. Basically, this means that there is an equivalence relation R linking states in the original system to states in the reduced system, in such a way that their initial states are related and all related states can mimic each other's transitions and divergences.

More precisely, for R to be a divergence-sensitive branching bisimulation, it is required that for all $(s, t) \in R$ and every extended transition $s \xrightarrow{\alpha} \mu$, there is a *branching transition* $t \overset{\alpha}{\Longrightarrow}_R \mu'$ such that $\mu \equiv_R \mu'$. The existence of such a branching transition depends on the existence of a certain *scheduler*. Schedulers resolve nondeterministic choices in an MA by selecting which transitions to take given a history; they are also allowed to terminate with some probability.

Now, a state t can do a branching transition $t \overset{\alpha}{\Longrightarrow}_R \mu'$ if either (1) $\alpha = \tau$ and $\mu' = \mathbb{1}_t$, or (2) there exists a scheduler that terminates according to μ', always schedules precisely one α-transition (immediately before terminating), does not schedule any other visible transitions and does not leave the equivalence class $[t]_R$ before doing an a-transition.

Example 8. Observe the MA in Figure 2 (left). We find that $s \overset{\alpha}{\Longrightarrow}_R \mu$, with

$$\mu(s_1) = \tfrac{8}{24} \qquad \mu(s_2) = \tfrac{7}{24} \qquad \mu(s_3) = \tfrac{1}{24} \qquad \mu(s_4) = \tfrac{4}{24} \qquad \mu(s_5) = \tfrac{4}{24}$$

by the scheduling depicted in Figure 2 (right), assuming $(s, t_i) \in R$ for all t_i. □

In addition to the mimicking of transitions by branching transitions, we require R-related states to either both be able to perform an infinite invisible path with probability 1 (*diverge)*, or to both not be able to do so. We write $s \leftrightarrow_{\mathrm{b}}^{\mathrm{div}} t$ if two states s, t are divergence-sensitive branching bisimilar, and $\mathcal{M}_1 \leftrightarrow_{\mathrm{b}}^{\mathrm{div}} \mathcal{M}_2$ if two MAs are (i.e., if their initial states are so in their disjoint union).

3 Confluence for Markov Automata

In [24] we defined three variants of probabilistic confluence: weak probabilistic confluence, probabilistic confluence and strong probabilistic confluence. They specify sufficient conditions for τ-transitions to not alter the behaviour of an MA. The stronger notions are easier to detect, but less powerful in their reductions.

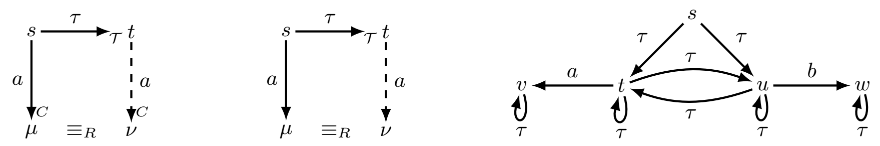
In a process-algebraic context, where confluence is detected heuristically over a syntactic description of a system, it is most practical to apply strong confluence. Therefore, in this paper we only generalise strong probabilistic confluence to the Markovian realm. Although MAs in addition to interactive transitions may also contain Markovian transitions, these are basically irrelevant for confluence. After all, states having a τ-transition can never execute a Markovian transition due to the maximal progress assumption. Hence, such transitions need not be mimicked. For the above reasons, the original definition of confluence for PAs might seem to still work for MAs. This is not true, however, for two reasons.

1. The old definition was not yet divergence sensitive. Therefore, Markovian transitions in an MA that are disabled by the maximal progress assumption, due to a divergence from the same state, may erroneously be enabled if that divergence is removed. Hence, the old notion does not even preserve Markovian divergence-*in*sensitive branching bisimulation. We now improve on the definition to resolve this issue, introducing τ-loops in the reduced system for states having confluent divergence in the original system (inspired by the way [18] deals with divergences). This not only makes the theory work for MAs, it even yields preservation of divergence-sensitive branching bisimulation, and hence of minimal reachability probabilities.
2. The old definition had a subtle flaw: earlier work relied on the assumption that confluent sets are closed under unions [7, 24]. In practical applications this was indeed a valid assumption, but for the theoretical notions of confluence this was not yet the case. We fix this flaw by classifying transitions into groups, defining confluence over sets of such groups and requiring transitions to be mimicked by a transition from their own group.

Additionally, compared to [7, 24] we improve on the way equivalence of distributions is defined, making it slightly more powerful and, in our view, easier to understand (inspired by the definitions in [19]).

Confluence Classifications and Confluent Sets. The original lack of closure under unions was due to the requirement that confluent transitions are mimicked by confluent transitions. When taking the union of two sets of confluent transitions, this requirement was possibly invalidated. To solve this problem, we classify the interactive transitions of an MA into groups—allowing overlap and not requiring all interactive transitions to be in at least one group. Together, we call such a set of groups $P = \{C_1, C_2, \ldots, C_n\} \subseteq \mathscr{P}(\hookrightarrow)$ a *confluence classification*[2]. Now, instead of designating individual transitions to be confluent and requiring confluent transitions to be mimicked by confluent transitions, we designate groups in P to be confluent (now called *Markovian confluent*) and require transitions from a group in P to be mimicked by transitions from the same group.

[2] We use $s \xrightarrow{a}_C \mu$ to denote that $(s \xrightarrow{a} \mu) \in C$, and abuse notation by writing $(s \xrightarrow{a} \mu) \in P$ to denote that $s \xrightarrow{a}_C \mu$ for some $C \in P$. Similarly, we subscript reachability, joinability and convertibility arrows to indicate that they only traverse transitions from a certain group or set of groups of transitions.



(a) $(s \xrightarrow{a} \mu) \in P$. (b) $(s \xrightarrow{a} \mu) \notin P$. (c) A simple state space.

Fig. 3. The confluence diagrams for $s \xrightarrow{\tau}_{\mathcal{T}} t$, and a simple state space. In (a,b): If the solid transitions are present, then so should the dashed ones be.

For a set $\mathcal{T} \subseteq P$ to be Markovian confluent, first of all—like in the PA setting [24, 3]—it is only allowed to contain invisible transitions with a Dirac distribution. (Still, giving priority to such transitions may very well reduce probabilistic transitions as well, as we will see in Section 4.) Additionally, each transition $s \xrightarrow{a} \mu$ enabled before a transition $s \xrightarrow{\tau}_{\mathcal{T}} t$ should have a mimicking transition $t \xrightarrow{a} \nu$ such that μ and ν are connected by $\mathcal{T}$-transitions, and mimicking transitions should be from the same group. The definition is illustrated in Figure 3.

Definition 9 (Markovian confluence). *Let $\mathcal{M} = \langle S, s^0, A, \hookrightarrow, \rightsquigarrow \rangle$ be an MA and $P \subseteq \mathscr{P}(\hookrightarrow)$ a confluence classification. Then, a set $\mathcal{T} \subseteq P$ is* Markovian confluent for P *if it only contains sets of invisible transitions with Dirac distributions, and for all $s \xrightarrow{\tau}_{\mathcal{T}} t$ and all transitions $(s \xrightarrow{a} \mu) \neq (s \xrightarrow{\tau} t)$:*

$$\begin{cases} \forall C \in P \,.\, s \xrightarrow{a}_C \mu \implies \exists \nu \in \mathrm{Distr}(S) \,.\, t \xrightarrow{a}_C \nu \ \wedge \mu \equiv_R \nu & \text{, if } (s \xrightarrow{a} \mu) \in P \\ \qquad\qquad\qquad\qquad\quad\ \exists \nu \in \mathrm{Distr}(S) \,.\, t \xrightarrow{a} \nu \ \wedge \mu \equiv_R \nu & \text{, if } (s \xrightarrow{a} \mu) \notin P \end{cases}$$

with R the smallest equivalence relation such that

$$R \supseteq \{(s,t) \in \mathrm{spt}(\mu) \times \mathrm{spt}(\nu) \mid (s \xrightarrow{\tau} t) \in \mathcal{T}\}.$$

A transition $s \xrightarrow{\tau} t$ is Markovian confluent *if there exists a Markovian confluent set $\mathcal{T}$ such that $s \xrightarrow{\tau}_{\mathcal{T}} t$. Often, we omit the adjective 'Markovian'.*

Note that $\mu \equiv_R \nu$ requires direct transitions from the support of μ to the support of ν. Also note that, even though a (symmetric) equivalence relation R is used, transitions from the support of ν to the support of μ do not influence R.

Remark 10. Due to the confluence classification, confluent transitions are always mimicked by confluent transitions. After all, transitions from a group $C \in P$ are mimicked by transitions from C. So, if C is designated confluent by $\mathcal{T}$, then all these confluent transitions are indeed mimicked by confluent transitions.

Although the confluence classification may appear restrictive, we will see that in practice it is obtained naturally. Transitions are often instantiations of higher-level constructs, and are therefore easily grouped together. Then, it makes sense to detect the confluence of such a higher-level construct. Additionally, to show that a certain set of invisible transitions is confluent, we can just take P to consist of one group containing precisely all those transitions. Then, the requirement for P-transitions to be mimicked by the same group reduces to the old requirement that confluent transitions are mimicked by confluent transitions.

Properties of Confluent Sets. Since confluent transitions are always mimicked by confluent transitions, confluent paths (i.e., paths following only transitions from a confluent set) are always joinable.

Proposition 11. *Let $\mathcal{M} = \langle S, s^0, A, \hookrightarrow, \rightsquigarrow \rangle$ be an MA, $P \subseteq \mathscr{P}(\hookrightarrow)$ a confluence classification for $\mathcal{M}$ and $\mathcal{T} \subseteq P$ a Markovian confluent set for P. Then,*

$$s \twoheadrightarrow\twoheadleftarrow_{\mathcal{T}} t \qquad \textit{if and only if} \qquad s \twoheadleftarrow\!\!\twoheadrightarrow_{\mathcal{T}} t$$

Due to the confluence classification, we now also do have a closure result. Closure under union tells us that it is safe to show confluence of multiple sets of transitions in isolation, and then just take their union as one confluent set. Also, it implies that there exists a unique maximal confluent set.

Theorem 12. *Let $\mathcal{M} = \langle S, s^0, A, \hookrightarrow, \rightsquigarrow \rangle$ be an MA, $P \subseteq \mathscr{P}(\hookrightarrow)$ a confluence classification for $\mathcal{M}$ and $\mathcal{T}_1, \mathcal{T}_2 \subseteq P$ two Markovian confluent sets for P. Then, $\mathcal{T}_1 \cup \mathcal{T}_2$ is also a Markovian confluent set for P.*

The next example shows why Theorem 12 would not hold without the use of a confluence classification. It applies to the old notions of confluence as well.

