Basic Research in Computer Science

BRICS NS-02-3 Vogler & Larsen (eds.): MTCS '02 Proceedings

Preliminary Proceedings of
the 3rd International Workshop on

Models for Time-Critical Systems

MTCS '02

Brno, Czech Republic, August 24, 2002

Walter Vogler
Kim G. Larsen
(editors)

BRICS Notes Series **NS-02-3**

ISSN 0909-3206 **August 2002**

See back inner page for a list of recent BRICS Notes Series publications. Copies may be obtained by contacting:

BRICS
Department of Computer Science
University of Aarhus
Ny Munkegade, building 540
DK–8000 Aarhus C
Denmark

Telephone: +45 8942 3360
Telefax: +45 8942 3255
Internet: BRICS@brics.dk

BRICS publications are in general accessible through the World Wide Web and anonymous FTP through these URLs:

`http://www.brics.dk`
`ftp://ftp.brics.dk`
This document in subdirectory `NS/02/3/`

Third International Workshop on Models for Time-Critical Systems

---

MTCS 2002

Brno, Czech Republic
August 24, 2002

Editors:
Walter Vogler, University of Augsburg, Germany
Kim Larsen, University of Aalborg, Denmark

This is a preliminary version.
The final version is considered for publication in
Electronic Notes in Theoretical Computer Science
http://www.elsevier.nl/locate/entcs

Table of Contents

Foreword

A large class of systems can be specified and verified by abstracting away from the temporal aspects. In time-critical systems, instead, time issues become essential. Their correctness depends not only on which actions a system can perform but also on their execution time. Due to their importance, time-critical systems have attracted the attention of a considerable number of computer scientists from various research areas.

This volume contains the *preliminary proceedings* of the 3rd International Workshop on Models for Time-Critical Systems (MTCS 2002); MTCS 2002 was held on August 24, 2002 as one of seven satellite workshops co-located with the 13th International Conference on Concurrency Theory (CONCUR 2002), held in Brno (Czech Republic) August 20-23, 2002.

The first workshop, MTCS 2000, was held in State College (Pennsylvania, USA) on 26 August 2000, co-chaired by Flavio Corradini and Paola Inverardi, while the second workshop, MTCS 2001, was held in Aalborg (Denmark) on August 25, 2001, co-chaired by Flavio Corradini and Walter Vogler. As for the first two workshops, the objectives of MTCS 2002 were (i) to present and discuss promising new proposals on models for time-critical systems, ranging from theory to practice and (ii) to promote interaction between different research areas in the field of time-critical systems. Despite its focus on time-critical systems, MTCS 2002 was also open for time-related issues in general, e.g. performance in systems where time is not critical for the functional behaviour.

The eight papers in this volume were selected for presentation by the Program Committee from submissions received in response to a Call for Papers. The final versions of these papers are considered for publication in Electronic Notes in Theoretical Computer Science: http://www.elsevier.nl/locate/entcs.

This volume also includes a short abstract of the contribution by the invited speaker Walter Vogler (University of Augsburg). Many thanks are due to the other invited speaker P. S. Thiagarajan (Chennai Mathematical Institute) and to the members of the Program Committee as well as their sub-referees for their accurate work. We would also like to thank Michael Mislove for his help during the editorial process for these proceedings and to BRICS for the publication of these preliminary proceedings. Finally, our thanks go to Petr Jancar and Mojmir Kretinsky (CONCUR 2002 Conference Chairs) and Antonin Kucera (Workshops Coordinator) for the opportunity they gave us to organize MTCS 2002 and for their support.

Walter Vogler, University of Ausburg, Germany
Kim G. Larsen, University of Aalborg, Denmark

MTCS 2002 - Program Committee

Rajeev Alur (USA)	Jos Baeten (Netherlands)
Frank de Boer (Netherlands)	Flavio Corradini (Italy)
Kim Larsen (Denmark)	Gerald Lüttgen (UK)
Jeff Magee (UK)	Andrea Maggiolo-Schettini (Italy)
Paul Pettersson (Sweden)	Steve Schneider (UK)
Bran Selic (USA)	Joseph Sifakis (France)
P. S. Thiagarajan (India)	Walter Vogler (Germany)

Measuring the Performance of Asynchronous Systems in the PAFAS Approach

Walter Vogler [1]

Institut für Informatik
Universität Augsburg
Germany

Flavio Corradini [2]

Dipartimento di Informatica
Università dell'Aquila
Italy

Abstract

PAFAS (Process Algebra for Faster Asynchronous Systems) is a process algebra where actions are assumed to occur within a given time bound, which for simplicity is taken to be 1. A testing-based faster-than relation is presented that compares asynchronous systems according to their worst-case efficiency [4,3]. Main results are: the resulting pre-order based on the more realistic real-valued time is the same as the pre-order based on the simpler integer-valued time – so we use the latter; the faster-than relation can be characterized with some sort of refusal traces. A larger example studying implementations of a buffer can be found in [1].

While the testing definition is qualitative, we point out that it can also be seen as considering a quantitative performance measure. Then we adapt the PAFAS-approach to a setting, where user behaviour is known to belong to a very specific, but often occurring class of request-response behaviours, and show how to determine an asymptotic performance measure for finite-state processes [2].

References

[1] F. Corradini, M. Di Berardini, and W. Vogler. PAFAS at work: comparing the worst-case efficiency of three buffer implementations. In Y.T .Yu and T.Y. Chen, editors, *2nd Asia-Pacific Conference on Quality Software APAQS 2001*, pages 231–240, IEEE, 2001.

[1] Email: vogler@informatik.uni-augsburg.de
[2] Email: flavio@univaq.it

This is a preliminary version and considered for the final proceedings, to be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

[2] F. Corradini and W. Vogler. Measuring the performance of asynchronous systems with PAFAS. Technical Report 2002-4, University of Augsburg, 2002, URL: `http://www.Informatik.Uni-Augsburg.DE/skripts/techreports/`.

[3] F. Corradini, W. Vogler, and L. Jenner. Comparing the worst-case efficiency of asynchronous systems with PAFAS. Internal Report 31-2000, University of L'Aquila, 2000, URL: `http://w3.dm.univaq.it/~flavio`.

[4] L. Jenner and W. Vogler. Comparing the efficiency of asynchronous systems. In J.-P. Katoen, editor, *AMAST Workshop on Real-Time and Probabilistic Systems*, LNCS 1601, 172–191. Springer, 1999. Revised full version as [3].

An Algebraic Approach for Compiling Real-Time Programs

Alvaro E. Arenas [1,2]

Laboratorio de Cómputo Especializado
Universidad Autónoma de Bucaramanga
Bucaramanga, Colombia

Abstract

Compiler Verification has been identified as a vital process in the implementation of correct safety-critical systems. We extend here Hoare's refinement-algebra approach to compilation in order to include real-time languages in which processes interact asynchronously via communication queues. The existence of unique fixed-points is exploited to verify the implementation of crucial operators such as asynchronous input, delay and timeout.

1 Introduction

In the development of safety-critical systems, compiler correctness represents an essential process. Since many safety-critical systems have timing constraints, some approaches have been expanded to include the concept of time [7,13]. This paper extends the refinement-algebra approach proposed by Hoare in [8,10] to model compilation for real-time languages in which processes communicate asynchronously via communication queues called shunts. A shunt can be seen as a directed channel with the capability of buffering messages. As the sender transmits a message, it stores it into the corresponding shunt and proceeds with the execution. When the receiver reads a shunt, it takes the oldest message deposited in it. However, if the shunt is empty, the receiver is blocked until a message arrives. The main advantage of this asynchronous mechanism is the loose coupling it provides between system parts: a sender is never blocked because a receiver is not ready to communicate. This communication scheme is adopted by several asynchronous models such as versions of CSP [12] and SDL [15].

[1] Thanks to He Jifeng for proposing exploration of fixed points in verifying the implementation of real-time programs. I am grateful to Jeff Sanders for valuable suggestions. This work was partially funded by the Colombian Research Council (Colciencias-BID).

[2] Email:aearenas@bumanga.unab.edu.co

This is a preliminary version and considered for the final proceedings, to be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

The work presented here is part of research aiming to achieve means of formally dealing with compilation and scheduling of real-time programs within a single framework [1]. The main contribution of this paper consists in devising a strategy for verifying the implementation of operators that require "waiting" for the occurrence of an event, such as buffered input, delay or timeout. The strategy is based on the existence of unique fixed points in recursive equations.

Section 2 describes the source programming language and presents its main algebraic laws. Next, section 3 introduces the target language and develops some algebraic properties of machine-level programs. Section 4 formalises the compiling correctness relation that must hold between source and target code and illustrates the compilation of constructors for sequential programs. Finally, section 5 gathers some concluding remarks.

2 The Source Language

Our programming language is a concurrent language with asynchronous communication and real-time facilities. Its syntax is given by the following description:

$$P ::= \bot \mid II \mid \overline{x} := \overline{e} \mid s\,!\,e \mid s\,?\,x \mid \Delta d \mid [d]P$$
$$\mid P;P \mid P \sqcap P \mid P \trianglelefteq b \trianglerighteq P \mid \texttt{while}(b,P) \mid [P \rhd^{s}_{d} P] \mid P \parallel P$$

where P stands for a process, $\overline{x}$ is a list of variables, x is a variable, s is a shunt, $\overline{e}$ is a list of expressions, e is an expression, b is a Boolean expression, and d is a time expression.

The chaotic process $\bot$ is the worst one; its execution is arbitrary and beyond control. The skip process II does nothing, terminating immediately. The multiple assignment $\overline{x} := \overline{e}$, where $\overline{x}$ is a list of distinct variables and $\overline{e}$ an equal-length list of expressions, evaluates the components of $\overline{e}$ and stores these results simultaneously into list $\overline{x}$, preserving the original ordering of the elements. We assume here that the evaluation of an expression always delivers a result and does not change the value of any variable, i.e. no *side-effect* is allowed. The output $s\,!\,e$ writes the value of the expression e into the output shunt s, leaving all program variables unchanged. The input $s\,?\,x$ reads the oldest message from shunt s and stores it into variable x. If the shunt is empty, the process is blocked until a message arrives. We adopt the realistic premise that all communicating processes take time, the amount of time consumed by the instruction not being specified.

Composition $P;Q$ represents a process that executes P first and, at termination of P, starts with the execution of Q. It is assumed that there is no delay associated with the transfer of control from P to Q. Process $P \sqcap Q$ represents the non-deterministic choice between the participating processes. The conditional $P \trianglelefteq b \trianglerighteq Q$ represents a choice between alternatives P and Q in accordance with the value of Boolean expression b; it behaves like P if b is true, and like Q if b is false. It is assumed that some arbitrary time is

spent in the evaluation of b. The iteration $\texttt{while}(b, P)$ executes process P while condition b is true, and terminates when the condition is false. It is also assumed that some time is spent in each iteration evaluating expression b.

The delay process Δd is guaranteed to wait for a minimum of d time units before terminating. The process $[d]P$ behaves as P and its execution does not take more than d time units. Another useful real-time constructor is the timeout process $[P \rhd_d^s Q]$, which monitors shunt s for d time units; if there is a message in s during that time, it executes process P, otherwise it executes process Q. Finally, the parallel composition of P and Q, denoted by $P \parallel Q$, describes the concurrent execution of the two processes. Each process has its own program state, which is inaccessible to its partner, and interacts with its partner and the external world via communication through shared shunts.