Example 13. Consider the system in Figure 3(c). Without the requirement that transitions are mimicked by the same group, the sets

$$\mathcal{T}_1 = \{(s,\tau,u),(t,\tau,t),(u,\tau,u),(v,\tau,v),(w,\tau,w)\}$$
$$\mathcal{T}_2 = \{(s,\tau,t),(t,\tau,t),(u,\tau,u),(v,\tau,v),(w,\tau,w)\}$$

would both be perfectly valid confluent sets. Still, $\mathcal{T} = \mathcal{T}_1 \cup \mathcal{T}_2$ is not an acceptable set. After all, whereas $t \twoheadleftarrow\!\!\twoheadrightarrow_{\mathcal{T}} u$, it fails to satisfy $t \twoheadrightarrow\twoheadleftarrow_{\mathcal{T}} u$. This property was ascertained in earlier work by requiring confluent transitions to be mimicked by confluent transitions or by explicitly requiring $\twoheadrightarrow\twoheadleftarrow_{\mathcal{T}}$ to be an equivalence relation. This is indeed not the case for $\mathcal{T}$, as the diamond starting with $s \xrightarrow{\tau} t$ and $s \xrightarrow{\tau} u$ can only be closed using the non-confluent transitions between t and u, and clearly $\twoheadrightarrow\twoheadleftarrow$ is not transitive. However, $\mathcal{T}_1$ and $\mathcal{T}_2$ do satisfy these requirements, and hence the old notions were not closed under union.

By using a confluence classification and requiring transitions to be mimicked by the same group, we ascertain that this kind of bad compositionality behaviour does not occur. After all, for $\mathcal{T}_1$ to be a valid confluent set, the confluence classification should be such that $s \xrightarrow{\tau} t$ and its mimicking transition $u \xrightarrow{\tau} t$ are in the same group. So, for $s \xrightarrow{\tau} t$ to be confluent (as prescribed by $\mathcal{T}_2$), also $u \xrightarrow{\tau} t$ would need to be confluent. The latter is impossible, since the b-transition from u cannot be mimicked from t, and hence $\mathcal{T}_2$ is disallowed. □

The final result of this section shows that confluent transitions indeed connect divergence-sensitive bisimilar states. This is a key result; it implies that confluent transitions can be given priority over other transitions without losing behaviour—when being careful not to indefinitely ignore any behaviour.

Theorem 14. *Let $\mathcal{M} = \langle S, s^0, A, \hookrightarrow, \rightsquigarrow \rangle$ be an MA, $s, s' \in S$ two of its states, $P \subseteq \mathscr{P}(\hookrightarrow)$ a confluence classification for $\mathcal{M}$ and $\mathcal{T} \subseteq P$ a Markovian confluent set for P. Then,*

$$s \twoheadleftarrow\!\!\twoheadrightarrow_{\mathcal{T}} s' \textit{ implies } s \leftrightarroweq_{\mathrm{b}}^{\mathrm{div}} s'.$$

4 State Space Reduction Using Confluence

We can reduce state spaces by giving priority to confluent transitions, i.e., by omitting all other transitions from a state that also enables a confluent transition (as long as no behaviour is ignored indefinitely). Better still, we aim at omitting all intermediate states on a confluent path altogether; after all, they are all bisimilar anyway by Theorem 14. Confluence even dictates that all visible transitions and divergences enabled from a state s can directly be mimicked from another state t if $s \twoheadrightarrow_{\mathcal{T}} t$. Hence, we can just keep following a confluent path and only retain the last state. To avoid getting stuck in an infinite confluent loop, we detect entering a bottom strongly connected component (BSCC) of confluent transitions and choose a unique *representative* from this BSCC for all states that can reach it. Since we showed that confluent joinability is transitive (as implied by Proposition 11), it follows immediately that all confluent paths emanating from a certain state s always end up in a unique BSCC.

Formally, we use the notion of a *representation map*, assigning a representative state $\varphi(s)$ to every state s. We make sure that $\varphi(s)$ indeed exhibits all behaviour of s due to being in a BSCC reachable from s.

Definition 15 (Representation map). *Let $\mathcal{M} = \langle S, s^0, A, \hookrightarrow, \rightsquigarrow \rangle$ be an MA and $\mathcal{T}$ a Markovian confluent set for $\mathcal{M}$. Then, a function $\varphi_{\mathcal{T}} \colon S \to S$ is a* representation map *for $\mathcal{M}$ under $\mathcal{T}$ if for all $s, s' \in S$*

- $s \twoheadrightarrow_{\mathcal{T}} \varphi_{\mathcal{T}}(s)$
- $s \to_{\mathcal{T}} s' \implies \varphi_{\mathcal{T}}(s) = \varphi_{\mathcal{T}}(s')$

Note that the first requirement ensures that every representative is reachable by all states it represents, while the second takes care that all $\mathcal{T}$-related states have the same representative. Together, they imply that every representative is in a BSCC. Since all $\mathcal{T}$-related states have the same BSCC, as discussed above, it is indeed always possible to find a representation map. We refer to [6] for the algorithm we use to construct it in our implementation.

As representatives exhibit all behaviour of the states they represent, they can be used for state space reduction. More precisely, it is possible to define the quotient of an MA modulo a representation map. This system does not have any $\mathcal{T}$-transitions anymore, except for self-loops on representatives that have outgoing $\mathcal{T}$-transitions in the original system. These ensure preservation of divergences.

Definition 16 (Quotient). *Given an MA $\mathcal{M} = \langle S, s^0, A, \hookrightarrow, \rightsquigarrow \rangle$, a confluent set $\mathcal{T}$ for $\mathcal{M}$, and a representation map $\varphi \colon S \to S$ for $\mathcal{M}$ under $\mathcal{T}$, the* quotient of $\mathcal{M}$ modulo φ *is the smallest system $\mathcal{M}/\varphi = \langle \varphi(S), \varphi(s^0), A, \hookrightarrow_{\varphi}, \rightsquigarrow_{\varphi} \rangle$ such that*

- $\varphi(S) = \{\varphi(s) \mid s \in S\}$;
- $\varphi(s) \xhookrightarrow{a}_{\varphi} \varphi(\mu)$ *if* $\varphi(s) \xhookrightarrow{a} \mu$;
- $\varphi(s) \overset{\lambda}{\rightsquigarrow}_{\varphi} \varphi(s')$ *if* $\lambda = \sum_{\lambda' \in \Lambda(s,s')} \lambda'$ *and* $\lambda > 0$,

where $\Lambda(s, s')$ is the multiset $\{\!| \lambda' \in \mathbb{R} \mid \exists s^* \in S \,.\, \varphi(s) \overset{\lambda'}{\rightsquigarrow} s^* \wedge \varphi(s^*) = \varphi(s') |\!\}$.

Note that each interactive transition from $\varphi(s)$ in $\mathcal{M}$ is lifted to $\mathcal{M}/\varphi$ by changing all states in the support of its target distribution to their representatives.

Additionally, each pair $\varphi(s), \varphi(s')$ of representative states in $\mathcal{M}/\varphi$ has a connecting Markovian transition with rate equal to the total outgoing rate of $\varphi(s)$ in $\mathcal{M}$ to states s^* that have $\varphi(s')$ as their representative (unless this sum is 0). It is easy to see that this implies $\varphi(s) \xrightarrow{\chi(\lambda)}_{\varphi} \varphi(\mu)$ if and only if $\varphi(s) \xrightarrow{\chi(\lambda)} \mu$.

Since $\mathcal{T}$-transitions connect bisimilar states, and representatives exhibit all behaviour of the states they represent, we can prove the following theorem. It shows that we indeed reached our goal of providing a reduction that is safe with respect to divergence-sensitive branching bisimulation.

Theorem 17. *Let $\mathcal{M} = \langle S, s^0, A, \hookrightarrow, \rightsquigarrow \rangle$ be an MA, $\mathcal{T}$ a Markovian confluent set for $\mathcal{M}$, and $\varphi \colon S \to S$ a representation map for $\mathcal{M}$ under $\mathcal{T}$. Then,*

$$\mathcal{M}/\varphi \;\underline{\leftrightarrow}_{\mathrm{b}}^{\mathrm{div}}\; \mathcal{M}.$$

5 Symbolic Detection of Markovian Confluence

Although the definition of confluence in Section 3 is useful to show the correctness of our approach, it is often not feasible to check in practice. After all, we want to reduce *on-the-fly* to obtain a smaller state space without first generating the unreduced one. Therefore, we use heuristics to detect Markovian confluence in the context of the process-algebraic modelling language MAPA [23]. As these heuristics only differ slightly from the ones in [24] for probabilistic confluence, we discuss the basics and explain how the old techniques can be reused.

MAPA is data-rich and expressive, and features a restricted form: the Markovian Linear Probabilistic Process Equation (MLPPE). Every MAPA specification can be translated easily to an equivalent specification in MLPPE [23]. Hence, it suffices to define our confluence-based reduction technique on this form.

The MLPPE Format. An MLPPE is a process with global variables, *interactive summands* (each yielding a set of interactive transitions) and *Markovian summands* (each yielding a set of Markovian transitions). Its semantics is given as an MA, whose states are valuations of the global variables. Basically, in each state a nondeterministic choice is made between the summands that are enabled given these values.

Each interactive summand has a condition (the guard) that specifies for which valuations of the global variables it is enabled. If so, an action can be taken and the next state (a new valuation for the global variables) is determined probabilistically. The action and next state may also depend on the current state. The Markovian summands are similar, except that they contain a rate and a unique next state instead of an action and a probabilistic next state. We assume an implicit confluence classification $P = \{C_1, \ldots, C_k\}$ that, for each interactive summand, contains a group consisting of all transitions generated by that summand. We note that this classification is only given for theoretical reasons; it is not actually constructed.

For a precise formalisation of the language and its semantics, we refer to [23].

Confluent Summands. We check for *confluent summands*: summands that are guaranteed to *only* yield confluent transitions, i.e., summands i such that the set $\mathcal{T} = \{C_i\}$ is confluent. Whenever during state space generation such a summand is enabled, all other summands can be ignored (continuing until reaching a representative in a BSCC, as explained in the previous section). By Theorem 12, the union of all confluent summands is also confluent.

Since only τ-transitions can be confluent, the only summands that might be confluent are interactive summands having action τ for all valuations of the global variables. Also, the next state of each of the transitions they generate should be unique. Finally, we verify whether all transitions that may result from these summands commute with all other transitions according to Definition 9.

We only need to check commutativity with all transitions possibly generated by the interactive summands, as the Markovian summands are never enabled at the same time as an invisible transition due to the maximal progress assumption. We overapproximate commutativity by checking whether, when two summands are enabled, they do not disable each other and do not influence each other's actions, probabilities and next states. After all, that implies that each transition can be mimicked by a transition from the same summand (and hence also that it is indeed mimicked by the same group of P). This can be formally expressed as a logical formula (see [24] for the details). Such a formula can be checked by an SMT solver, or approximated using heuristics. We implemented basic heuristics, checking mainly whether two summands are never enabled at the same time or whether the variables updated by one are not used by the other and vice versa. Additionally, some laws from the natural numbers have been implemented, taking for instance into account that $x := x + 1$ cannot disable $x > 2$. In the future, we hope to extend this to more advanced theorem proving.