In previous work [2], we have given a specification-oriented semantics to our language and derived its main algebraic laws. The semantic is constructed by following the predicative approach described in [9], where a process is modelled as a predicate that describes all the *observations* that it can generate. In [1] we have proposed notation $P \equiv Q$ to denote that processes P and Q are semantically equivalent and proved that all derived laws are sound with respect to the model. Let us now introduce a subset of the laws useful in verifying the compiler described in later sections.

Laws for primitive programs coincide with classical laws for imperative sequential programs and communicating processes.

Law 2.1 *Laws for Primitive Programs*

(1) $P;\ II \equiv II;\ P \equiv P$

(2) $\bot;\ P \equiv \bot$

(3) $x := e;\ s\,!f(x) \equiv x := e;\ s\,!f(e)$

(4) $x, y := e, y \equiv x := e$

(5) $x := e;\ x := f(x) \equiv x := f(e)$

(6) $s\,?\,y;\ x := y \equiv s\,?\,x;\ y := x$.

Let us explain some of the above laws. Law 2.1 (2) expresses that once a process is out of control, its sequential composition with another process does not redeem the situation. In our formal model [2], an assignment may take time, but the amount of time consumed by the instruction is not specified; this allows us to derive law 2.1 (5).

We define an ordering relation $P \sqsubseteq Q$ to mean that Q is at least as deterministic as P. It is defined in terms of choice as $P \sqsubseteq Q \ \widehat{=}\ (P \sqcap Q) \equiv P$. All compound processes are monotic with respect to the ordering relation [1].

The following laws describe some properties of the real-time operators.

Law 2.2 *Laws for Real-Time Operators*

(1) $\Delta d_1;\ \Delta d_2 \equiv \Delta(d_1 + d_2)$

(2) $[d_1]P \sqsubseteq [d_2]P$ provided $d_2 \leq d_1$

(3) $[P \rhd_1^s [P \rhd_d^s Q]] \equiv [P \rhd_{d+1}^s Q]$

(4) $[[P \rhd_0^s R] \rhd_d^s Q] \equiv [P \rhd_d^s Q]$.

2.1 Some Auxiliary Processes

We introduce here some auxiliary processes useful in reasoning about process behaviour. The idle process Δ represents a process that may terminate at any arbitrary time without changing any variable or shunt. The conditional process $(P \lhd b \rhd Q)$ selects one alternative depending on the value of expression b; if b is true, it acts like process P, otherwise it behaves like Q. It differs from the conditional of our programming language in that it is assumed that the evaluation of b does not take time. The miracle program, denoted by $\top$, stands for a product that can never be used because its conditions of use are impossible to satisfy. The assumption of b, denoted by $b^\top$, can be regarded as a miracle test: it behaves like II if b is true; otherwise it behaves like $\top$. By contrast, the assertion of b, denoted by $b_\bot$, also behaves like II if b is true, otherwise it fails, behaving like $\bot$. The next law illustrates the use of assumption/assertion.

Law 2.3 *Laws Applying Assumption and Assertion*

(1) $b^\top;\ (P \lhd b \rhd Q) \equiv b^\top;\ P$ (3) $(s \neq \langle\rangle)^\top;\ [P \rhd^s_d\ Q] \equiv (s \neq \langle\rangle)^\top;\ \Delta;\ P$

(2) $x := e;\ (x = e)^\top \equiv x := e$ (4) $(s = \langle\rangle)^\top;\ [P \rhd^s_0\ Q] \equiv (s = \langle\rangle)^\top;\ \Delta;\ Q$

The declaration $\mathtt{var}\, x$ introduces new program variable x and permits x to be used in the portion of the program that follows it. The complementary operation, $\mathtt{end}\, x$, terminates the region of permitted use for variable x.

Let X be the name of a recursive program we wish to construct, and $F(X)$ a function on the space of processes denoting the intended behaviour of the program. We can show that the space of processes forms a complete lattice [1]. Notation $\mu\, X.F(X)$ stands for the least fixed point of function F and notation $\nu X.F(X)$ denotes the greatest fixed point of F. The following law illustrates the main properties of these operators.

Law 2.4 *Fixed Point Laws*

(1) $F(\mu\, X.F(X)) \equiv \mu\, X.F(X)$ (3) $F(\nu X.F(X)) \equiv \nu X.F(X)$

(2) $F(Y) \sqsubseteq Y \ \Rightarrow\ \mu\, X.F(X) \sqsubseteq Y$ (4) $F(Y) \sqsupseteq Y \ \Rightarrow\ \nu X.F(X) \sqsupseteq Y$.

The iteration $b * P$ can be defined as the least fixed point of the equation $\mu\, X.((P;\ X) \lhd b \rhd II)$. Typical laws for the loop include the following.

Law 2.5 *Loop Laws*

(1) $b^\top;\ b * P \equiv b^\top;\ P;\ b * P$ (2) $(\neg\, b)^\top;\ b * P \equiv (\neg\, b)^\top$.

There is an interesting case in which the least and greatest fixed points coincide, as described below.

Theorem 2.6 *Unique Fixed Point*
Let $F(X)$ be a monotonic function on the space of processes. If it is guaranteed that there is a delay of at least one time unit before invoking the recursion, then the fixed point of F is unique. □

3 The Target Language

Our target machine has a rather simple architecture, consisting of a store for instructions $m : Addr \to Instr$, modelled as a function from the set of addresses to the set of machine instructions; a program counter $pc : Addr$ that points to the current instruction; and a data stack $st : \mathrm{seq}\,\mathbb{Z}$, used to hold temporary values. The target language is an *intermediate-representation* language close to, but more abstract than the final machine code. Following tradition, the machine instructions are represented by updates to machine components. These instructions are introduced in Table 1.

Let us explain some of the instructions. Instruction $\texttt{LD}(x)$ has variable x as its operand; its execution pushes the value of x onto the evaluation stack, and increases the value of the program counter pc by 1. Symbol $+\!\!+$ denotes concatenation of sequences; $\texttt{last}.st$ stands for the last element of sequence st; $\texttt{front}.st$ corresponds to the sequence obtained from eliminating last element of st. Instruction $\texttt{ST}(x)$ pops the value at the top of the stack into variable x, and then passes the control to the next instruction; the requirement of having at least one element in the stack is expressed as an initial assumption in the instruction. Instructions $\texttt{EV}(e), \texttt{EVB}(b)$ and $\texttt{EVT}(d)$ are used to evaluate integer, Boolean and time expressions respectively; the instructions push the result of evaluating the expression onto the top of the stack and increment the program counter by one. When non-integer values are stored into the stack, they are translated into the appropriate representation by using a *representation function* R_T, of type $T \to \mathbb{Z}$ for each basic type T, as presented in [14,13]. Arithmetic instructions are introduced by means of the ADD and SUB instructions; the operation $\texttt{front2}.st$ obtains the front of $\texttt{front}.st$; similarly, the operation $\texttt{last2}.st$ obtains the last element of $\texttt{front}.st$. Comparison of the last two elements of the evaluation stack is introduced by the instructions LE and LT. Instructions JP, JPF and JPT are used for unconditional and conditional jump respectively. The instruction DUP duplicates the value stored at the top of the evaluation stack st. The output instruction $\texttt{OUT}(s)$ sends the value on top of the stack through shunt s, taking that value out of the stack. The input instruction $\texttt{IN}(s)$ is executed only when shunt s is not empty; it inputs the oldest message from s and leaves it at the top of the stack. Instruction $\texttt{TST}(s)$ tests whether there is a message in shunt s. Instruction $\texttt{STM}(s)$ stores in top of the stack the time stamp of the oldest unread message of s. The TIM instruction reads the current time and places it on top of the stack; it is simply a specification that the hardware implementator must guarantee.

The target language is a distinguished subset of the modelling language. The assignment statements are "timed" assignments so that time passes while an instruction executes. Let $\mathcal{T} : Instr \to Time$ be a function denoting the duration of executing a machine instruction such that $\mathcal{T}(\texttt{INSTR}) > 0$ for $\texttt{INSTR} \in Instr$. Notation $\mathcal{T}$ is used later to define the execution time of blocks of machine code.

Table 1
The Target Language

$\texttt{LD}(x)$	$\widehat{=}\ pc, st := pc + 1, st +\!\!+ \langle x \rangle$
$\texttt{ST}(x)$	$\widehat{=}\ (\#st \geq 1)^{\top};\ pc, st, x := pc + 1, \texttt{front}.st, \texttt{last}.st$
$\texttt{EV}(e)$	$\widehat{=}\ pc, st := pc + 1, st +\!\!+ \langle e \rangle$
$\texttt{EVB}(b)$	$\widehat{=}\ pc, st := pc + 1, st +\!\!+ \langle R_{\mathbb{B}}.b \rangle$
$\texttt{EVT}(d)$	$\widehat{=}\ pc, st := pc + 1, st +\!\!+ \langle R_{Time}.d \rangle$
$\texttt{ADD}$	$\widehat{=}\ (\#st \geq 2)^{\top};\ pc, st := pc + 1, \texttt{front2}.st +\!\!+ \langle \texttt{last2}.st + \texttt{last}.st \rangle$
$\texttt{SUB}$	$\widehat{=}\ (\#st \geq 2)^{\top};\ pc, st := pc + 1, \texttt{front2}.st +\!\!+ \langle \texttt{last2}.st - \texttt{last}.st \rangle$
$\texttt{LE}$	$\widehat{=}\ (\#st \geq 2)^{\top};\ pc, st := pc + 1, \texttt{front2}.st +\!\!+ \langle 1 \lhd \texttt{last}.st \leq \texttt{last2}.st \rhd 0 \rangle$
$\texttt{LT}$	$\widehat{=}\ (\#st \geq 2)^{\top};\ pc, st := pc + 1, \texttt{front2}.st +\!\!+ \langle 1 \lhd \texttt{last}.st < \texttt{last2}.st \rhd 0 \rangle$
$\texttt{JP}(l)$	$\widehat{=}\ pc := l$
$\texttt{JPF}(l)$	$\widehat{=}\ (\#st \geq 1)^{\top};\ pc, st := (l \lhd \texttt{last}.st = 0 \rhd pc + 1), \texttt{front}.st$
$\texttt{JPT}(l)$	$\widehat{=}\ (\#st \geq 1)^{\top};\ pc, st := (l \lhd \texttt{last}.st = 1 \rhd pc + 1), \texttt{front}.st$
$\texttt{DUP}$	$\widehat{=}\ (\#st \geq 1)^{\top};\ pc, st := pc + 1, st \frown \langle last(st) \rangle$
$\texttt{OUT}(s)$	$\widehat{=}\ (\#st \geq 1)^{\top};\ s\,!\,\texttt{last}.st;\ pc, st := pc + 1, \texttt{front}.st$
$\texttt{IN}(s)$	$\widehat{=}\ (\overleftarrow{s} \neq \langle\rangle)^{\top};\ \texttt{var}\,x;\ s\,?\,x;\ pc, st := pc + 1, st +\!\!+ \langle x \rangle;\ \texttt{end}\,x$
$\texttt{TST}(s)$	$\widehat{=}\ pc, st := pc + 1, st +\!\!+ \langle 1 \lhd \overleftarrow{s} = \langle\rangle \rhd 0 \rangle$
$\texttt{STM}(s)$	$\widehat{=}\ (\overleftarrow{s} \neq \langle\rangle)^{\top};\ pc, st := pc + 1, st +\!\!+ \langle stamp(\overleftarrow{s}) \rangle$
$\texttt{TIM}$	$\widehat{=}\ pc, st := pc + 1, st +\!\!+ \langle t \rangle$ where $t \in [t_\alpha, t_\omega]$ and t_α, t_ω stand for the time when starts and finishes the execution of the instruction

3.1 Execution of Target Programs

The execution of a target program is represented by the repetition of a set of machine instructions. In this part we formalise such concepts, borrowing some elements from [9].