6 Case Studies

We implemented confluence reduction in our tool SCOOP [22]. It takes MAPA specifications as input, is able to perform several reduction techniques and can generate state spaces in multiple formats, among which the one for the IMCA tool for model checking MAs [17]. We already showed in [23] the benefits of dead variable reduction. Here, we apply only confluence reduction, to focus on the power of our novel technique. We present the size of the state spaces with and without confluence reduction, as well as the time to generate them with SCOOP and to subsequently analyse them with IMCA. That way, the impact of confluence reduction on both MA generation and analysis becomes clear[3].

We conjecture that the (quantitative) behavioural equivalence induced by branching bisimulation leaves invariant the time-bounded reachability probabilities, expected times to reachability and long-run averages computed by IMCA. This indeed turned out to be the case for all our models. A logic precisely characterising Markovian branching bisimulation would be interesting future work.

Leader Election Protocol. The first case study is a leader election protocol (Algorithm $\mathcal{B}$ from [15]), used in [24] as well to demonstrate confluence reduction for probabilistic automata. It uses asynchronous channels and allows for multiple nodes, throwing dice to break the symmetry. We added a rate 1 to a node

[3] The tool (for download and web-based usage [5]), all MAPA models and a test script can be found on `http://fmt.cs.utwente.nl/~timmer/scoop/papers/formats`.

Table 1. State space generation and analysis using confluence reduction (on a 2.4 GHz 4 GB Intel Core 2 Duo MacBook). Runtimes in SCOOP and IMCA are in seconds.

	Original state space				Reduced state space				Impact	
Specification	States	Trans.	SCOOP	IMCA	States	Trans.	SCOOP	IMCA	States	Time
`leader-3-7`	25,505	34,257	4.7	102.5	5,564	6,819	5.1	9.3	-78%	-87%
`leader-3-9`	52,465	71,034	9.7	212.0	11,058	13,661	10.4	17.8	-79%	-87%
`leader-3-11`	93,801	127,683	18.0	429.3	19,344	24,043	19.2	31.9	-79%	-89%
`leader-4-2`	8,467	11,600	2.1	74.0	2,204	2,859	2.5	6.8	-74%	-88%
`leader-4-3`	35,468	50,612	9.0	363.8	7,876	10,352	8.7	33.3	-78%	-89%
`leader-4-4`	101,261	148,024	25.8	1,309.8	20,857	28,023	24.3	94.4	-79%	-91%
`polling-2-2-4`	4,811	8,578	0.7	3.7	3,047	6,814	0.7	2.3	-37%	-32%
`polling-2-2-6`	27,651	51,098	12.7	91.0	16,557	40,004	5.4	49.0	-40%	-48%
`polling-2-4-2`	6,667	11,290	0.9	39.9	4,745	9,368	0.9	26.6	-29%	-33%
`polling-2-5-2`	27,659	47,130	4.0	1,571.7	19,721	39,192	4.0	1,054.6	-29%	-33%
`polling-3-2-2`	2,600	4,909	0.4	7.1	1,914	4,223	0.5	4.8	-26%	-29%
`polling-4-6-1`	15,439	29,506	3.1	330.4	4,802	18,869	3.0	109.4	-69%	-66%
`polling-5-4-1`	21,880	43,760	5.1	815.9	6,250	28,130	5.1	318.3	-71%	-61%
`processor-2`	2,508	4,608	0.7	2.8	1,514	3,043	0.8	1.2	-44%	-43%
`processor-3`	10,852	20,872	3.1	66.3	6,509	13,738	3.3	23.0	-45%	-62%
`processor-4`	31,832	62,356	10.8	924.5	19,025	41,018	10.3	365.6	-45%	-60%

throwing a die to get an MA model based on the original case study, making the example more relevant and interesting in the current situation. We computed the minimal probability (with error bound 0.01) of electing the first node as leader within 5 time units. The results are presented in Table 1, where we denote by `leader-i-j` the variant with `i` nodes and `j`-sided dice. The computed probability varies from 0.09 for `leader-4-2` to 0.32 for `leader-3-11`. Confluence saved almost 90% of the total time to generate and analyse the models. The substantial reductions are due to extensive interleaving with little communication.

Queueing System. The second case study is the queueing system from [23]. It consists of multiple stations with incoming jobs, and one server that polls the stations for work. With some probability, communication fails. There can be different sizes of buffers in the stations, and multiple types of jobs with different service rates. In Table 1, we let `polling-i-j-k` denote the variant with `i` stations, all having buffers of size `j` and `k` types of jobs. Note that, although significant reductions are obtained, the reduction in states precisely corresponds to the reduction in transitions; this implies that only trivially confluent transitions could be reduced (i.e., invisible transitions without any other transitions from the same source state). We computed the minimal and maximal expected time to the situation that all buffers are full. This turns out to be at least 1.1—for `polling-3-2-2`—and at most 124—for `polling-2-5-2`. Reductions were less substantial, due to the presence of many probabilistic and Markovian transitions.

Processor Architecture. The third case study is a GSPN model of a 2×2 concurrent processor architecture, parameterised in the level k of multitasking, taken from Figure 11.7 in [1]. We constructed a corresponding MAPA model, modelling each place as a global variable and each transition as a summand. As in [1], we computed the throughput of one of the processors, given by the long-run average of having a token in a certain place of the GSPN. Whereas [1] resolved all nondeterminism and found for instance a throughput of 0.903 for $k = 2$, we can retain the nondeterminism and obtain the more informative interval $[0.811, 0.995]$. (When resolving nondeterminism as before, we reproduce the result 0.903.)

Our results clearly show the significant effect of confluence reduction on the state space sizes and the duration of the heavy numerical computations by IMCA. The generation times by SCOOP are not reduced as much, due to the additional overhead of computing representative states. To keep memory usage in the order of the reduced state space, the representative map is deliberately not stored and therefore potentially recomputed for some states.

7 Conclusions

We introduced confluence reduction for MAs: the first reduction technique for this model that abstracts from invisible transitions. We showed that it preserves divergence-sensitive branching bisimulation, and hence yields quantitatively behavioural equivalent models. In addition to working on MAs, our novel notion of confluence reduction has two additional advantages over previous notions. First, it preserves divergences, and hence does not alter minimal reachability probabilities. Second, it is closed under unions, enabling us to separately detect confluence of different sets of transitions and combine the results. We also showed that the representation map approach can still be used safely to reduce systems on-the-fly, and discussed how to detect confluence syntactically on the process-algebraic language MAPA. Case studies with our tool SCOOP on several instances of three different models show state space reductions up to 79%. We linked SCOOP to the IMCA model checker to illustrate the significant impact of these reductions on the expected time, time-bounded reachability and long-run average computations. Due to confluence reduction, for some models the entire process from MAPA specification to results is now more than ten times as fast.

As future work we envision to search for even more powerful ways of using commutativity for state space reduction, for instance by allowing confluent transitions to be probabilistic. Preferably, this would enable even more aggressive reductions that, instead of preserving the conservative notion of bisimulation we used, preserve the more powerful weak bisimulation from [14].

Acknowledgements. We thank Stefan Blom and Joost-Pieter Katoen for their useful suggestions, and Dennis Guck for his help with the case studies.

References

[1] Ajmone Marsan, M., Balbo, G., Conte, G., Donatelli, S., Franceschinis, G.: Modelling with Generalized Stochastic Petri Nets. John Wiley & Sons, Inc. (1994)
[2] Ajmone Marsan, M., Conte, G., Balbo, G.: A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems. ACM Transactions on Computer Systems 2(2), 93–122 (1984)
[3] Baier, C., D'Argenio, P.R., Größer, M.: Partial order reduction for probabilistic branching time. In: QAPL. ENTCS, vol. 153(2), pp. 97–116 (2006)
[4] Baier, C., Größer, M., Ciesinski, F.: Partial order reduction for probabilistic systems. In: QEST, pp. 230–239 (2004)
[5] Belinfante, A., Rensink, A.: Publishing your prototype tool on the web: PUPTOL, a framework. Technical Report TR-CTIT-13-15, Centre for Telematics and Information Technology, University of Twente (2013)
[6] Blom, S.C.C.: Partial τ-confluence for efficient state space generation. Technical Report SEN-R0123, CWI, Amsterdam (2001)

[7] Blom, S.C.C., van de Pol, J.C.: State space reduction by proving confluence. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, pp. 596–609. Springer, Heidelberg (2002)

[8] Boudali, H., Crouzen, P., Haverkort, B.R., Kuntz, M., Stoelinga, M.I.A.: Architectural dependability evaluation with arcade. In: DSN, pp. 512–521 (2008)

[9] Boudali, H., Crouzen, P., Stoelinga, M.I.A.: A rigorous, compositional, and extensible framework for dynamic fault tree analysis. IEEE Transactions on Dependable and Secure Compututing 7(2), 128–143 (2010)

[10] Bozzano, M., Cimatti, A., Katoen, J.-P., Nguyen, V.Y., Noll, T., Roveri, M.: Safety, dependability and performance analysis of extended AADL models. The Computer Journal 54(5), 754–775 (2011)

[11] D'Argenio, P.R., Niebert, P.: Partial order reduction on concurrent probabilistic programs. In: QEST, pp. 240–249 (2004)

[12] Deng, Y., Hennessy, M.: On the semantics of Markov automata. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011, Part II. LNCS, vol. 6756, pp. 307–318. Springer, Heidelberg (2011)

[13] Eisentraut, C., Hermanns, H., Zhang, L.: Concurrency and composition in a stochastic world. In: Gastin, P., Laroussinie, F. (eds.) CONCUR 2010. LNCS, vol. 6269, pp. 21–39. Springer, Heidelberg (2010)

[14] Eisentraut, C., Hermanns, H., Zhang, L.: On probabilistic automata in continuous time. In: LICS, pp. 342–351 (2010)

[15] Fokkink, W., Pang, J.: Simplifying Itai-Rodeh leader election for anonymous rings. In: AVoCS. ENTCS, vol. 128(6), pp. 53–68 (2005)

[16] Godefroid, P.: Partial-Order Methods for the Verification of Concurrent Systems. LNCS, vol. 1032. Springer, Heidelberg (1996)

[17] Guck, D., Hatefi, H., Hermanns, H., Katoen, J.-P., Timmer, M.: Modelling, reduction and analysis of Markov automata. In: Joshi, K., Siegle, M., Stoelinga, M., D'Argenio, P.R. (eds.) QEST 2013. LNCS, vol. 8054, pp. 55–71. Springer, Heidelberg (2013)