Definition 3.1 *Labelled Instruction*
Let $\texttt{INSTR} : \mathit{Instr}$ be a machine instruction as defined in Table 1 and $l : \mathit{Addr}$ a machine location. Labelled instruction $l : \texttt{INSTR}$ expresses that instruction

$\texttt{INSTR}$ is executed when the program counter has value l. It is defined as $l : \texttt{INSTR} \mathrel{\widehat{=}} (\texttt{INSTR} \lhd pc = l \rhd II)$.

Labelled instructions are used to model the possible actions during the execution of a target program. The fact that the executing mechanism can perform one of a set of actions according to the current value of the program counter can be modelled by a program of the form $l_1 : \texttt{INSTR}_1 \,[]\, l_2 : \texttt{INSTR}_2 \,[]\, \cdots \,[]\, l_n : \texttt{INSTR}_n$ where locations $l_1, \cdots, l_n$ are pairwise disjoint and operator [] denotes the *assembly* of machine programs.

Definition 3.2 *Assembly and Continuation Set*
– Let C be a machine program consisting only of labelled instruction $l : \texttt{INSTR}$. Then, C is an assembly program with continuation set $L.C = \{l\}$.
– Let C and D be assembly programs with disjoint continuation sets $L.C$ and $L.D$ respectively. The assembly program $(C \,[]\, D)$ and its continuation set are defined as follows:

$$C \,[]\, D \quad \mathrel{\widehat{=}} (C \lhd pc \in L.C \rhd D) \lhd (pc \in L.C \cup L.D) \rhd II$$

$$L.(C[]D) \mathrel{\widehat{=}} L.C \cup L.D \ .$$

The continuation of assembly C denotes its set of valid locations. The value of the program counter determines the instruction to be executed.

Law 3.3 *Program Counter and Assembly Program*
Let $C = (l_1 : \texttt{INSTR}_1 \,[]\, l_2 : \texttt{INSTR}_2 \,[]\, \cdots \,[]\, l_n : \texttt{INSTR}_n)$ be an assembly program. Then $(pc = l_i \wedge l_i \in L.C)^\top;\ C \equiv (pc = l_i \wedge l_i \in L.C)^\top;\ \texttt{INSTR}_i$.

The execution of an assembly program is modelled as a loop which iterates the program as long as the program counter remains within the continuation set.

Definition 3.4 *Execution*
Let C be an assembly program. *Execution* of program C is defined as follows: $C^* \mathrel{\widehat{=}} (pc \in L.C) * C$. The evaluation of the guard in the loop does not consume time. All execution time overheads are accounted for in the machine instructions.

4 Compiling Sequential Programs

This section specifies a compiler that translates a sequential program into a target program represented as an assembly of single machine instructions whose behaviour represents an improvement with respect to that of the original source program. We also derive the execution time of each target program generated by the compiler.

Definition 4.1 *Compilation*
The correct compilation of a program is represented by a predicate $\mathcal{C}(P, a, C, z)$ where P is the source program; C is a machine program stored in the code

memory m, consisting of an assembly of single machine instructions; a and z stand for the initial and final addresses of program C, respectively. Predicate $\mathcal{C}(P, a, C, z)$ is formally defined by the following refinement:

$$\mathcal{C}(P, a, C, z) \mathrel{\widehat{=}} P \sqsubseteq (\mathtt{var}\, pc, st;\ (pc = a)^{\top};\ C^{*};\ (pc = z)_{\bot};\ \mathtt{end}\, pc, st)\,.$$

The declaration $\mathtt{var}\, pc, st$ introduces the machine components. The assumption $(pc = a)^{\top}$ expresses that program counter pc should be positioned at location a at the beginning of execution of C. The assertion $(pc = z)_{\bot}$ states the obligation to terminate with program counter pc positioned at location z. Notation $\mathcal{T}_{\mathcal{C}}(P)$ is used to denote the worst-case execution time of the machine code that compiler specification $\mathcal{C}$ associates to source program P.

The compiler is specified by defining predicate $\mathcal{C}$ recursively over the syntax of sequential source programs. Correctness of the compiling relation follows from the algebraic laws of the language. We omit the proof for the classical sequential operators, since it follows lines similar to those of the untimed case, and refer the reader to [1]. We outline the proof for input and timeout operators.

Assignment $x := e$ is implemented by a piece of code that evaluates expression e and stores the result into the corresponding program-variable store. Note that the duration of an assignment was unspecified at source level, however the code implementing it has an exact duration equal to the addition of the duration of each participating machine instruction.

Theorem 4.2 *Assignment Compilation*
$\mathcal{C}\,(\,x := e,\ a, (a : \mathtt{EV}(e)\,[]\,a^{+1} : \mathtt{ST}(x)), a + 2)\ .$
$\mathcal{T}_{\mathcal{C}}(x := e)\ =\ \mathcal{T}(\mathtt{EV}) + \mathcal{T}(\mathtt{ST})\ .$

Notation l^{+i} : $\mathtt{INSTR}$ states that machine instruction $\mathtt{INSTR}$ is located at position $l + i$. For simplicity, we are assuming that the evaluations of the integer expressions all have the same duration. We can determine the duration of evaluating an expression by using techniques for simplifying expressions.

Skip is implemented as an empty segment of code. Obviously, the duration of the code implementing the skip is zero. Let us assume that II also denotes a machine program with an empty location set, i.e $L.II = \emptyset$.

Theorem 4.3 *Skip Compilation*
$\mathcal{C}\,(\,II,\ a,\ II,\ a\,)\ .$
$\mathcal{T}_{\mathcal{C}}(II)\ =\ 0\ .$

The output process is implemented by a piece of code that evaluates the expression to be transmitted and then sends the value to the corresponding shunt. The duration time of the implementation is equal to the addition of its constituent machine instructions.

Theorem 4.4 *Output Compilation*
$\mathcal{C}\,(\,s\,!\,e,\ a,\ (a : \mathtt{EV}(e)\,[]\,a^{+1} : \mathtt{OUT}(s)),\ a + 2\,)\ .$

$\mathcal{T}_{\mathcal{C}}(s\,!\,e) \;=\; \mathcal{T}(\texttt{EV}) + \mathcal{T}(\texttt{OUT})$.

The implementation of input instruction $s\,?\,x$ is split into three parts. The first one, code A below, tests whether there exists a message in shunt s. The second part, code B below, jumps back to execute code A again if there is no message in s. Finally, in case there is a message in s, code I does input the oldest message and finishes storing it into variable x. To determine the execution time of the implementation of an input is an infeasible problem, since the arrival of messages into a shunt depends on the environment's behaviour; however, we can estimate the execution time of the input implementation if we know that the shunt is not empty.

Theorem 4.5 *Input Compilation*

Let $\quad A = (\; a : \texttt{TST}(s)\,[]\,a^{+1} : \texttt{JPF}(a^{+3})\;)$, $B = (\; a^{+2} : \texttt{JP}(a)\;)$

and $\quad I = (\; a^{+3} : \texttt{IN}(s)\,[]\,a^{+4} : \texttt{ST}(x))$.

Then $\quad \mathcal{C}\;(\; s\,?\,x,\;\; a,\; (A\,[]\,B\,[]\,I),\; a+5\;)$.

If s is not empty, then $\quad \mathcal{T}_{\mathcal{C}}(s\,?\,x) = \mathcal{T}(A) + \mathcal{T}(I)$

$$= \mathcal{T}(\texttt{TST}) + \mathcal{T}(\texttt{JPF}) + \mathcal{T}(\texttt{IN}) + \mathcal{T}(\texttt{ST})\ .$$

Proof. We use a novel strategy in which the uniqueness of the fixed point for recursive equations plays an important role. Let us start by defining a function F that portrays the execution of code $(A\,[]\,B\,[]\,I)$.
Let $\quad M = pc, st$, $C = (A\,[]\,B\,[]\,I)$, $START = \texttt{var}\,M;\; (pc = a)^{\top}$
$END = (pc = a^{+4})_{\perp};\; \texttt{end}\,M$, $END_0 = (pc = a \vee pc = a^{+4})_{\perp};\; \texttt{end}\,M$,
$G(X) = A^*;\; ((B^*;\; X) \lhd pc = a^{+2} \rhd I^*)$ and
$F(X) = START;\; G(X);\; END_0$.
Function F starts by executing code A. Depending on the value of the program counter at the end of the execution of A, it proceeds either to execute code I or to execute code B and then to invoke parameter program X. As all involved instructions take time, function F is time-guarded for variable X. From theorem 2.6, it follows that F has a unique fixed point. Our strategy consists in proving first that $s\,?\,x$ is a pre-fixed point of F, i.e. $s\,?\,x \;\sqsubseteq\; F(s\,?\,x)$, concluding by the strongest fixed point law, law 2.4 (4), that $s\,?\,x \;\sqsubseteq\; \mu\,X \bullet F(X)$. Then we proceed by proving that $(START;\; C^*;\; END)$ is a post-fixed point of F, i.e. $F(START;\; C^*;\; END) \;\sqsubseteq\; (START;\; C^*;\; END)$, concluding by the weakest fixed point law, law 2.4 (2), that $\mu\,X \bullet F(X) \;\sqsubseteq\; (START;\; C^*;\; END)$. The desired result follows from the transitivity of the refinement relation. Complete proof of this theorem is presented in [1]. □

The strategy employed in the implementation of the input program can be used to prove the implementation of constructors that require to wait for the occurrence of an event, namely delay and timeout. The code implementing the delay program Δd is divided into two parts: codes S and T. Execution

of code S determines the time when delay Δd should finish: it is equal to the addition of the current time to the value of time parameter d, leaving the result on top of the evaluation stack. Code T compares the current time with the value at the top of the stack, in order to determine whether the delay has expired.

Theorem 4.6 *Delay Compilation*

Let $S = (\, a : \texttt{TIM} \,[]\, a^{+1} : \texttt{EVT}(d) \,[]\, a^{+2} : \texttt{ADD} \,)$

and $T = (\, a^{+3} : \texttt{DUP} \,[]\, a^{+4} : \texttt{TIM} \,[]\, a^{+5} : \texttt{LT} \,[]\, a^{+6} : \texttt{JPT}(a^{+3}) \,)$.

Then $\mathcal{C}\,(\,\Delta d,\ a,\ (S \,[]\, T),\ a+7\,)$.

$\mathcal{T}_{\mathcal{C}}(\Delta d) \ =\ d + \mathcal{T}(S) + \mathcal{T}(T)$.

Let us now turn to the implementation of compound processes. Sequential composition can be compiled componentwise, having as target code the assembly of its components.

Theorem 4.7 *Sequential Composition Compilation*

Let $\mathcal{C}\,(P,\ a,\ C,\ h)$, $\mathcal{C}\,(Q,\ h,\ D,\ z)$ and $(L.C \cap L.D) = \emptyset$.

Then $\mathcal{C}\,(P; Q,\ a,\ (C \,[]\, D),\ z)$.

$\mathcal{T}_{\mathcal{C}}(P; Q) \ =\ \mathcal{T}_{\mathcal{C}}(P) + \mathcal{T}_{\mathcal{C}}(Q)$.

The compilation of a timed conditional includes an initial piece of code that evaluates the corresponding guard and then, depending on the result of the evaluation, chooses one of the participating programs.

Theorem 4.8 *Conditional Compilation*

Let $B = (\, a : \texttt{EVB}(b) \,[]\, a^{+1} : \texttt{JPF}(h) \,)$, $\mathcal{C}\,(P,\ a^{+2},\ C,\ z)$, $\mathcal{C}\,(Q,\ h,\ D,\ z)$,

$(L.C \cap L.D) = \emptyset$ and $(L.B \cap L.C \cap L.D) = \emptyset$.

Then $\mathcal{C}\,(P \trianglelefteq b \trianglerighteq Q,\ a,\ (C \,[]\, B \,[]\, D),\ z)$.

$\mathcal{T}_{\mathcal{C}}(P \trianglelefteq b \trianglerighteq Q) \ =\ \mathcal{T}(\texttt{EVB}) + \mathcal{T}(\texttt{JPF}) + \max(\mathcal{T}_{\mathcal{C}}(P), \mathcal{T}_{\mathcal{C}}(Q))$.

The iteration program is implemented by a piece of machine code that evaluates the guard. In case the guard holds, the body of the program is executed. Once it has terminated, it jumps back to repeat the whole process. To determine the execution time of the iteration program, it is necessary to know the upper bound on the possible number of iterations.

Theorem 4.9 *Iteration Compilation*

Let $B = (\, a : \texttt{EVB}(b) \,[]\, a^{+1} : \texttt{JPF}(z) \,)$, $J = (\, j : \texttt{JP}(a) \,)$

$\mathcal{C}\,(P,\ a^{+2},\ C,\ j)$ and $(L.B \cap L.J \cap L.C) = \emptyset$.

Then $\mathcal{C}\,(\texttt{while}(b, P),\ a,\ (B \,[]\, C \,[]\, J),\ z)$.

Let T be the maximum number of iterations of the program $\texttt{while}(b, P)$. Then, $\mathcal{T}_{\mathcal{C}}(\texttt{while}(b,P)) \;=\; T * (\mathcal{T}(B) + \mathcal{T}(C) + \mathcal{T}(J)) + \mathcal{T}(B)$.

The timeout $[P \;\rhd_d^s\; Q]$ is implemented by a machine program that monitors shunt s for at most d time units. If a message arrives in that period of time, the program jumps to execute the code associated to program P. After d time units, if a message has not arrived on shunt s, the program jumps to execute the code associated to program Q.

Theorem 4.10 *Timeout Compilation*

Let $S = (\, a : \texttt{TIM} \,[]\, a^{+1} : \texttt{EVT}(d) \,[]\, a^{+2} : \texttt{ADD}\,)$,

$E = (\, a^{+3} : \texttt{TST}(s) \,[]\, a^{+4} : \texttt{JPF}(a^{+10})\,)$,

$T = (\, a^{+5} : \texttt{DUP} \,[]\, a^{+6} : \texttt{TIM} \,[]\, a^{+7} : \texttt{LT} \,[]\, a^{+8} : \texttt{JPF}(h)\,)$,

$J = (\, a^{+9} : \overline{\texttt{JP}(a^{+3})}\,)$,

$M = (\, a^{+10} : \texttt{STM}(s) \,[]\, a^{+11} : \texttt{LE} \,[]\, a^{+12} : \texttt{JPF}(h)\,)$,

$\mathcal{C}\,(P,\;\; a^{+13},\, B,\, z)$, $\mathcal{C}\,(Q,\;\; h,\, D,\, z)$, $(L.B \cap L.D) = \emptyset$,

$(L.S \cap L.E \cap L.T \cap L.J \cap L.M \cap L.B \cap L.D) = \emptyset$ and

$C = (S \,[]\, E \,[]\, T \,[]\, J \,[]\, M \,[]\, B \,[]\, D)$.

Then $\mathcal{C}\,([P \;\rhd_d^s\; Q],\;\; a,\, C,\, z)$.

$$\mathcal{T}_{\mathcal{C}}([P \;\rhd_d^s\; Q]) \;=\; d + \mathcal{T}(S) + \mathcal{T}(E) + \mathcal{T}(T) + \mathcal{T}(J) + \mathcal{T}(M) + \max(\mathcal{T}_{\mathcal{C}}(P), \mathcal{T}_{\mathcal{C}}(Q)) \;.$$

Proof. Let us first explain the implementation of $[P \;\rhd_d^s\; Q]$, assuming that $\mathcal{C}\,(P,\; a^{+13},\; B,\; z)$ and $\mathcal{C}\,(Q,\; h,\; D,\; z)$. Code S refers to the evaluation of the timeout parameter; it reads the current time, and then adds to it the value of parameter d, leaving the result at the top of the evaluation stack. Code E determines whether there exists messages in the shunt. In case there are no messages in the shunt, code T compares the current time with the value at the top of the stack, to determine if a timeout has occurred. In case of a timeout, the program jumps to location h to execute piece of code D. If there is no timeout, the program proceeds with the execution of code J, which simply jumps to repeat code E. If there is a message in shunt s, code M determines if it arrived before the timeout; to do so, it obtains the time stamp of the oldest unread message, and compares it with the timeout value that is stored at the top of the evaluation stack. In case of the stamp being less than the timeout, code M jumps to location a^{+13}, where it continues with the execution of B. In case of the stamp being greater than the timeout value, a timeout has happened (although some messages could have arrived after the timeout, in which case they are not considered), code M jumps then to location h, the initial location of D.

In the proof, we follow a strategy similar to the one used for proving the input instruction. It starts with the definition of a function F that mimics

the execution of code C and then exploits the uniqueness of its fixed point to get the desired result.

$$\begin{array}{ll} \text{Let} & START = M;\ (pc = a)^{\top} \quad , \quad END = (pc = z)^{\top};\ \texttt{end}\, M \quad , \\ & G(X) = E^*;\ [\ (\ T^*;\ (J^*;\ X \lhd pc = a^{+9} \rhd D^*)\ ) \quad \lhd pc = a^{+6} \rhd \\ & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (\ M^*;\ (B^* \lhd pc = a^{+13} \rhd D^*)\ )\ ] \\ & F(X) = START;\ S^*;\ G(X);\ END\ . \end{array}$$

Invocation of X in $F(X)$ is preceded by instructions that take time. Then, by Theorem 2.6, it follows that F has a unique fixed point. The proof strategy consists in showing first that $[P \rhd_d^s Q] \sqsubseteq F([P \rhd_d^s Q])$. Such proof follows by induction on time parameter d, using law 2.2 (3). Then, according to the strongest fixed point law, it follows that $[P \rhd_d^s Q] \sqsubseteq \mu X \bullet F(X)$. The second part consists in showing that $(START;\ C^*;\ END)$ is a post fixed point of F, $F(START;\ C^*;\ END) \sqsubseteq (START;\ C^*;\ END)$. By the weakest fixed point law, it follows that $\mu X \bullet F(X) \sqsubseteq (START;\ C^*;\ END)$. The result arises from transitivity of the refinement relation. □

Our compilation process restricts the compilation of deadline to the case in which it is the outermost operator. Let notation $\mathcal{C}_D(P,\ a,\ C,\ z)$ stand for $([D]P \sqsubseteq (\texttt{var}\, pc, st;\ (pc = a)^{\top};\ [D]C^*;\ (pc = z)_{\perp};\ \texttt{end}\, pc, st))$. The following theorem illustrates the compilation of deadline.

Theorem 4.11 *Deadline Compilation*

$$\begin{array}{ll} \text{Let} & \mathcal{C}(P\,, a, C, z) \quad \text{and} \quad \mathcal{T}_{\mathcal{C}}(P) \leq D. \\ \text{Then} & \mathcal{C}_D(P,\ a,\ C,\ z). \\ \mathcal{T}_{\mathcal{C}}([D]P) = \mathcal{T}_{\mathcal{C}}(P) & . \end{array}$$

We are following an approach similar to [6] by considering the compilation of deadline as a sort of annotation on the target code, annotation that will be used in the later stage of scheduling analysis.

We have not dealt with compilation of concurrent programs. Intended future work includes extending the approach in order to enable compilation of concurrency by modelling the scheduling of parallel machine programs into a uniprocessor machine. We plan to define a priority-based scheduler that takes into consideration the deadline associated with each program, assuming that the set of programs has passed some schedulability test such as the *worst-case response time* analysis [11].

5 Concluding Remarks

Many authors have shown that unique fixed points arise naturally in real-time contexts when restricting the model to allow the progress of time [5]. In this paper we have taken advantage of this characteristic to verify the

implementation of a real-time language using the refinement-algebra approach to compilation.

Implementation of classical sequential constructors (such as assignment, sequential composition, conditional and iteration) has followed lines similar to those of the untimed case. The novelty in our work consisted in devising a strategy for proving the implementation of constructors that are required to wait for the occurrence of an event (input, delay and timeout). The strategy can be summarised as follows. Let P be the source program to be implemented and C the associated target code. In order to prove that C implements P, i.e. $P \sqsubseteq C$, we pursued the following steps: (1) finding a recursive function $F(X)$ that simulates the execution of C; (2) showing that the recursion in F is time-guarded, which implies uniqueness of its fixed point; (3) proving that P is a pre-fixed point of F, i.e. $P \sqsubseteq F(P)$; (4) proving that C is a post-fixed point of F, i.e. $F(C) \sqsubseteq C$. The result $P \sqsubseteq C$ followed from properties of fixed points and transitivity of the refinement relation.

The approach to prove correctness of compiling specification using algebraic laws was originally suggested by Hoare in [8,10]. Hoare's work was accomplished in the context of the ProCoS project [4] and has inspired several investigations. Notable is the work of Müller-Olm [14], that describes the design of a code generator translating the language of while programs — extended by communication statements and upper-bound timing — to the Inmos Transputer. Emphasis is put on modularity and abstraction of the proofs, which is achieved by constructing a hierarchy of increasingly more abstract views of the Transputer's behaviour, starting from bit-code level up to assembly levels with symbolic addressing.

In [7], a compilation is defined for a real-time sequential language with memory-mapped input and output commands. Both the source and target languages are modelled in the Interval Temporal Logic, and a set of algebraic laws are derived in a way similar to that presented here. The compilation process is simplified by representing the compilation of communication processes as a compilation of assignments to port variables.

Also influenced by Hoare's work, but using an alternative approach, Lermer and Fidge define compilation for real-time languages with asynchronous communication [13]. Their semantic model is based on the real-time refinement calculus of Hayes where communication is achieved by shared variables, and the language offers delay and deadline constructors. Our intermediate target language is very close to their target code, also modelled as a subset of the source language. The operation of composition of machine programs is achieved by means of an operation for merging loops, similar to our model of execution of machine programs. Although there are many similarities between the two studies, this approach does not define a compiling relation. Instead, a set of "compilation laws" are derived, where each law looks like a rule of the well-known refinement calculus of Morgan.

References

[1] A. E. Arenas. *Implementation of an Asynchronous Real-Time Programming Language*. D.Phil Thesis, Oxford University Computing Laboratory, 2000.

[2] A. E. Arenas. A Specification-Oriented Semantics for Asynchronous Real-Time Programming. In *Proceedings of CLEI'2001, XXVII Conferencia Latinoamericana de Informática*, 2001.

[3] J. P. Bowen, editor. *Towards Verified Systems*, volume 2 of *Real-Time Safety Critical Systems*. Elsevier, 1994.

[4] J. P. Bowen, C. A. R. Hoare, H. Langmaack, E.-R. Olderog, and A. P. Ravn. A ProCoS II Project Final Report: ESPRIT Basic Research Project 7071. *Bulletin of the European Association for Theoretical Computer Science (EATCS)*, 59:76–99, 1996.