[18] Hansen, H., Timmer, M.: A comparison of confluence and ample sets in probabilistic and non-probabilistic branching time. In: TCS (to appear, 2013)

[19] Hartmanns, A., Timmer, M.: On-the-fly confluence detection for statistical model checking. In: Brat, G., Rungta, N., Venet, A. (eds.) NFM 2013. LNCS, vol. 7871, pp. 337–351. Springer, Heidelberg (2013)

[20] Katoen, J.-P.: GSPNs revisited: Simple semantics and new analysis algorithms. In: ACSD, pp. 6–11 (2012)

[21] Peled, D.: All from one, one for all: on model checking using representatives. In: Courcoubetis, C. (ed.) CAV 1993. LNCS, vol. 697, pp. 409–423. Springer, Heidelberg (1993)

[22] Timmer, M.: SCOOP: A tool for symbolic optimisations of probabilistic processes. In: QEST, pp. 149–150 (2011)

[23] Timmer, M., Katoen, J.-P., van de Pol, J.C., Stoelinga, M.I.A.: Efficient modelling and generation of Markov automata. In: Koutny, M., Ulidowski, I. (eds.) CONCUR 2012. LNCS, vol. 7454, pp. 364–379. Springer, Heidelberg (2012)

[24] Timmer, M., Stoelinga, M.I.A., van de Pol, J.C.: Confluence reduction for probabilistic systems. In: Abdulla, P.A., Leino, K.R.M. (eds.) TACAS 2011. LNCS, vol. 6605, pp. 311–325. Springer, Heidelberg (2011)

[25] Timmer, M., van de Pol, J.C., Stoelinga, M.I.A.: Confluence reduction for Markov automata (extended version). Technical Report TR-CTIT-13-14, Centre for Telematics and Information Technology, University of Twente (2013)

[26] Valmari, A.: Stubborn sets for reduced state space generation. In: Rozenberg, G. (ed.) APN 1989. LNCS, vol. 424, pp. 491–515. Springer, Heidelberg (1990)

Optimal Control for Linear-Rate Multi-mode Systems

Dominik Wojtczak

University of Liverpool, Liverpool, UK
d.wojtczak@liverpool.ac.uk

Abstract. Linear-Rate Multi-Mode Systems is a model that can be seen both as a subclass of switched linear systems with imposed global safety constraints and as hybrid automata with no guards on transitions. We study the existence and design of a controller for this model that keeps at all times the state of the system within a given safe set. A sufficient and necessary condition is given for such a controller to exist as well as an algorithm that finds one in polynomial time. We further generalise the model by adding costs on modes and present an algorithm that constructs a safe controller which minimises the peak cost, the average-cost or any cost expressed as a weighted sum of these two. Finally, we present numerical simulation results based on our implementation of these algorithms.

1 Introduction

Optimisation of electricity usage is an increasingly important issue because of the growing energy prices and environmental concerns. In order to make the whole system more efficient, not only the average electricity consumption should be minimised but also its peak demand. The energy produced during the peak times, typically occurring in the afternoon due to the heaters or air-conditioning units being switched on at the same time once people come back from work, is not only more expensive because the number of consumers outweighs the suppliers, but also the *peaking power plants*, which provide the supply at that time, are a lot less efficient. Therefore, the typical formula that is used for charging companies for electricity is a weighted average of its peak and average electricity demand [15,1]. Optimisation of the usage pattern of heating, ventilation and air-conditioning units (HVAC) not only can save electricity but also contribute to their longer lifespan, because they do not have to be used just as much.

In [10] Nghiem et al. considered a model of a building consisting of a number of decoupled zones whose temperatures have to remain within a specified comfort temperature interval. Each zone has a heater with a number of possible output settings, but only one of these setting can be in use. That is, the heater can either be on in that one setting or it has to be off otherwise. A further restriction is that only some fixed number of heaters can be on at any time. The temperature evolution in each zone is governed by a linear differential equation whose parameters depend on the physical characteristics of the zone, the outside temperature, the heater's picked setting and whether it is on or off. The aim is to find a safe controller, i.e. a sequence of time points at which to switch the heaters on or off, in order for the temperature in each zone to remain in its comfort interval, which is given as the input. In the end, it was shown that a sufficient condition for such a controller to exist requires just one simple inequality to hold. This technique can be used to minimise the maximum number of heaters used simultaneously at any point in time, but if the heaters have different energy consumption, then this may not minimise the peak energy cost.

We strictly generalise the model presented in [10] and define linear-rate Multi-Mode Systems (MMS). The evolution of our system is the same, i.e. it consists of a number of zones, which we call variables, whose evolution do not directly influence each other. However, we do not assume that all HVAC units in the zones are heaters, so the system can cope with a situation when cooling is required during the day and heating during the night. Moreover, rather than having all possible combinations of settings allowed, our systems have a list of allowable joint settings for all the zones instead; we will call such a joint setting a *mode*. This allows to model specific behaviours, for instance, heat pumps, i.e. when the heat moves from one zone into another, and a central heating system, which can only heat all the zones at the same time. Finally, we will be looking for the actual minimum peak cost without restricting ourselves to just one setting per heater nor the number of heaters being switched on at the same time, while keeping the running time polynomial in the number of modes. We also show how to find the minimum average-cost schedule and finally how to minimise the energy bill expressed as a weighted sum of the peak and the average energy consumption.

Related Work. MMSs can be seen both as switched linear systems (see, e.g. [6,13]) with imposed global safety constraints or as hybrid automata ([4,7]) with no guards on transitions. The analysis of switched linear systems typically focuses on several forms of stabilisation, e.g. whether the system can be steered into a given stable region, which the system will never leave again. However, all these analyses for switched systems are done in the limit and do not impose any constraints on the state of the system before it reaches the safe region. Such an analysis may suffice for systems where the constraints are soft, e.g. nothing serious will happen if the temperature in a room will briefly be too high or too low. However, it may not be enough when studying safety-critical systems, e.g. when cooling nuclear reactors. Each zone in an MMS is given a safe value interval in which the zone has to be at all times. This causes an interesting behaviour, because even if the system stabilises while staying forever in any single mode, these stable points may be all unsafe and therefore the controller has to constantly switch between different modes to keep the MMS within the safety set. For instance, a heater in a room has to constantly switch itself on and off as otherwise the temperature will become either too high or too low. On the other hand, even the basic questions are undecidable for hybrid automata (see, e.g. [8]) and therefore MMSs constitute its natural subclass with decidable and even tractable safety analysis. Although, some existing techniques, such as barrier certificates and differential invariants, used in safety verification of hybrid systems (see, e.g. [3] for an overview) can check whether no trajectory of our system enters the unsafe region, we are not aware of any existing technique applicable to our model which could check in polynomial-time whether a safe trajectory exists.

As mentioned earlier, we strictly generalise the model in [10], which was then further generalised by the same authors in several other papers with their most general model appearing in [11]. In that model disturbances and interactions between zones were added to the dynamics of the system. This makes it incomparable to our model, as we do not consider these aspects. Instead we allow each heater to have multiple settings in use and allow for restricting the set of valid joint settings by utilising modes.

In [2] we recently studied a different incomparable class of constant-rate Multi-Mode Systems where in each mode the state of a zone changes with a constant-rate as opposed to being governed by a linear differential equation as in linear-rate MMS. Specifically, for every mode $m \in M$ and variable $\mathbf{x}_i$ the value of $\mathbf{x}_i$ after spending time t in mode m increases by $c_i^m \cdot t$ where $c_i^m \in \mathbb{R}$ is the constant rate of change of $\mathbf{x}_i$ in mode m. Unlike linear-rate MMSs, that model is a subclass of *linear hybrid automata* ([4]), which has constant-rate dynamics and linear functions as guards on transitions. We showed a polynomial-time algorithm for the safe controllability and safe reachability questions, as well as for finding optimal safe controllers in the generalised model where each mode has an associated cost per time unit.

There are many other approaches to reduce energy consumption and peak usage in buildings. One particularly popular one is model predictive control (MPC) [5]. In [12] stochastic MPC was used to minimize building's energy consumption, while in [9] the peak electricity demand reduction was considered. The drawback of using MPC in our setting is its high computational complexity and the fact it cannot guarantee optimality.

Results. The key contribution of the paper is a polynomial-time algorithm, which we present in Section 3, that can check for any MMS and starting point in the interior of the safety set whether a safe controller exists and is able to construct it. Unlike in [10], we not only show a sufficient, but also necessary, condition for such a safe controller to exist. The condition is a system of linear inequalities that can be solved using polynomial-time algorithms for linear programming (see, e.g. [14]) and because that system does not depend on the starting state, this shows that either all points in the interior of the safe set have a safe controller or none of them has one. Furthermore, we show that if a safe controller exists then also a safe periodic controller exists with polynomial-size *minimum dwell time*, i.e. the smallest amount of time between two mode switches. Such a minimum dwell time may be still too small for practical purposes, because it may require too frequent switches between modes. However, we prove that the problem of checking whether there is a safe controller with the minimum dwell time higher than 1 (or any other fixed constant) is PSPACE-hard. This means that any approximation of the largest minimum dwell time among all safe controllers is unlikely to be tractable. It should be noted that the definition of MMS allows for an arbitrary switching between modes. However, it is possible to enrich the dynamics of the MMS model by adding a restriction on the order in which the modes can be used. As formally stated in Corollary 1, such an extended model can still be analysed in polynomial-time.

In Section 4 we generalise the MMS model by associating cost per time unit with each mode and search for a safe schedule that minimises the peak cost or the long-time average cost. Similarly as before, if there is at least one safe controller, then the optimal cost do not depend on the starting point and there is always a periodic optimal controller. To compute the minimum peak cost we use a binary search algorithm. On the other hand, finding a controller with the minimum average-cost is more involved. In order to prove that the controller that we construct has the minimum average-cost it is crucial that the condition for the existence of a safe controller, which we describe in Section 3, is both sufficient and necessary. Finally, we show how to find a controller with the minimum cost calculated as a weighted sum of the peak cost and the average cost. The challenging part is that peak cost generally increases when the set of modes is expanded while the average-cost decreases. Therefore, the weighted cost may not be monotone in the size of the set of modes and so a binary search may not work and finding the minimum cost may require checking many possible subsets of the set of modes. Nevertheless, we show that all these safe optimal controllers can be constructed in time polynomial in the number of modes.

On the other hand, if one considers the set of modes to be given implicitly as in [10], where each zone has a certain number of settings and all their possible combinations are allowed, then the number of modes becomes exponential in the size of the input. In Algorithm 2, whose performance is tested in Section 5, we try to cope with this problem by performing a bottom-up binary search in order to avoid analysing large sets of modes and use other techniques to keep the running time manageable in practice.

Due to the space constraints most details of the proofs are omitted and can be found in the full version of this paper [16].