[5] J. Davies and S. Schneider. Recursion Induction for Real-Time Processes. *Formal Aspects of Computing*, 5(6):530–553, 1993.

[6] C. Fidge, I. Hayes, and G. Watson. The Deadline Command. *Software*, 146(2):104–111, 1999.

[7] R. W. S. Hale. Program Compilation. In Bowen [3], chapter 7, pages 131–146.

[8] C. A. R. Hoare. Refinement Algebra Proves Correctness of Compiling Specifications. In C. C. Morgan and J. C. P. Woodcock, editors, *3rd Refinement Workshop*, Workshops in Computing, pages 33–48. Springer-Verlag, 1991.

[9] C. A. R. Hoare and He Jifeng. *Unifying Theories of Programming*. Prentice Hall Series in Computer Science, 1998.

[10] C. A. R. Hoare, He Jifeng, and A. Sampaio. Normal Form Approach to Compiler Design. *Acta Informatica*, 30(8):701–739, 1993.

[11] M. Joseph and P. K. Pandya. Finding Response Times in a Real-Time System. *The Computer Journal*, 29(5):390–395, 1986.

[12] K. N. Kumar and P. K. Pandya. ICSP and its Relationship with ACSP and CSP. In R. K. Shyamasundar, editor, *Foundations of Software Technology and Theoretical Computer Science*, volume 761 of *Lecture Notes in Computer Science*, pages 358–372. Springer-Verlag, 1993.

[13] K. Lermer and C. Fidge. A Formal Model of Real-Time Program Compilation. In J. P. Katoen, editor, *Formal Methods for Real-Time and Probabilistic Systems*, volume 1601 of *Lecture Notes in Computer Science*, pages 192–210. Springer-Verlag, 1999.

[14] M. Müller-Olm. *Modular Compiler Verification*, volume 1283 of *Lecture Notes in Computer Science*. Springer-Verlag, 1997.

[15] A. Sarma. Introduction to SDL-92. *Computer Networks and ISDN Systems*, 28(12):1603–1615, 1996.

Balanced timed regular expressions [1]

Eugene Asarin [2] Cătălin Dima [3]

VERIMAG, 2 av. de Vignate, 38610 Gières, France

Abstract

Several classes of regular expressions for timed languages accepted by timed automata have been suggested in the literature. In this article we introduce *balanced timed regular expressions with colored parentheses* which are equivalent to timed automata, and, differently from existing definitions, do not refer to clock values, and do not use additional operations such as intersection and renaming.

1 Introduction

Regular expressions are an important and convenient formalism to specify sets of discrete behaviors. The lack of such a language-based algebraic formalism for timed behaviors motivated several researchers to look for some variants of "timed regular expressions" equivalent to timed automata. The formalisms suggested in the literature give some solutions to this problem, but none of them is as perfect as classical "discrete" regular expressions of Kleene. In fact some of them [5,6,7] make use of information external to the language (such as clock values), while others [3,4] use heavy operations such as intersection and renaming.

In this paper we suggest a new approach to this problem. We introduce *balanced timed regular expressions with colored parentheses* which are equivalent to timed automata, do not refer to clock values, and do not use "bad" operations. The price to pay is a two-stage semantics of our expressions and a non-trivial algorithm for checking whether an expression is syntactically correct.

The structure of the paper is the following: in section 2 we recall some definitions concerning timed automata and timed languages and state the key technical result: Theorem 2.1 on transforming timed automata to a special form. In Section 3 we recall timed regular expressions from [4], discuss their

[1] Partially supported by the European community project IST-2001-35304 AMETIST

[2] Email: asarin@imag.fr. Home page: www-verimag.imag.fr/~asarin.

[3] Email: ctdima@imag.fr. Home page: www-verimag.imag.fr/~ctdima.

This is a preliminary version and considered for the final proceedings, to be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

drawbacks, and come up with our definition of *balanced timed regular expressions* and their semantics. At the end of this section we state the main result of the paper: balanced timed regular expressions are as expressive as timed automata. In section 4 we give an algorithm for checking correctness of an expression.

We are thankful to Christian Boitet for a motivating question and to Paul Caspi, Oded Maler and Matthieu Moy for useful discussions.

2 Timed automata

Behaviors of timed systems can be modeled by **timed words** over a set of symbols Σ. A timed word is a finite sequence of nonnegative numbers and symbols from Σ. For example, the sequence $1.2\,a\,1.3\,b$ denotes a behavior in which an action a occurs 1.2 time units after the beginning of the observation, and after another 1.3 time units action b occurs. The set of timed words over Σ can be organized as a monoid, and be represented as the *direct sum* of the monoid of nonnegative numbers $(\mathbb{R}_{\geq 0}, +, 0)$ and the free monoid $(\Sigma^*, \cdot, \varepsilon)$ [4,11]. We denote this monoid as $\mathsf{TW}(\Sigma)$. Note that in this monoid, concatenation of two reals amounts to summation of the reals. Thence, $a\,1.3 \cdot 1.7\,b = a(1.3 + 1.7)b = a\,3\,b$. The length $\ell(w)$ of a timed word w is the sum all the reals in it, e.g. $\ell(1.2\,a\,1.3\,b) = 1.2 + 1.3 = 2.5$.

A **timed automaton** [1] is a tuple $\mathcal{A} = (Q, \mathcal{X}, \Sigma, \delta, Q_0, Q_f)$ where Q is a finite set of *states*, $\mathcal{X}$ is a finite set of *clocks*, Σ is a finite set of *action symbols*, $Q_0, Q_f \subseteq Q$ are sets of *initial*, resp. *final* states, and δ is a finite set of tuples (*transitions*) (q, C, a, X, r) where $q, r \in Q$, $X \subseteq \mathcal{X}$, $a \in \Sigma \cup \{\varepsilon\}$ and C is a finite conjunction of *clock constraints* of the form $x \in I$, where $x \in \mathcal{X}$ and $I \subseteq [0, \infty[$ is an interval with integer (or infinite) bounds.

For each transition $(q, C, a, X, r) \in \delta$, the component C is called the *guard* of the transition, a is called the *action label* of the transition, and X is called the *reset component* of the transition. We will usually order the set of clocks $\mathcal{X} = \{x_1, \ldots, x_n\}$, and then identify each reset component X with the subset of indices of the clocks in X, that is, with $\{i \mid i \in [n], x_i \in X\}$ (here $[n]$ stands for $\{1, \ldots, n\}$).

The semantics of a timed automaton is given in terms of a *timed transition system* $\mathcal{T}(\mathcal{A}) = (\mathcal{Q}, \theta, \mathcal{Q}_0, \mathcal{Q}_f)$ where $\mathcal{Q} = Q \times \mathbb{R}^n_{\geq 0}$, $\mathcal{Q}_0 = Q_0 \times \{\mathbf{0}_n\}$, $\mathcal{Q}_f = Q_f \times \mathbb{R}^n_{\geq 0}$ and

$$\theta = \{(q, v) \xrightarrow{t} (q, v') \mid v'_i = v_i + t,\ \forall i \in [n]\} \cup$$
$$\{(q, v) \xrightarrow{a} (q', v') \mid \exists (q, C, a, X, q') \in \delta \text{ such that } v \models C \text{ and for all } i \in [n],$$
$$\text{if } i \in X \text{ then } v'_i = 0 \text{ and if } i \notin X \text{ then } v'_i = v_i\}$$

Informally, the automaton can make t-transitions representing time-passage in a state, in which all clocks advance by t, and discrete transitions, in which state changes. The discrete transitions are enabled when the "current clock

valuation" v satisfies the guard C of a certain tuple $(q, C, a, X, r) \in \delta$, and when they are executed, the clocks in the "reset component" X are set to zero.

A **run** in $\mathcal{T}(\mathcal{A})$ is a chain $(q^0, v^0) \xrightarrow{\xi_1} (q^1, v^1) \xrightarrow{\xi_2} \dots \xrightarrow{\xi_k} (q^k, v^k)$ of transitions from θ. An **accepting run** in $\mathcal{T}(\mathcal{A})$ is a run which starts in $\mathcal{Q}_0$ and ends in $\mathcal{Q}_f$ (the last transition should not be a t-transition). The **accepted language** of $\mathcal{A}$ is then the set of timed words which label some accepting run of $\mathcal{T}(\mathcal{A})$. Two timed automata are called **equivalent** iff they have the same language.

The first theorem gives a "normal form" to which each timed automaton can be brought. For it, we need several notations and conventions: for each clock $x_i \in \mathcal{X}$ and each transition $\tau = (q, C, a, X, r) \in \delta$, if $i \in X$ then we say that x_i *is reset on* τ. If the guard C contains a constraint $x_i \in I$ for some interval I, then we say that τ *checks* x_i, and we also write $(x_i \in I) \in C$. Note that the *true* guard contains no constraint.

Theorem 2.1 *Any timed automaton $\mathcal{A}$ is equivalent to a timed automaton $\overline{\mathcal{A}}$ in which on each accepting run, each clock is checked exactly once after each reset.*

Proof. We will decompose the construction into two steps as follows:

(i) We transform the given automaton into an automaton in which, each clock is checked against *the same* interval I, wherever it is checked.

(ii) We obtain the desired construction by splitting each clock x into two copies such that each copy is checked only once after each reset.

The first construction is accomplished along the following ideas: for each clock $x \in \mathcal{X}$ and interval I for which $(x \in I)$ occurs on some guard, we create a new clock x_I. The set of all clocks x_I will replace the clock x, in the following sense: each time x is reset, we reset all clocks x_I; then, each time $(x \in I)$ occurs on the guard of some transition, we replace this constraint with $(x_I \in I)$.

More formally we replace each transition $(q, C, a, X, r) \in \delta$ by the transition (q, C', a, X', r) with

$$C' = \bigwedge_{(x \in I) \in C} (x_I \in I)$$

and with $X' = \{x_I \mid x \in X\}$.

If we want to be practical, we should also remove the "unused clocks" as described in [10] (see also [9]).

The second construction is an adaptation of the convexity-based techniques of [4,2]. The rough idea is that in a chain of transitions which all contain $(x \in I)$ and do not reset x, all but the first and the last are "redundant". That happens because if the value of clock x on the first and on the last transition of the chain is in the interval I, then it must be in the interval

I throughout all the behavior of the timed automaton in between these two transitions – as a consequence of the convexity of I.

Therefore, we create two copies of each clock $x \in \mathcal{X}'$, denote them x^1 and x^2, and utilize them as follows: we reset both on each transition on which x is reset, we check $(x^1 \in I)$ on the first transition after the reset on which $(x \in I)$ occurs, we ignore all the other checks for $(x \in I)$ before the next reset, with the exception of the *last*, on which we check $(x^2 \in I)$. This last transition before a reset of x on which the constraint $(x \in I)$ occurs is not deterministically found, but rather "guessed".

The states of the resulting automaton will be tuples (q, ϕ) in which $q \in Q'$ and $\phi : \mathcal{X}' \to \{0, 1, 2\}$ gives our guess for the utilization of each clock x. An attribute $\phi(x) = 0$ means that, since the last reset for x we have never encountered the constraint $(x \in I)$ (remind that each clock is tested against a *unique* interval!). The attribute is set to $\phi(x) = 1$ the first time when such a constraint is met. It then remains 1 until we "guess" that, from now on and before the next reset for x, we will never take a transition with $(x \in I)$. Of course, such a guess is made on a transition on which $(x \in I)$ occurs, and this guess sets x's attribute to $\phi(x) = 2$.

We will also use two copies of each clock x, denoted x_1 and x_2. They will actually replace x on each transition, in the following sense: each time x is to be reset, we reset both x^1 and x^2. Subsequently, on the first transition which checks $(x \in I)$, we put the constraint $(x_1 \in I)$, and on the last transition which checks $(x \in I)$, we put the constraint $(x_2 \in I)$.

Formally, for each transition $(q, C, a, X, r) \in \delta'$ and for each state (q, ϕ) in the resulting automaton we draw a transition $\big((q, \phi), \overline{C}, a, Y, (r, \psi)\big)$ if and only if the following conditions are satisfied:

- $x \in X$ iff $x^1, x^2 \in Y$.
- If $\phi(x) = 2$ then $(x \in I) \notin C$ and also $(x^1 \in I), (x^2 \in I) \notin \overline{C}$.
- If $x \in X$ then $\phi(x) = 2$ and $\psi(x) = 0$.
- If $(x \in I) \in C$ then $\phi(x) \neq 2, \psi(x) \neq 0$.
- If $(x \in I) \in C$ and $\phi(x) = 0$ then either $\psi(x) = 1$ and $(x^1 \in I) \in \overline{C}$, or $\psi(x) = 2$ and $(x^1 \in I), (x^2 \in I) \in \overline{C}$.
- If $(x \in I) \in C$ and $\phi(x) = 1$ then either $\psi(x) = 1$ and $(x^1 \in I), (x^2 \in I) \notin \overline{C}$, or $\psi(x) = 2$ and $(x^2 \in I) \in \overline{C}$.

□

3 Expressions

3.1 Expressions from [4]

The class of *timed regular expressions* is built using the following grammar:

$$E ::= 0 \mid \varepsilon \mid \underline{\mathbf{t}}z \mid E + E \mid E \cdot E \mid E^* \mid \langle E \rangle_I, \tag{1}$$

where $z \in \Sigma \cup \{\varepsilon\}$ and I is an interval.

The semantics of timed regular expressions is in terms of timed words:

$$\begin{array}{cc} \|\underline{\mathbf{t}}z\| = \{tz \mid t \in \mathbb{R}_{\geq 0}\} & \|E_1 + E_2\| = \|E_1\| \cup \|E_2\| \\ \|E_1 \cdot E_2\| = \|E_1\| \cdot \|E_2\| & \|\langle E \rangle\|_I = \{\sigma \in \|E\| \mid \ell(\sigma) \in I\} \\ \|E^*\| = \|E\|^* & \|0\| = \emptyset\ , \qquad \|\varepsilon\| = \{\varepsilon\} \end{array}$$

We abuse notation and write $\langle E \rangle_\alpha$ for $\langle E \rangle_{[\alpha,\alpha]}$. We also denote $\Sigma_{\underline{\mathbf{t}}} = \Sigma \cup \{\underline{\mathbf{t}}\}$.

The following theorem shows a nice relationship between timed automata and timed regular expressions:

Theorem 3.1 ([4]) *The class of timed languages accepted by timed automata equals the class of timed languages accepted by timed regular expressions with* intersection *and* renaming*, that is, expressions generated by the grammar*

$$E ::= 0 \mid \varepsilon \mid \underline{\mathbf{t}}z \mid E + E \mid E \cdot E \mid E^* \mid \langle E \rangle_I \mid E \wedge E \mid [a \mapsto z]E,$$

where $z \in \Sigma \cup \{\varepsilon\}, a \in \Sigma$.

We interpret $\wedge$ as intersection, and $[a \mapsto z]$ as renaming of any occurrence of the symbol a with the symbol z. For example, $\big\|[a \mapsto b](\underline{\mathbf{t}}a)\| = \|\underline{\mathbf{t}}b\|$.

It was shown in [3] that intersection is necessary for representing timed automata. The example there is the timed regular expression $\underline{\mathbf{t}}a\langle \underline{\mathbf{t}}b\underline{\mathbf{t}}c\rangle_1 \wedge \langle \underline{\mathbf{t}}a\underline{\mathbf{t}}b\rangle_1\underline{\mathbf{t}}c$, which cannot be expressed without conjunction. The timed language accepted by this expression is

$$L_0 = \{t_1 a t_2 b t_3 c \mid t_1 + t_2 = 1, t_2 + t_3 = 1\}$$

Moreover, in [13] it was shown that renaming is necessary too, his example being the timed automaton in the figure 1.

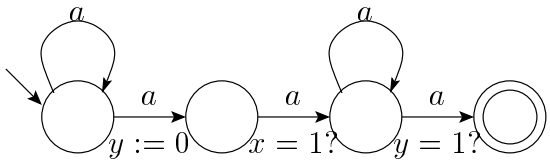

Fig. 1.

The language of this automaton equals

$$[x \mapsto a]\big((\underline{\mathbf{t}}a)^*\langle \underline{\mathbf{t}}x(\underline{\mathbf{t}}a)^*\rangle_1 \wedge \langle (\underline{\mathbf{t}}a)^*\underline{\mathbf{t}}x\rangle_1(\underline{\mathbf{t}}a)^*\big).$$

These results show that, in spite of their ease in use, timed regular expressions suffer from some expressiveness problems.

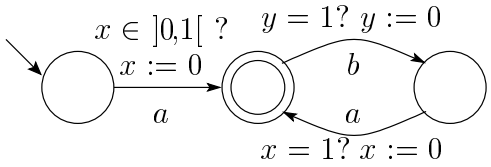

Fig. 2.

3.2 Colored parentheses

In [4], the authors suggest that, by employing "colored" parentheses, these drawbacks might be overcome. For example, L_0 would be specified by the expression $\langle \underline{\mathbf{t}}a \lfloor \underline{\mathbf{t}}b \rangle_1 \underline{\mathbf{t}}c \rfloor_1$. Similarly, the language of the timed automaton in Figure 1 would be specified by $\langle (\underline{\mathbf{t}}a)^* \lfloor \underline{\mathbf{t}}a \rangle_1 (\underline{\mathbf{t}}a)^* \rfloor_1$. Note that these expressions no longer use intersection or renaming.

But then, if we want to specify also cyclic behaviors, we cannot cope with the simple automaton on the figure 2. The expression with intersection for its language is

$$\langle \underline{\mathbf{t}}a \rangle_{]0,1[} \langle \underline{\mathbf{t}}b \underline{\mathbf{t}}a \rangle_1^* \wedge \langle \underline{\mathbf{t}}a \underline{\mathbf{t}}b \rangle_1^* \underline{\mathbf{t}}a$$

We conjecture that the semantics of this formula cannot be expressed with colored parentheses (in the sense of the suggestion from [4]).

Another problem with colored parentheses is that the set of syntactically correct expressions is non-context-free. Indeed, let us consider $\mathcal{J}$ an index set (the set of *colors*) and $card(\mathcal{J})$ sets of matching parentheses:

$$P_i = \left\{ `\langle^i`, `\rangle_I^i` \mid I \text{ interval }, I \subseteq [0, \infty[\right\} \text{ for all } i \in \mathcal{J} \tag{2}$$

$$\Pi = \bigcup_{i \in \mathcal{J}} P_i. \tag{3}$$

We may define $card(\mathcal{J})$ *deletion morphisms* (or *color filters*), $(\eta_i)_{i \in \mathcal{J}}$, $\eta_i : \Pi \to P_i$, each η_i deleting all parentheses not in P_i. E.g., for $P_{blue} = \left\{ `\langle`, `\rangle_I` \mid I \subseteq [0, \infty[\right\}$ and $P_{red} = \left\{ `\lfloor`, `\rfloor_I` \mid I \subseteq [0, \infty[\right\}$ $\eta_{blue}\big(\langle \lfloor \rangle_1 \langle \rfloor_1 \rangle_1\big) = \langle \rangle_1 \langle \rangle_1$.

For each set of parentheses $P_i = \left\{ `\langle^i`, `\rangle_I^i` \mid I \text{ interval}, I \subseteq [0, \infty[\right\}$, we denote by Δ_i the set of words with balanced parentheses[4] over P_i, which is generated by the following context-free grammar:

$$S ::= \varepsilon \mid \langle^i S \rangle_I^i \mid SS$$

The language of *balanced parentheses* over $\bigcup_{i=1}^n P_i$ is defined as follows:

$$L_{\text{par}} = \left\{ w \in \Pi^* \mid \text{ for each } i \in [1 \ldots n], \eta_i(w) \in \Delta_i \right\}$$

This language is unfortunately context-sensitive for $card(\mathcal{J}) \geq 2$: just consider the intersection of L_{par} with $\big(\langle\big)^* \big(\lfloor\big)^* \big(\rangle_1\big)^* \big(\rfloor_1\big)\big)^*$, which gives a language of the form $\{a^k b^l c^k d^l \mid k, l \in \mathbb{N}\}$, which is an easy prey to the Bar-Hillel

[4] This is a slight generalization of the notion of Dyck languages [14].

(pumping) lemma for context-free languages [14]. In a related paper [12], one of the authors investigates on the possibility to define regular expressions with colored parentheses by using a different concatenation operation.

3.3 Balanced timed expressions

We explain our approach with an example: consider again the language of the automaton from the figure 2. This language can be regarded as the union of all the "word expressions" of the kind

$$\langle \lfloor \underline{\mathbf{t}} a \rfloor_{]0,1[} \lfloor \underline{\mathbf{t}} b \rangle_1 \langle \underline{\mathbf{t}} a \rfloor_1 \lfloor \underline{\mathbf{t}} b \rangle_1 \dots$$

The following expression generates all these "word expressions":

$$E = \lfloor \underline{\mathbf{t}} a \rfloor_{]0,1[} + \langle \lfloor \underline{\mathbf{t}} a \rfloor_{]0,1[} \big(\lfloor \underline{\mathbf{t}} b \rangle_1 \langle \underline{\mathbf{t}} a \rfloor_1 \big)^* \lfloor \underline{\mathbf{t}} b \rangle_1 \underline{\mathbf{t}} a \rfloor_1 \tag{4}$$

Note that the resulting expression E contains the subexpression $\lfloor \underline{\mathbf{t}} b \rangle_1 \langle \underline{\mathbf{t}} a \rfloor_1$ in which the parentheses *are not balanced* – the first blue parenthesis is the right one. In order to give a sense to E and to other expressions of this kind we adopt a *two-step semantics* approach, in which we first build the classical semantics of the expression – i.e. a set of words over $\Sigma_{\underline{\mathbf{t}}} \cup \Pi$. In the second step we give timed semantics to these words, provided they have well-balanced parentheses. Hence, a balanced timed regular expression will be defined as an expression over $\Sigma_{\underline{\mathbf{t}}} \cup \Pi$ which generates only words with well-balanced parentheses, and its timed semantics will be the union of the semantics of these words.

Observe that this process is not "compositional", that is, we do not define the semantics of a balanced timed regular expression by induction on their structure. In fact, even the definition of the class of regular expressions with balanced parentheses is not a "structural" once. This is the point of difference with [12].

The problem of checking whether an expression generates only well-balanced words is the subject of the last section.