2 Linear Multi-mode Systems

Let us set the notation first. We write $\mathbb{N}$ to denote the set of nonnegative integer numbers. Also, we write $\mathbb{R}$, $\mathbb{R}_{\geq 0}$, and $\mathbb{R}_{>0}$ for the sets of all, non-negative and strictly positive real numbers, respectively. States of our system will be points in the Euclidean space $\mathbb{R}^n$ equipped with the standard *Euclidean norm* $\|\cdot\|$. By $\overline{x}$,$\overline{y}$ we denote points in this state space, by $\vec{f}$,$\vec{v}$ vectors, while $\overline{x}(i)$ and $\vec{f}(i)$ will denote the i-th coordinate of point $\overline{x}$ and vector $\vec{f}$, respectively. For $\diamond \in \{\leq,<,\geq,>\}$, we write $\overline{x} \diamond \overline{y}$ if $\overline{x}(i) \diamond \overline{y}(i)$ for all i. For a n-dimensional vector $\vec{v}$ by $\mathrm{diag}(\vec{v})$ we denote a $n \times n$ dimensional matrix whose diagonal is $\vec{v}$ and the rest of the entries are 0. We can now formally define our model.

Definition 1. *A linear-rate multi-mode system (MMS) is a tuple $\mathcal{H} = (M,N,A,B)$ where M is a finite nonempty set of* modes*, N is the number of continuous-time variables in the system, and $A : M \to \mathbb{R}^N_{>0}, B : M \to \mathbb{R}^N$ give for each mode the coefficients of the linear differential equation that governs the dynamics of the system.*

Note that the number of modes of $\mathcal{H}$ is $|M|$. In all further computational complexity considerations, we assume that all real numbers are rational and represented in the standard way by writing down the numerator and denominator in binary. Throughout the paper we will write a_i^m and b_i^m as a shorthand for $A(m)(i)$ and $B(m)(i)$, respectively.

A *controller* of an MMS specifies a timed sequence of mode switches. Formally, a *controller* is defined as a finite or infinite sequences of *timed actions*, where a timed action $(m,t) \in M \times \mathbb{R}_{>0}$ is a tuple consisting of a mode and a time delay. We say that an infinite controller $\langle (m_1,t_1),(m_2,t_2),\ldots \rangle$ is *Zeno* if $\sum_{k=1}^{\infty} t_k < \infty$ and is *periodic* if there exists $l \geq 1$ such that for all $k \geq 1$ we have $(m_k,t_k) = (m_{(k \bmod l)+1}, t_{(k \bmod l)+1})$. Zeno controllers require infinitely many mode-switches within a finite amount of time, and hence, are physically unrealizable. However, one can argue that a controller that switches after $t_k = 1/k$ amount of time at the k-th timed action is also infeasible, because it requires the switches to occur infinitely frequently in the limit. Therefore, we will call a non-Zeno controller *feasible* if its minimum dwell time, i.e. the smallest amount of time between two mode switches, can be bounded from below by a positive constant. We will relax this assumption and allow for the modes that are not used at all by a feasible controller to occur in its sequence of time actions with time delays equal to 0. For a controller $\sigma = \langle (m_1,t_1),(m_2,t_2),\ldots\rangle$, we write $T_k(\sigma) \overset{\text{def}}{=} \sum_{i=1}^{k} t_i$ for the total time elapsed up to step k of the controller σ, $T_k^m(\sigma) \overset{\text{def}}{=} \sum_{i\le k: m_i=m} t_i$ for the total time spent in mode m up to step k, and finally $t_{\min}(\sigma) = \inf_{\{k:\, t_k>0\}} t_k$ defines the minimum dwell time of σ. For any non-Zeno controller σ we have that $\lim_{k\to\infty} T_k(\sigma) = \infty$ and for any feasible controller σ we also have $t_{\min}(\sigma) > 0$. Finally, for any $t \ge 0$ let $\sigma(t)$ denote the mode the controller σ directs the system to be in at the time instance t. Formally, we have $\sigma(t) = m_k$ where $k = \min\{i : t \le T_i(\sigma)\}$.

The state of MMS $\mathscr{H}$ initialized at a starting point $\overline{x}_0$ under control σ is a N-tuple of continuous-time *variables* $\overline{\mathbf{x}}(t) = (\mathbf{x}_1(t),\ldots,\mathbf{x}_N(t))$ such that $\overline{\mathbf{x}}(0) = \overline{x}_0$ and $\dot{\overline{\mathbf{x}}}(t) = B(\sigma(t)) - \operatorname{diag}(A(\sigma(t)))\overline{\mathbf{x}}(t)$ holds at any time $t \in \mathbb{R}_{\ge 0}$. It can be seen that if $\mathscr{H}$ is in mode m during the entire time interval $[t_0,t_0+t]$ then the following holds $\mathbf{x}_i(t_0+t) = b_i^m/a_i^m + (\mathbf{x}_i(t_0) - b_i^m/a_i^m)e^{-a_i^m t}$. Notice that this expression is monotonic in t and converges to b_i^m/a_i^m, because based on the definition of MMS we have $a_i^m > 0$ for all m and i.

Given a set $S \subseteq \mathbb{R}^N$ of safe states, we say that a controller σ is S-safe for MMS $\mathscr{H}$ initialised at $\overline{x}_0$ if for all $t \ge 0$ we have $\mathbf{x}(t) \in S$. We sometimes say safe instead of S-safe if S is clear from the context. In this paper we restrict ourselves to safe sets being hyperrectangles, which can be specified by giving lower and upper bound value for each variable in the system. This assumption implies that controller σ is S-safe iff $\mathbf{x}(t) \in S$ for all $t \in \{T_k(\sigma) : k \ge 0\}$, because each $\mathbf{x}_i(t)$ is monotonic when $\mathscr{H}$ remains in the same mode and so if system is S-safe at two time points, the system is S-safe in between these two time points as well. This fact is crucial to the further analysis and allows us to only focus on S-safety at the mode switching time points of the controller. Formally, to specify any hyperrectangle S, it suffices to give two points $\overline{l},\overline{u} \in \mathbb{R}^N$, which define this region as follows $S = \{\overline{x} : \overline{l} \le \overline{x} \le \overline{u}\}$. The fundamental decision problem for MMS that we solve in this paper is the following.

Definition 2 (Safe Controllability). *Decide whether there exists a feasible S-safe controller for a given MMS $\mathscr{H}$, a hyperrectangular safe set S given by two points $\overline{l}$ and $\overline{u}$ and an initial point $\overline{x}_0 \in S$.*

The fact that $a_i^m > 0$ for all m and i make the system *stable* in any mode, i.e. if the system stays in any fixed mode forever, it will converge to an equilibrium point. However, none of these equilibrium points may be S-safe and as a result the controller may need to switch between modes in order to be S-safe. We present an algorithm to solve the safe controllability problem in Section 3 and later, in Section 4, we generalise the model to MMS with costs associated with modes and the aim being finding a feasible S-safe controller with the minimum average-cost, peak cost, or some weighted sum of these. As the following example shows, safe controllability can depend on the starting point if it lies on the boundary of the safe set. We will not consider this special case further and assume instead that the starting point belongs to the interior of the safe set.

Example 1. Consider an apartment with two rooms and one heater. The heater can only heat one room at a time. When it is off, the room temperature converges to the outside temperature of 12°C, while if it is constantly on, the temperature of the room converges to 30°C. We assume the comfort temperature to be between 18°C and 22°C. The table to the right shows the coefficients b_i^m for all modes m and rooms i, while all the a_i^m-s are assumed to be equal to 1. Intuitively, when heating room 1 (mode m_2) and room 2 (mode m_3) half of the time each, the temperature in each room should oscillate around (30°C +12°C)/2 = 21°C and never leave the comfort zone assuming the switching occurs frequently enough (every 0.005 as shown later). We will prove this intuition formally in Section 3. Therefore, as long as the temperature in one of the rooms is above 18°C at the very beginning, a safe controller exists. However, if the temperatures in both rooms start at 18°C (a state which is safe), a safe controller does not exists, because the temperature drops in at least one of the rooms in each mode and so the state becomes unsafe under any control.

Modes	m_1	m_2	m_3
b_1^m (Room 1)	12	30	12
b_2^m (Room 2)	12	12	30

3 Safe Schedulability

Let us fix in this section a linear-rate MMS $\mathcal{H} = (M, N, A, B)$ and a safe set S given by two points $\overline{l}, \overline{u} \in \mathbb{R}^N$, such that $\overline{l} < \overline{u}$ and $S = \{\overline{x} : \overline{l} \leq \overline{x} \leq \overline{u}\}$. We call any vector $\vec{f} \in \mathbb{R}^M_{\geq 0}$ such that $\sum_{m \in M} \vec{f}(m) = 1$ a *frequency vector*. Let us define $F_i(\vec{f}, y) := \sum_{m \in M} \vec{f}(m)(b_i^m - a_i^m y)$. Intuitively $F_i(\vec{f}, y)$ describes the trend of the value of the i-th variable when its value is y and the frequency of each mode is given by $\vec{f}$, because it has the same sign as this variable's derivative. Notice that for a fixed i and frequency vector $\vec{f}$, function $F_i(\vec{f}, y)$ is continuous and strictly decreasing in y. Moreover, $F_i(\alpha\vec{f} + \beta\vec{g}, y) = \alpha F_i(\vec{f}, y) + \beta F_i(\vec{g}, y)$. For a frequency vector $\vec{f}$, variable $\mathbf{x}_i$ is called *critical* if $F_i(\vec{f}, \overline{l}_i) = 0$ or $F_i(\vec{f}, \overline{u}_i) = 0$ holds.