Formalization

For the sequel, we will work with n sets of colored parentheses P_i, as defined in (2) on the previous page. That is, we assume $\mathcal{J} = \{1, \dots, n\}$. We will utilize here the deletion morphism $\eta_i : (\Sigma_{\underline{\mathbf{t}}} \cup \Pi)^* \to P_i^*$, which deletes all symbols not in P_i, and the "partial" deletion morphism $\overline{\eta}_i : (\Sigma_{\underline{\mathbf{t}}} \cup \Pi)^* \to P_i^*$, which deletes from each word the symbols not in P_i or not in $\Sigma_{\underline{\mathbf{t}}}$. For example,

$$\begin{aligned} \eta_{blue}\big(\lfloor \underline{\mathbf{t}} a \rangle_1 \langle \underline{\mathbf{t}} a \rfloor_1 \big) &= \rangle_1 \langle \\ \overline{\eta}_{blue}\big(\lfloor \underline{\mathbf{t}} a \rangle_1 \langle \underline{\mathbf{t}} a \rfloor_1 \big) &= \underline{\mathbf{t}} a \rangle_1 \langle \underline{\mathbf{t}} a \end{aligned}$$

Definition 3.2 A **balanced word** over $\Sigma_{\underline{\mathbf{t}}}$ with parentheses from Π is a word $w \in (\Sigma_{\underline{\mathbf{t}}} \cup \Pi)^*$ such that $\eta_i(w) \in \Delta_i$ for all i. The set of balanced words over $(\Sigma_{\underline{\mathbf{t}}} \cup \Pi)^*$ is denoted $\mathcal{W}_\Pi(\Sigma)$.

Note that, for each $1 \leq i \leq n$ and $w \in \mathcal{W}_\Pi(\Sigma)$, $\overline{\eta}_i(w)$ is a *timed regular expression*. The **timed semantics** of a balanced word is the set of timed words σ which belong to the semantics of each $\overline{\eta}_i(w)$, regarded as a timed regular expression:

$$\|w\| = \left\{\sigma \in \mathsf{TW}(\Sigma) \mid \forall i \in \mathcal{J}, \sigma \in \|\overline{\eta}_i(w)\|\right\}$$

Our *regular expressions with colored parentheses* are classical regular expressions over $\Sigma_{\underline{\mathbf{t}}} \cup \Pi$, that is, generated by the grammar

$$E ::= 0 \mid a \mid \underline{\mathbf{t}} \mid \langle^i \mid \rangle_I^i \mid E + E \mid E \cdot E \mid E^*$$

where $a \in \Sigma$ and $\langle^i, \rangle_I^i \in P_i$, for some $1 \leq i \leq n$. As classical regular expressions, they have a *word semantics* in terms of languages over $(\Sigma_{\underline{\mathbf{t}}} \cup \Pi)^*$:

$$\begin{aligned} |a| &= \{a\} & |\underline{\mathbf{t}}| &= \{\underline{\mathbf{t}}\} \\ |0| &= \emptyset & |\langle^i| &= \{\langle^i\}, \\ |\rangle_I^i| &= \{\rangle_I^i\} & |E_1 + E_2| &= |E_1| \cup |E_2| \\ |E_1 \cdot E_2| &= |E_1| \cdot |E_2| & |E^*| &= |E|^* \end{aligned}$$

Definition 3.3 A **balanced (timed) regular expression** with colored parentheses in Π is a regular expression with colored parentheses whose word semantics contains only balanced words.

The *timed* semantics of a balanced regular expression is then the union of the timed semantics of each balanced word in its (word) semantics:

$$\|E\| = \bigcup \left\{\|w\| \mid w \in |E|\right\}$$

This definition already points out the difficulty of constructing regular expressions over $\Sigma_{\underline{\mathbf{t}}} \cup \Pi$ that correspond to timed automata: we first need to build classical semantics of regular expressions in order to check whether all the words in this semantics have balanced parentheses. Only after this validation we may construct the timed semantics of the given expression.

Thanks to the particular form of timed automata provided by Theorem 2.1, we may prove the following form of the Kleene theorem:

Theorem 3.4 (Kleene theorem for timed automata) *The class of timed languages accepted by timed automata equals the class of timed languages which are the timed semantics of some balanced regular expression.*

Proof. We transform each timed automaton in the special form provided by Theorem 2.1 into a finite automaton whose transitions are labeled with *words*

over $\Sigma_{\underline{\mathbf{t}}} \cup \Pi$ (such automata are called *extended*, e.g., in [16]). The rough idea is very simple: each reset for clock x_i is transformed into the parenthesis $\langle^i$, while each clock check $x_i \in I$ is transformed into the parenthesis $\rangle_I^i$. We apply next the classical Kleene theorem to convert this automaton into a regular expression over $\Sigma_{\underline{\mathbf{t}}} \cup \Pi$. This expression is balanced since the automaton is in the special form of Theorem 2.1.

The reverse proof follows by mirroring the above pattern, that is, by transforming each parenthesis $\langle^i$ into a clock reset, and each parenthesis $\rangle_I^i$ into a clock check. However, the exact identity of the clock which is reset when encountering $\langle^i$ depends on the "nesting" of the parentheses of color i – that is, for each parenthesis $\langle^i$ we need to create several clocks, as many as the *maximal nesting degree* of the parentheses of color i. The finiteness of this maximal nesting degree is a consequence of Proposition A.1 from the appendix. The proof of this implication will be given in the full version of our paper.

□

4 Checking regular expressions for balance

In this section we give an algorithm for deciding whether an expression E over $\Sigma_{\underline{\mathbf{t}}} \cup \Pi$ is balanced. The idea is to associate special attributes to each sub-expression of E. These attributes represent the number of left and right parentheses of each color which are not balanced. The algorithm computes recursively these attributes for all the sub-expressions and decides that E is balanced if and only if all its attributes are zero.

In this section we use the "nonnegative" subtraction ("monus"),

$$a \dot{-} b = \begin{cases} a - b & \text{iff } a - b \geq 0 \\ 0 & \text{iff } a - b < 0 \end{cases}$$

We extend this operation and the addition to sets of naturals, in the straightforward manner: given two sets of natural numbers $A, B \subseteq \mathbb{N}$, we put $A+B = \{a + b \mid a \in A, b \in B\}$ and $A \dot{-} B = \{a \dot{-} b \mid a \in A, b \in B\}$.

Let us do first the following exercise:

Proposition 4.1 *Consider the binary operation $\ominus$ on $\mathbb{N} \times \mathbb{N}$, defined by*

$$(a, b) \ominus (c, d) = \big(a \dot{-} d + c, d \dot{-} a + b\big)$$

Then $\big(\mathbb{N} \times \mathbb{N}, \ominus, (0, 0)\big)$ is a monoid.

Furthermore, the mapping $\varphi : \mathbb{N} \times \mathbb{N} \to \mathbb{Z}$ defined by $\varphi(a, b) = a - b$, is a monoid morphism to the target monoid $(\mathbb{Z}, +, 0)$.

We extend all these operations onto sets of *n-tuples* of naturals (resp. integers). As a corollary of Proposition 4.1, $\big(\mathbb{N}^n \times \mathbb{N}^n, \ominus, (\mathbf{0}_n, \mathbf{0}_n)\big)$ is also a monoid and the mapping $\varphi_n : \mathbb{N}^n \times \mathbb{N}^n \to \mathbb{Z}^n$ defined by $\varphi_n(\overline{a}, \overline{b}) = \overline{b} - \overline{a}$ is a monoid morphism.

We also use the "unit" vectors $e_i = (e_i^1, \ldots, e_i^n) \in \mathbb{N}^n$, given by

$$e_i^j = \begin{cases} 1 & \text{for } j = i \\ 0 & \text{for } j \neq i \end{cases}$$

And finally, for any set $A \subseteq \mathbb{N}^n \times \mathbb{N}^n$ let

$$A^{k\ominus} = \underbrace{A \ominus \ldots \ominus A}_{k \text{ times}}$$

We proceed now to our construction of attributes: we first associate, to each word $w \in (\Sigma_{\underline{\mathbf{t}}} \cup \Pi)^*$, two naturals $l(w)$ and $r(w)$ which, intuitively, give the number of left, resp. right parentheses which are not balanced in w. The formal definition proceeds by induction on the length of the word:

$$\begin{aligned} l(\varepsilon) = r(\varepsilon) &= \mathbf{0}_n, & l(w\xi) &= l(w), \ \forall \xi \in \Sigma_{\underline{\mathbf{t}}} \\ l(w\langle^i) &= l(w) + e_i, & r(w\langle^i) &= r(w) \dot{-} e_i \\ l(w\rangle_I^i) &= l(w) \dot{-} e_i, & r(w\rangle_I^i) &= r(w) + e_i \end{aligned}$$

Together, l and r define a mapping from $(\Sigma_{\underline{\mathbf{t}}} \cup \Pi)^*$ to $\mathbb{N} \times \mathbb{N}$, denoted in the sequel by lr:

$$\mathsf{lr}(w) = \big(l(w), r(w)\big)$$

Proposition 4.2 *$\mathsf{lr} : (\Sigma_{\underline{\mathbf{t}}} \cup \Pi)^* \to \mathbb{N} \times \mathbb{N}$ is a monoid morphism, i.e. $\mathsf{lr}(w_1 w_2) = \mathsf{lr}(w_1) \ominus \mathsf{lr}(w_2)$.*

Remark 4.3 *w is a balanced word iff $l(w) = r(w) = 0$.* Or, in other words, the set of balanced words is the *kernel* of the morphism lr, that is, $\mathsf{lr}^{-1}(0,0)$.

The next step is to extend lr to *regular expressions*, by the rule

$$\mathsf{lr}_e(E) = \bigcup \big\{ \mathsf{lr}(w) \mid w \in |E| \big\}$$

Proposition 4.4 *E is a balanced timed regular expression if and only if*

$$\mathsf{lr}_e(E) = \big\{ (\mathbf{0}_n, \mathbf{0}_n) \big\}.$$

Our first aim is to show that lr_e can be computed *by structural induction* on the expression E. The following property paves the way for this approach:

Proposition 4.5 *For any regular expressions E, E_1, E_2 over $\Sigma_{\underline{\mathbf{t}}} \cup \Pi$,*

$$\begin{aligned} \mathsf{lr}_e(E_1 + E_2) &= \mathsf{lr}_e(E_1) \cup \mathsf{lr}_e(E_2) \\ \mathsf{lr}_e(E_1 \cdot E_2) &= \mathsf{lr}_e(E_1) \ominus \mathsf{lr}_e(E_2) \\ \mathsf{lr}_e(E^*) &= \bigcup_{k \in \mathbb{N}} \mathsf{lr}_e(E)^{k\ominus} \end{aligned}$$

For example, for $P_1 = \{`\langle`, `\rangle_I` \mid I \subseteq [0,\infty[\}$ and $P_2 = \{`\lfloor`, `\rfloor_I` \mid I \subseteq [0,\infty[\}$,

$$\mathsf{lr}_e\Big(\big(\ \underline{\mathtt{t}}a\lfloor b\ \big)^*\Big) = \left\{ \left(\begin{pmatrix}0\\k\end{pmatrix}, \begin{pmatrix}0\\0\end{pmatrix}\right) \,\middle|\, k \in \mathbb{N} \right\} \tag{5}$$

$$\mathsf{lr}_e\Big(\big(\ \lfloor\underline{\mathtt{t}}a\rangle_1\langle\underline{\mathtt{t}}a\rfloor_1 + \rfloor_2\underline{\mathtt{t}}a\lfloor\ \big)^*\Big) =$$

$$= \left\{ \left(\begin{pmatrix}1\\0\end{pmatrix},\begin{pmatrix}1\\0\end{pmatrix}\right), \left(\begin{pmatrix}0\\1\end{pmatrix},\begin{pmatrix}0\\1\end{pmatrix}\right), \left(\begin{pmatrix}1\\1\end{pmatrix},\begin{pmatrix}1\\1\end{pmatrix}\right) \right\} \tag{6}$$

The result from Proposition 4.5 is not sufficient for deciding whether an expression E is balanced. The reason is that star might generate *infinite* sets of tuples of naturals – as we can see in example (5) above.