Definition 3. *A frequency vector $\vec{f}$ is* good *if for every variable $\mathbf{x}_i$ the following conditions hold (I) $F_i(\vec{f}, \overline{l}) \geq 0$, and (II) $F_i(\vec{f}, \overline{u}) \leq 0$. A frequency vector $\vec{f}$ is* implementable *if it is good and for every variable $\mathbf{x}_i$ we additionally have (III) if $F_i(\vec{f}, \overline{l}) = 0$ then $\vec{f}(m) = 0$ for every $m \in M$ such that $b_i^m / a_i^m \neq \overline{l}_i$, and (IV) if $F_i(\vec{f}, \overline{u}) = 0$ then $\vec{f}(m) = 0$ for every $m \in M$ such that $b_i^m / a_i^m \neq \overline{u}_i$.*

Theorem 1. *If there exists a feasible S-safe controller then there exists an implementable frequency vector.*

Proof (Sketch). Denote the feasible S-safe controller by σ. Let $f_k^{(m)} = T_k^m(\sigma)/T_k(\sigma)$ be the fraction of the time spent by σ in mode m up to its k-th timed action; note that $f_k^{(m)} \in [0,1]$, and $\sum_{m \in M} f_k^{(m)} = 1$ for all k. Let us look at the sequence of vectors $\langle \vec{f}_k \in [0,1]^M \rangle_{k=1}^{\infty}$ where we set $\vec{f}_k(m) = f_k^{(m)}$. Since this sequence is bounded, by the Bolzano-Weierstrass theorem, there exists an increasing integer sequence $j_1, j_2, \ldots$ such that $\lim_{k \to \infty} \vec{f}_{j_k}$ exists and let us denote this limit by $\vec{f}$. The rest of the proof shows by contradiction that $\vec{f}$ is an implementable frequency vector. □

Theorem 2. *If there exists an implementable frequency vector then there exists a periodic S-safe controller for any initial state in the interior of the safety set.*

Proof (Sketch). Let $\vec{f}$ be an implementable frequency vector. We claim that the following periodic controller $\sigma = \langle (m_k, t_k) \rangle_{k=1}^{\infty}$ with period of length $|M|$ is S-safe for sufficiently small total time duration of each period, denoted by τ: $m_k = (k \bmod |M|) + 1$ and $t_k = \vec{f}(m_k) \cdot \tau$. As we already know it suffices to check S-safety of the system at time points T_k for all k. By induction we can show that

$$\mathbf{x}_i(T_k) = \frac{b_i^{m_k}}{a_i^{m_k}} + \sum_{n=1}^{k-1} \left(\frac{b_i^{m_n}}{a_i^{m_n}} - \frac{b_i^{m_{n+1}}}{a_i^{m_{n+1}}} \right) e^{-\sum_{j=k-n+1}^{k} a_i^{m_j} t_j} + \left(\overline{x}_0(i) - \frac{b_i^{m_1}}{a_i^{m_1}} \right) e^{-\sum_{j=1}^{k} a_i^{m_j} t_j}$$

Now, let $\overline{x}_l \stackrel{\text{def}}{=} \overline{\mathbf{x}}(T_{l|M|})$ for all $l \in \mathbb{N}$. Notice that since σ is periodic with period $|M|$ from this equation we get that $\overline{x}_{l+1}(i) = \alpha_i(\tau)\overline{x}_l(i) + \beta_i(\tau)$ for all l, $\alpha_i(\tau) \stackrel{\text{def}}{=} e^{-\tau \cdot \sum_{j=1}^{|M|} a_i^{m_j} \vec{f}(m_j)}$ and some $\beta_i : \mathbb{R}_{>0} \to \mathbb{R}$. Now, because $0 < \alpha_i(\tau) < 1$, the sequence $\overline{x}_l(i)$ converges monotonically to $\beta_i(\tau)/(1-\alpha_i(\tau))$ as $l \to \infty$ for any initial value $\overline{x}_0(i)$. Looking at the exact form of the functions α_i and β_i, we then find a τ such that this limit is in the safe set as well as all the intermediate points along the way. □

Coming back to Example 1, we can check that the suggested there frequency vector $\vec{f} = (0, \frac{1}{2}, \frac{1}{2})$ is implementable, because $F_1(\vec{f}, 18°\text{C}) = F_2(\vec{f}, 18°\text{C}) = 0 \cdot (12°\text{C} - 18°\text{C}) + \frac{1}{2} \cdot (12°\text{C} - 18°\text{C}) + \frac{1}{2} \cdot (30°\text{C} - 18°\text{C}) = 3°\text{C} > 0$ and $F_1(\vec{f}, 22°\text{C}) = F_2(\vec{f}, 22°\text{C}) = \text{-}1°\text{C} < 0$. Also, assuming the initial temperature of each room is 20°C, the total time duration of each period τ can be computed to be 0.01, which gives rise to the following S-safe periodic feasible controller $\langle (m_1, 0), (m_2, 0.005), (m_3, 0.005), (m_1, 0), (m_2, 0.005), (m_3, 0.005), \ldots \rangle$.

Theorem 3. *Algorithm 1 returns in polynomial time a S-safe feasible controller from* $\overline{x}_0$ *if there exists one.*

Proof. We first need the following lemma.

Lemma 1. *Either there is a variable which is critical for all good frequency vectors or there is a good frequency vector in which no variable is critical.*

Now, let σ be the controller returned by Algorithm 1. Notice that the frequency vector $\vec{f}_*$, the controller σ is based on, is implementable. This is because $\vec{f}_*$ satisfies the constraints at line 12 which imply the conditions (I) and (II) of $\vec{f}$ being implementable and from Lemma 1 it follows that all modes that could violate the conditions (III) and (IV) were removed in the loop between lines 5–11. Moreover, constant τ used in the construction of σ is exactly the same as the one used in Theorem 2, which guarantees σ to be S-safe.

On the other hand, from Theorem 1, if there exists a feasible S-safe controller then there also exists an implementable frequency vector $\vec{f}$. Such a vector will satisfy the constraints of being good at line 2 of the algorithm. In the loop between the lines 5–11, all variables that are critical in $\vec{f}$ are first checked whether they satisfy conditions (III) and (IV), and they will satisfy them because $\vec{f}$ is implementable, and after that these critical variables are removed. Finally, $\vec{f}$ consisting of just the remaining variables will satisfy the constraints at line 7 of being implementable with no critical variables. Therefore, Algorithm 1 will always return a controller if there exists a S-safe one.

Algorithm 1. Finds a S-safe feasible controller from a given $\overline{x}_0 \in S$.

```
Input: MMS $\mathscr{H} = (M,N,A,B)$, two points $\overline{l}$ and $\overline{u}$ that define a hyperrectangle
       $S = \{\overline{x} : \overline{l} \leq \overline{x} \leq \overline{u}\}$ and an initial point $\overline{x}_0 \in S$ such that $\overline{l} < \overline{x}_0 < \overline{u}$.
Output: NO if no $S$-safe feasible controller exists from $\overline{x}_0$, and a periodic such controller,
        with the corresponding implementable frequency vector $\vec{f}_*$, otherwise.
1  $I := \{1,\ldots,N\}$;
2  Check whether the following linear program is satisfiable for some frequency vector $\vec{f}$:
       $F_i(\vec{f},\overline{l}_i) \geq 0$ for all $i \in I$
       $F_i(\vec{f},\overline{u}_i) \leq 0$ for all $i \in I$.
   if no satisfying assignment exists then
3      return NO
4  Let $\vec{f}^*$ be any frequency vector of polynomial size that satisfies conditions in step 2.
5  repeat
6      foreach $j \in I$ do
7          Check whether the following linear program is satisfiable for some frequency
           vector $\vec{f}$:
               $F_i(\vec{f},\overline{l}_i) \geq 0$ for all $i \in I \setminus \{j\}$
               $F_i(\vec{f},\overline{u}_i) \leq 0$ for all $i \in I \setminus \{j\}$
               $F_j(\vec{f},\overline{l}_j) > 0$ and $F_j(\vec{f},\overline{u}_j) < 0$.
           if no satisfying assignment exists then
8              If $F_j(\vec{f}^*,\overline{l}_i) = 0$, discard all modes $m$ such that $b_i^m/a_i^m \neq \overline{l}_i$ and set $\vec{f}_*(m) = 0$.
9              If $F_j(\vec{f}^*,\overline{u}_i) = 0$, discard all modes $m$ such that $b_i^m/a_i^m \neq \overline{u}_i$ and set $\vec{f}_*(m) = 0$.
10             Remove $j$ from $I$.
11 until no variable was removed from I in this iteration;
12 Check whether the following linear program is satisfiable for some frequency vector $\vec{f}$:
       $F_i(\vec{f},\overline{l}_i) > 0$ for all $i \in I$
       $F_i(\vec{f},\overline{u}_i) < 0$ for all $i \in I$.
   if no satisfying assignment exists or $I = \emptyset$ then
13     return NO
14 Let $\vec{f}_*$ be any frequency vector of polynomial size that satisfies conditions in step 12.
15 Let
       $\tau := \min_{i \in I} \left( \frac{\min\{x_0(i) - \overline{l}_i, \overline{u}_i - x_0(i)\}}{\max_m |b_i^m - a_i^m x_0(i)|}, \frac{\min(F_i(\vec{f}_*,\overline{l}_i), -F_i(\vec{f}_*,\overline{u}_i))}{(|\overline{l}_i| + |\overline{u}_i| + 2 \cdot \max_m |b_i^m/a_i^m|)(\sum_m a_i^m \vec{f}_*(m))^2} \right)$.
16 return the following periodic controller with period $|M|$: $m_k = (k \bmod |M|) + 1$ and
   $t_k = \vec{f}_*(m_k) \cdot \tau$.
```

It is easy to see that Algorithm 1 runs in polynomial time, because at least one critical variable is removed in each iteration of the loop between lines 5–11, one iteration checks the satisfiability of the linear conditions for at most N remaining variables, and each such a check requires calling a linear programming solver which runs in polynomial time. Finally, the requirements of steps 4 and 14 are satisfiable, because if a linear program has a solution then it has a solution of polynomial size (see, e.g. [14]). This shows that the size of the returned controller is always polynomial. □

Notice that the controller returned by Algorithm 1 has a polynomial-size minimum dwell time. We do not know whether finding a safe controller with the largest possible dwell time is decidable, nor is checking whether such a minimum dwell time can be greater than ≥ 1. We now show that the latter is PSPACE-hard, so it is unlikely to be tractable. Finally, this also implies PSPACE-hardness of checking whether a safe controller exists in the case the system is controlled using a digital clock, i.e. when all timed delays have to be a multiple of some given sampling rate $\Delta > 0$.

Theorem 4. *For a given MMS $\mathscr{H}$, hyperrectangular safe set S described by two points $\overline{l},\overline{u}$, starting point $\overline{x}_0 \in S$, checking whether there exists a S-safe controller with minimum dwell time ≥ 1 is* PSPACE*-hard.*

The proof reduces from the acceptance problem for linear bounded automata (LBAs). Now, notice that the periodic controller returned by Algorithm 1 just cycles forever over the set of modes in some fixed order which can be arbitrary. This allows us to extend the model by specifying an initial mode m_0 and a directed graph $G \subseteq M \times M$, which specifies for each mode which modes can follow it. Formally, we require any controller $\langle (m_1,t_1),(m_2,t_2),\ldots\rangle$ to satisfy $(m_i,m_{i+1}) \in G$ for all $i \geq 1$ and $m_1 = m_0$.

Corollary 1. *Deciding whether there exists a feasible S-safe controller for a given MMS $\mathscr{H}$ with a mode order specification graph G, initial mode m_0, a hyperrectangular safe set S given by two points $\overline{l}$ and $\overline{u}$ and an initial point $\overline{l} < \overline{x}_0 < \overline{u}$ can be done in polynomial time.*

4 Optimal Control

In this section we extend our results on S-safe controllability of MMS to a model with costs per time unit on modes. We will call this model *priced linear-rate multi-mode systems*. The aim is to find an S-safe controller with the minimum cost where the cost is either defined as the peak cost, the (long-time) average cost or a weighted sum of these.

Definition 4. *A priced linear-rate multi-mode system (MMS) is a tuple $\mathscr{H} = (M,N,A,B,\pi)$ where (M,N,A,B) is a MMS and $\pi : M \to \mathbb{R}_{\geq 0}$ is a cost function such that $\pi(m)$ characterises the cost per-time unit of staying in mode m.*

We define the (long-time) average cost of an infinite controller $\sigma = \langle (m_1,t_1),(m_2,t_2),\ldots\rangle$ as the long-time average of the cost per time-unit over time, i.e.

$$\mathrm{AvgCost}(\sigma) \stackrel{\text{def}}{=} \limsup_{k\to\infty} \frac{\sum_{i=1}^{k} \pi(m_i)\cdot t_i}{\sum_{i=1}^{k} t_i}.$$

For the results to hold it is crucial that lim sup is used in this definition instead of lim inf. In the case of minimising the average cost, it is more natural to minimise its lim sup anyway, which intuitively corresponds to its reoccurring maximum value. On the other hand, the peak cost is simply defined as $\mathrm{PeakCost}(\sigma) \stackrel{\mathrm{def}}{=} \sup_{\{k:\, t_k>0\}} \pi(m_k)$.

The following example shows that the approach of [10] which minimises the number of heaters being switched on at any time does not in general minimise the peak cost.

Example 2. Consider three zones, each with an identical heater inside. The heater can either be off, or in setting 1 with energy cost 1, or in a higher output setting 2 with energy cost 3. This generates a priced MMS with 3 variables $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$ and $3^3 = 27$ modes $M = \{m_1, m_2, m_3, \dots\}$, but we will focus only on 10 modes important to us. The table below shows for each mode m and variable $\mathbf{x}_i$ the value of the constants b_i^m as well as the cost of that mode. We assume that all a_i^m-s are equal to 1. The safe value interval for each variable is $[1,2]$, i.e. $\overline{l}_i = 1$, $\overline{u}_i = 2$ for all i.

Modes	m_1	m_2	m_3	m_4	m_5	m_6	m_7	m_8	m_9	m_{10}	...
b_1^m	0	2	0	0	2	2	0	5	0	0	...
b_2^m	0	0	2	0	2	0	2	0	5	0	...
b_3^m	0	0	0	2	0	2	2	0	0	5	...
π (cost)	0	1	1	1	2	2	2	3	3	3	...

It is easy to see that just by using modes m_1, m_2, m_3, m_4, in which at most one heater is in setting 1 and the rest are switched off, the system will not remain in the safe set, because for any frequency vector $\vec{f}$ we have $\sum_i F_i(\vec{f}, \overline{l}_i) = -1 - 2\vec{f}(m_1) < 0$ and so $F_i(\vec{f}, \overline{l}_i) < 0$ for some i while all of them should be > 0. This means that there is no safe controller with peak cost 1. On the other hand, using only modes m_8, m_9, m_{10}, in which exactly one heater is in setting 2 and the others are off, has a safe controller. It suffices to use frequency vector $\vec{f}$ with $\frac{1}{3}$ for each of these modes, and get $F_i(\vec{f}, \overline{l}_i) = \frac{2}{3}$ and $F_i(\vec{f}, \overline{u}_i) = -\frac{1}{3}$ for all i. This shows that the maximum number of heaters that has to be on while the system is safe is 1. However, this controller has peak cost 3, while there is a controller which has peak cost 2. It suffices to use the modes m_5, m_6, m_7 instead, in which exactly two heaters are on in setting 1, and using the same frequency vector $\vec{f}$ for these new modes get $F_i(\vec{f}, \overline{l}_i) = \frac{1}{3}$ and $F_i(\vec{f}, \overline{u}_i) = -\frac{2}{3}$ for all i.

We answer the following question for the priced MMSs.

Definition 5 (Optimal Controllability). *Given a priced MMS $\mathcal{H}$, a hyperrectangular safe set S defined by two points $\overline{l}$ and $\overline{u}$, an initial point $\overline{x}_0 \in S$ such that $\overline{l} < \overline{x}_0 < \overline{u}$, and constants $\varepsilon, \mu_{avg}, \mu_{peak} \geq 0$, find an S-safe controller σ for which $\mu_{avg}\,\mathrm{AvgCost}(\sigma) + \mu_{peak}\,\mathrm{PeakCost}(\sigma)$ is at most ε higher than its minimum.*

Another example priced MMS in [16] shows that such a weighted cost does not always increase with the increase in the peak cost. The algorithm that we devise for this problem is designed to cope with systems where the set of modes is large and given implicitly like in [10], where the input is a list of heaters with different output levels and energy costs. Each heater is placed in a different zone and any possible on/off combination of the heaters gives us a different mode in our setting. This leads to exponentially many modes in the size of the input. The cost of a mode is the sum of the energy cost of all heaters switched on in that particular mode. We try to deal with this setting by using binary search and a specific narrowing down technique to consider only the peak costs for which the weighted cost can be (ε-)optimal. Unfortunately our algorithm will not run in time polynomial in the number of heaters, but the techniques used can reduce the running time in practice. An algorithm that is only required to run in time polynomial in the number of modes is much simpler and can be found in [16]. This is because the number of different possible peak costs is at most equal to the number of modes, and for each of such peak cost p checking the weighted cost of the subset of all the modes with their cost at most equal to p can be done in time polynomial in the number of modes. Let us now fix a MMS with costs $\mathscr{H} = (M,N,A,B,\pi)$, the safe set S and a starting point $\overline{x}_0$ in the interior of S.

Algorithm 2. Finds an (ε-)optimal S-safe feasible controller from a given $\overline{x}_0 \in S$.

```
Input: A priced MMS $\mathscr{H}$, two points $\overline{l}$ and $\overline{u}$ that define a hyperrectangle
       $S = \{\overline{x} : \overline{l} \leq \overline{x} \leq \overline{u}\}$ and an initial point $\overline{x}_0 \in S$ such that $\overline{l} < \overline{x}_0 < \overline{u}$, error bound
       $\varepsilon > 0$, and constants $\mu_{\text{avg}}$ and $\mu_{\text{peak}}$ which define the weighted cost of a controller.
Output: NO if no $S$-safe feasible controller exists from $\overline{x}_0$, and an periodic such controller
        $\sigma$ for which $\mu_{\text{peak}}\,\text{PeakCost}(\sigma) + \mu_{\text{avg}}\,\text{AvgCost}(\sigma)$ is ($\varepsilon$-)optimal, otherwise.
1  min-size := 1;
2  repeat
3      min-size := 2 · min-size;
4      Pick minimal $p$ such that $M_{\leq p}$, the set of all modes with cost at most $p$, has size at
       least min-size.
5      Call Algorithm 1 for the set of modes $M_{\leq p}$.
6  until min-size < $|M|$ and the call returned NO;
7  if the last call to Algorithm 1 returned NO then
8      return NO.
9  Perform a binary search to find the minimal $p$ such that $M_{\leq p}$ is feasible using the just
   found upper bound on the minimal feasible set of modes.
10 Modify Algorithm 1 by adding the objective function Minimise $\sum_{m \in M} \vec{f}_m \pi(m)$ to the
   linear program at line 12. Let OptAvgCost($M'$) be the value of this objective when
   Algorithm 1 is called for the set of modes $M'$.
11 $p' := p' + \frac{\mu_{\text{avg}}}{\mu_{\text{peak}}}\,\text{OptAvgCost}(M_{\leq p})$;
12 repeat
13     $p' := p' + \frac{\mu_{\text{avg}}}{\mu_{\text{peak}}}\left(\text{OptAvgCost}(M_{\leq p}) - \text{OptAvgCost}(M_{\leq p'})\right)$;
14 until $p'$ decreases;
15 Pick a peak value $p^* \in [p, p']$ for which $\mu_{\text{peak}} p^* + \mu_{\text{avg}}\,\text{OptAvgCost}(M_{\leq p^*})$ is the smallest.
16 return a slightly perturbed version of the periodic controller returned by the modified
   version of Algorithm 1 called for the set of modes $M_{\leq p^*}$ in order to make it safe without
   increasing the objective by more than $\varepsilon$.
```

Theorem 5. *Algorithm 2 finds a S-safe feasible periodic controller with the weighted cost defined by the peak and average cost coefficients μ_{peak} and μ_{avg} within ε of its minimum.*

Proof. Let $M_{\leq p}$ denote the set of modes with cost at most p. First, to find the minimum peak cost among all S-safe controllers we can first order all the modes according to their costs and then the algorithm makes a binary search on the possible peak cost p That is, it guesses the initial p and checks whether $M_{\leq p}$ has a S-safe controller; if it does not then it doubles the value of p and if it does then halves the value of p. In may be best to start with a small value of p first, because the bigger p is, the bigger is the set of modes and the slower is checking its feasibility.

Second, to find the minimum average cost among all S-safe controllers, notice that the average cost of the periodic controller returned by Algorithm 1 based on the frequency vector $\vec{f}_*$ is $\sum_{m \in M} \vec{f}_*(m)\pi(m)$. Therefore, if we find an implementable frequency vector which minimises that value, then we will also find a safe controller with the minimum average-cost among all periodic safe controllers. This can be easily done by adding the objective *Minimise* $\sum_m \vec{f}(m)\pi(m)$ to the linear program at line 12 of Algorithm 1. However, using similar techniques as in Theorem 1 we can show that no other controller can have a lower average-cost. Just like in the proof of Theorem 1, we denote the frequency of being in mode m up to the k-th timed action by $f_k^{(m)}$ and pick such numbers $\langle j_k \rangle_{k \in \mathbb{N}}$ so that $f_{j_k}^{(m)}$ converges for every m and denote this limit by $\vec{f}(m)$. It can be shown that $\vec{f}$ is an implementable frequency vector. Now, the key observation is the fact that $\text{AvgCost}(\sigma) = \limsup_{k \to \infty} \sum_{m \in M} f_k^{(m)} \pi(m) \geq \limsup_{k \to \infty} \sum_{m \in M} f_{j_k}^{(m)} \pi(m) = \sum_m \vec{f}(m)\pi(m)$, based on the fact that lim sup of a subsequence is at most equal to the lim sup of the whole sequence. Therefore, the minimum average-cost of any controller σ cannot be lower than the minimum value of $\sum_m \vec{f}(m)\pi(m)$ over all an implementable frequency vectors $\vec{f}$. Finally, it may happen that this minimum can only obtained on the boundary of the region defined using strict inequalities at line 12 of Algorithm 1. In such a case, one can perturb the solution so it satisfies all the strict inequalities and its objective is within ε of the optimum (see, e.g. [14]).

For any set of modes $M' \subseteq M$, let $\text{OptAvgCost}(M')$ denote the minimum average cost when only modes in M' can be used. Now, if $\mu_{\text{peak}} = 0$ then it suffices to compute the optimal average cost for the whole set of modes to find the minimum weighted cost. Otherwise, to find a safe controller with the minimum value of $\mu_{\text{peak}} \text{PeakCost}(\sigma) + \mu_{\text{avg}} \text{AvgCost}(\sigma)$ the algorithm first finds a feasible set of modes with the minimum peak cost and let us denote that peak cost by p_{min}. If $\mu_{\text{avg}} = 0$ then this suffices. Otherwise, observe that from the definition the cost of each mode is always nonnegative and so the average cost has to be as well. Even if we assume that the average cost is equal to 0 for some larger set of modes with peak cost p, the weighted cost will at least be equal to $\mu_{\text{peak}} p$ as compared to $\mu_{\text{peak}} p_{min} + \mu_{\text{avg}} \text{OptAvgCost}(M_{\leq p_{min}})$, which gives us an upper bound on the maximum value of p worth considering to be $p' = p_{min} + \frac{\mu_{\text{avg}}}{\mu_{\text{peak}}}(\text{OptAvgCost}(M_{\leq p_{min}}))$. But now we can check the actual value of $\text{OptAvgCost}(M_{\leq p'})$ instead of assuming it is $= 0$ and calculate again a new bound on the maximum peak value worth considering and so on. To generate modes on-the-fly in the order of increasing costs, we can use Dijkstra algorithm with a priority queue. □

Coming back to Example 1, and assuming the cost of switching on a heater is equal to 1 in both zones, we can compute $\vec{f} = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$ to be the frequency vector with the smallest possible average cost of $2/3$. The controller which $\vec{f}$ generates is not safe though as $F_1(\vec{f}, 12°\text{C}) = 0$. However, by changing the frequencies in $\vec{f}$ by a small amount, we can obtain a safe controller. For instance, $\vec{f}' = (0.3, 0.35, 0.35)$ has average cost 0.7 which is just 5% higher than the minimum. Assuming the initial temperature of each room is 20°C, the total time duration τ of each period of the periodic controller σ based on $\vec{f}'$ can be computed to be 0.003 and the controller itself $\sigma = \langle(m_1, 0.009), (m_2, 0.0105), (m_3, 0.0105), \ldots\rangle$.

5 Numerical Simulations

We have implemented Algorithms 1 and 2 in Java using a basic implementation of the simplex algorithm as their underlying linear program solver. The tests were run on Intel Core i5 1.7 GHz with 1GB of RAM. The examples are based on the model of a

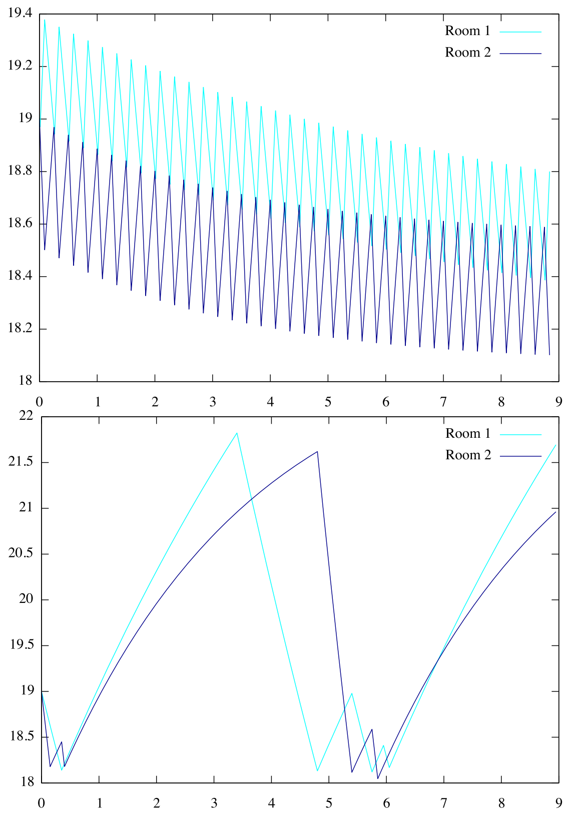


Fig. 1. Comparison of the temperature evolution under the optimal and lazy control in a system consisting of two zones. The safe temperature is between 18°C and 22°C. On the top, a periodic controller with the minimum peak cost which was then optimised for the minimal average-cost. On the bottom, the behaviour of the lazy controller. The y-axis is temperature in °C and the x-axis measures time in hours. The optimal controller uses 3 modes and its minimum dwell time is 43 seconds. The lazy controller uses 5 different modes and its minimum dwell time is 180 seconds.

building with decoupled zones as in [10] and were randomly generated with exactly the same parameters as described there. We implemented also a simple lazy controller to compare its peak and average energy consumption to our optimal controller. Simply asking the lazy controller to let the temperature oscillate around the minimum comfort temperature in each room is risky and causes high peak costs, so our "lazy" controller uses a different approach. It switches any heater to its minimum setting if its zone has reached a temperature in the top 5% of its allowable value range. On the other hand, if the temperature in a zone is in the bottom 5% of its allowable value range, then the lazy controller finds and switches its heater to the minimum setting that will prevent the temperature in that zone dropping any further. However, before it does that, it first checks whether there are any zones with their temperature above 10% of their allowable value range and switches them off. This tries to minimise the number of heaters being switched on at the same time and thus also tries to minimise the peak cost.

We have tested our systems for a building with eight zones and each heater having six possible settings, which potentially gives $6^8 > 10^6$ possible modes. Zones parameters and their settings were generated using the same distribution as described in [10] and the outside temperature was set to 10°C. The simulation of the optimal and the lazy controller was performed with a time step of three minutes and the total time duration of nine hours.

First, in Figure 1 we can compare the difference in the behaviour of the optimal controller as compared to the lazy one in the case of just two zones. In the case of the optimal controller, we can see that the temperature in each zone stabilises around the lower safe bound by using a constant switching between various modes. On the other hand, for the lazy controller the temperature oscillates between the lower and upper safe value, which wastes energy. The peak cost was 15 kW for the optimal controller and 18.43 kW for the lazy one, while the average energy usage was 13.4 kW and 15.7 kW, respectively. This gives 23% savings in the peak energy consumption and 17% savings in the average energy consumption. Note that any safe controller cannot use more than 16.9 kW of energy on the average, because otherwise it would exceed the upper comfort temperature for one of the rooms, so the maximum possible savings in the average energy consumption cannot exceed 26%. For a building with eight rooms, the running time of our algorithm, which crucially depends on how many modes are necessary to ensure safe controllability of the system, was between less than a second to up to a minute with an average equal to 40 seconds. The lazy controller was found to have on the average 40% higher peak cost than the optimal controller and 15% higher average-cost. In the extreme cases it had 70% higher peak cost and 22% higher average-cost. Again, the reason why the lazy controller did better in the average energy consumption than the peak consumption is that the comfort zone is so narrow and any safe controller cannot waste too much energy without violating the upper comfort temperature in one of the rooms.

6 Conclusions

We have proposed and analysed a subclass of hybrid automata with dynamics govern by linear differential equations and no guards on transitions. This model strictly generalises the models studied by Nghiem et al. in [10] in the context of peak minimisation for energy consumption in buildings. We gave a sufficient and necessary condition for the existence of a controller that keeps the state of the system within a given safe set at all times as well as an algorithm that find such a controller in polynomial time. We also analysed an extension of this model with costs per time unit associated with modes and gave an algorithm that constructs a safe controller which minimises the peak cost, the average cost or any cost expressed as a weighted sum of these two. Finally, we implemented some of these algorithms and showed how they perform in practice.

From the practical point of view, the future work will involve turning the prototype implementation of the algorithms in this paper into a tool. Our model could be extended by adding disturbances and interactions between zones to the dynamics of the model like in [11]. It is not clear whether the formula for the time duration of each period of the safe controller can be stated explicitly in such a setting as it was done at line 15 of Algorithm 1. The special cases which can be looked at are the initial state being on the boundary of the safe set and checking whether Theorem 1 also holds for all non-Zeno controllers not just for controllers with a positive minimum dwell time. An interesting problem left open is the decidability of finding a safe controller with the minimum dwell time above a fixed constant.

Acknowledgments. We would like to thank the anonymous reviewers who helped to improve this paper. This work was supported by EPSRC grant EP/G050112/2.

References

1. Albadi, M.H., El-Saadany, E.F.: Demand response in electricity markets: An overview. In: Proc. of IEEE Power Engineering Society General Meeting, pp. 1–5 (June 2007)
2. Alur, R., Trivedi, A., Wojtczak, D.: Optimal scheduling for constant-rate mulit-mode systems. In: Proc. of Hybrid Systems: Computation and Control 2012 (2012)
3. Alur, R.: Formal verification of hybrid systems. In: Proc. of International Conference on Embedded Software 2011, pp. 273–278 (2011)
4. Alur, R., Courcoubetis, C., Henzinger, T.A., Ho, P.-H.: Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In: Grossman, R.L., Ravn, A.P., Rischel, H., Nerode, A. (eds.) HS 1991 and HS 1992. LNCS, vol. 736, pp. 209–229. Springer, Heidelberg (1993)
5. Camacho, E.F., Bordons, C., Camacho, E.F., Bordons, C.: Model predictive control, vol. 303. Springer, Berlin (1999)
6. Giua, A., Seatzu, C., Van der Mee, C.: Optimal control of switched autonomous linear systems. In: Proc. of the 40th Conference on Decision and Control, pp. 2472–2477 (2001)
7. Henzinger, T.A.: The theory of hybrid automata. In: Proc. of the 11th LICS Symposium (1996)
8. Henzinger, T.A., Kopke, P.W., Puri, A., Varaiya, P.: What's decidable about hybrid automata? Journal of Comp. and Sys. Sciences 57, 94–124 (1998)
9. Ma, Y., Borrelli, F., Hencey, B., Coffey, B., Bengea, S., Haves, P.: Model predictive control for the operation of building cooling systems. In: Proc. of American Control Conference, pp. 5106–5111 (2010)
10. Nghiem, T.X., Behl, M., Mangharam, R., Pappas, G.J.: Green scheduling of control systems for peak demand reduction. In: Proc. of Conference on Decision and Control (2011)

11. Nghiem, T.X., Behl, M., Pappas, G.J., Mangharam, R.: Green scheduling for radiant systems in buildings. In: Proc. of Conference on Decision and Control (2012)
12. Oldewurtel, F., Ulbig, A., Parisio, A., Andersson, G., Morari, M.: Reducing peak electricity demand in building climate control using real-time pricing and model predictive control. In: Proc. of 49th IEEE Conference on Decision and Control, pp. 1927–1932 (2010)
13. Shuzhi, S., Sun Ge, Z.: Switched linear systems: Control and design. Springer (2005)
14. Schrijver, A.: Theory of linear and integer programming. John Wiley & Sons, Inc., New York (1986)
15. TRFund Energy Study. Understanding PECO's general service tariff
16. Wojtczak, D.: Optimal scheduling for linear-rate multi-mode systems. CoRR, abs/1302.4406 (2013)

Author Index