The first idea to bypass this problem is the following: whenever $A \subseteq \mathbb{N}\times\mathbb{N}$ is an infinite set, for any set $B \subseteq \mathbb{N} \times \mathbb{N}$, $A \ominus B$ will be an infinite set. This means that, in our inductive calculus on the structure of a regular expression, whenever we would obtain an infinite lr_e, we should halt the computation and decide that the whole expression is non-balanced. Therefore, it only remains to find out when does $\mathsf{lr}_e(E^*)$ consist of an infinite set.

A further observation helps us here: note, from example (6) above, that, when in an expression E the difference between the left and the right parentheses of a certain kind is not zero, then $\mathsf{lr}_e(E^*)$ will be an infinite set. On the contrary, whenever $\mathsf{lr}_e(E)$ contains only tuples which give the same difference between left and right parentheses, $\mathsf{lr}_e(E^*)$ is finite.

The idea is then to consider, for each expression E, the set

$$\varphi_e(E) = \big\{ l(w) - r(w) \mid w \in |E| \big\} = \big\{ \overline{a} - \overline{b} \mid (a,b) \in \mathsf{lr}_e(E) \big\}$$

Then we may observe that if, for an expression E, $card(\varphi_e(E)) \geq 2$, that is, if E generates at least two words in which the difference between left and right parentheses is not the same, then for any expression E' which contains E as a subexpression, $\mathsf{lr}_e(E')$ would contain two or more elements. Therefore, E' cannot be balanced.

Recall that a *Kleene algebra* [8] is an algebra $\big(A, \cup, \cdot, *, \emptyset, \{e\}\big)$ that verifies all the equations valid in the structure $\big(\mathcal{P}(\Sigma^*), \cup, \cdot, *, \emptyset, \{\varepsilon\}\big)$. Furthermore, remind that each monoid $(M, e, \cdot)$ naturally generates a Kleene algebra $\big(\mathcal{P}(M), \cup, \cdot, *, \emptyset, \{\varepsilon\}\big)$, by putting, for each $S \subseteq M$,

$$S^* = \bigcup\nolimits_{n\in\mathbb{N}} \Big(\underbrace{S \cdot \ldots \cdot S}_{n \text{ times}} \Big)$$

Therefore, the monoid $\big(\mathbb{N}^n \times \mathbb{N}^n, \ominus, (\mathbf{0}_n, \mathbf{0}_n)\big)$ generates a Kleene algebra, in which the star is denoted $\circledast$:

$$A^{\circledast} = \{(\mathbf{0}_n, \mathbf{0}_n)\} \cup \bigcup_{k\in\mathbb{N}} A^{k\ominus}$$

Furthermore, the monoid $(\mathbb{Z}^n, +, \mathbf{0}_n)$ generates, on its turn, a Kleene algebra, in which

$$A^* = \{\mathbf{0}_n\} \cup \bigcup_{k\in\mathbb{N}} \Big(\underbrace{A + \ldots + A}_{k \text{ times}} \Big)$$

Proposition 4.6 (i) *Consider the monoid morphism $\varphi_n : \mathbb{N}^n \times \mathbb{N}^n \to \mathbb{Z}^n$ defined in Proposition 4.1. Then, for any $A \subseteq \mathbb{Z}^n$, $card(\varphi_n^{-1}(A)) \geq card(A)$.*

(ii) *φ_n can be lifted to a Kleene algebra morphism – which we denote φ_n too. It is the morphism $\varphi_n : \mathcal{P}(\mathbb{N}^n \times \mathbb{N}^n) \to \mathcal{P}(\mathbb{Z})$ defined by $\varphi_n(A) = \big\{\varphi_n(\overline{a}) \mid \overline{a} \in A\big\}$.*

(iii) *For any two sets $A, B \subseteq \mathbb{Z}^n$,*

$$card(A + B) \geq card(A) + card(B) \text{ and } card(A^*) \geq card(A).$$

(iv) *For any finite set $A \subseteq \mathbb{N}^n \times \mathbb{N}^n$, $card(\varphi_n(A^{\circledast})) < \infty$ if and only if $\varphi_n(A) = \{0_n\}$, and in this case we have*

$$A^{\circledast} = \{\mathbf{0}_{2n}\} \cup \bigcup_{k \leq card(A)} A^{k\ominus}$$

As a corollary, $A^{\circledast}$ is finite.

Proof. We will only prove the last property. The left-to-right implication is straightforward, since $\varphi_n(A) \neq \{0_n\}$ implies that, for any $\overline{a} \in \varphi_n(A)$, $k\overline{a} = (ka_1, \ldots ka_n) \in \varphi_n(A^{\circledast})$. For the reverse implication, suppose $\varphi_n(A) = \{\mathbf{0}_n\}$. Hence, for any $(\overline{a}, \overline{a}') \in A$, $\overline{a} = \overline{a}'$.

Let us observe that, for any $\overline{a}, \overline{b} \in \mathbb{N}^n$,

$$\big(\overline{a}, \overline{a}\big) \ominus \big(\overline{b}, \overline{b}\big) = \big(\overline{b}, \overline{b}\big) \ominus \big(\overline{a}, \overline{a}\big) \tag{7}$$

$$\big(\overline{a}, \overline{a}\big) \ominus \big(\overline{a}, \overline{a}\big) = \big(\overline{a}, \overline{a}\big) \tag{8}$$

Then for any $\overline{c} \in A^{k\ominus}$, with $k > card(A)$, we have that

$$\overline{c} = \overline{c}_1 \ominus \overline{c}_2 \ominus \ldots \ominus \overline{c}_k. \tag{9}$$

But since A has less than k elements, two c_is must be equal – say, $c_i = c_j$, for $1 \leq i < j \leq k$. On the other hand, by identity (7) above, we may rearrange the decomposition of c from (9) such that c_i and c_j occur one next to the other. But then identity (8) assures us that $c_i \ominus c_j = c_i$, hence c is decomposed into $k - 1$ elements from A. The result then follows by induction on k. □

Proposition 4.7 *Suppose $card(\varphi_e(E)) \geq 2$ for some expression E over $\Sigma_{\underline{\mathbf{t}}} \cup \Pi$. If E' is a regular expression with colored parentheses which contains E as a subexpression, then E' cannot be balanced.*

Proof. We will actually prove that, for any regular expression with colored parentheses E' which contains E as a subexpression we have the inequality

$card(\varphi_e(E')) \geq 2$ and $card(\mathsf{lr}_e(E')) \geq 2$. The result will then follow by means of Remark 4.4.

The proof of the two claims runs by straightforward structural induction on E', using Proposition 4.6. The interesting cases are when $E' = E_1 \cdot E_2$ and when $E' = E_1^*$. □

By assembling results from Propositions 4.5 and 4.6, we may give the following effective variant of the mapping lr_e:

$$\begin{aligned}
\mathsf{eff}(\xi) &= \{(\mathbf{0}_n, \mathbf{0}_n)\} \text{ for all } \xi \in \Sigma_{\underline{\mathbf{t}}} \\
\mathsf{eff}(\langle^i) &= \{(e_i, \mathbf{0}_n)\} \\
\mathsf{eff}(\rangle_I^i) &= \{(\mathbf{0}_n, e_i)\} \\
\mathsf{eff}(E_1 + E_2) &= \begin{cases} \bot & \text{iff } \mathsf{eff}(E_1) = \bot \text{ or } \mathsf{eff}(E_2) = \bot \\ \mathsf{eff}(E_1) \cup \mathsf{eff}(E_2) & \text{otherwise} \end{cases} \\
\mathsf{eff}(E_1 \cdot E_2) &= \begin{cases} \bot & \text{iff } \mathsf{eff}(E_1) = \bot \text{ or } \mathsf{eff}(E_2) = \bot \\ \mathsf{eff}(E_1) \ominus \mathsf{eff}(E_2) & \text{otherwise} \end{cases} \\
\mathsf{eff}(E^*) &= \begin{cases} \bot & \text{iff } \mathsf{eff}(E) = \bot \\ \bot & \text{iff } \varphi(\mathsf{eff}(E)) \neq \{\mathbf{0}_n\} \\ \bigcup_{k \leq card(\mathsf{eff}(E))} \mathsf{eff}(E)^{k\ominus} & \text{otherwise} \end{cases}
\end{aligned}$$

Since the recursive definition above involves only finite sets, it can be used as an algorithm to compute $\mathsf{eff}(E)$. The relation between $\mathsf{eff}(E)$ and $\mathsf{lr}_e(E)$ is described in the following proposition:

Proposition 4.8 *For each regular expression E over $\Sigma_{\underline{\mathbf{t}}} \cup \Pi$,*

$$\mathsf{eff}(E) = \begin{cases} \bot & \textit{iff } card(\mathsf{lr}_e(E)) = \infty \\ \mathsf{lr}_e(E) & \textit{otherwise} \end{cases}$$

The main result of this section is now immediate:

Theorem 4.9 *A regular expression E over $\Sigma_{\underline{\mathbf{t}}} \cup \Pi$ is balanced iff $\mathsf{eff}(E) = \{(\mathbf{0}_n, \mathbf{0}_n)\}$.*

5 Conclusion

The main contribution of this paper is a new class of regular expressions capable to describe all timed regular languages. This formalism, unlike its predecessors, does not use strange operations nor explicit clocks in the expressions, at the price of having a two-stage non-compositional semantics. The problem of practical methods to specify timed behaviors, which could be based on existing or new formalisms, needs further investigation (see [15] for a preliminary discussion).

Another (more technical) contribution of this paper is theorem 2.1 based on a convexity-based "normalisation" of timed automata. We have found this transformation very useful. A similar transformation has been applied directly to expressions in [4].

References

[1] Alur, R. and D. Dill, *A theory of timed automata*, Theoretical Computer Science **126** (1994), pp. 183–235.

[2] Annichini, A., E. Asarin and A. Bouajjani, *Symbolic techniques for parametric reasoning about counter and clock systems*, in: *Proceedings of CAV'00*, LNCS **1855**, 2000, pp. 419–434.

[3] Asarin, E., P. Caspi and O. Maler, *A Kleene theorem for timed automata*, in: *Proceedings of LICS'97*, 1997, pp. 160–171.

[4] Asarin, E., P. Caspi and O. Maler, *Timed regular expressions*, Journal of the ACM **49** (2002), pp. 172–206.

[5] Bouyer, P. and A. Petit, *Decomposition and composition of timed automata*, in: *Proceedings of ICALP'99*, LNCS **1644**, 1999, pp. 210–219.

[6] Bouyer, P. and A. Petit, *A Kleene/Büchi-like theorem for clock languages*, Journal of Automata, Languages and Combinatorics (2002), to appear.

[7] Bouyer, P., A. Petit and D. Thérien, *An algebraic characterization of data and timed languages*, in: *Proceedings of CONCUR'2001*, LNCS **2154**, 2001, pp. 248–261.

[8] Conway, J. H., "Regular Algebra and Finite Machines," Chapman and Hall, 1971.

[9] Daws, C., "Méthodes d'analyse de systèmes temporisés: de la théorie à la pratique," Ph.D. thesis, Institut National Polytechnique de Grenoble, France (1998).

[10] Daws, C. and S. Yovine, *Reducing the number of clock variables of timed automata*, in: *Proceedings of RTSS'96* (1996).

[11] Dima, C., "An algebraic theory of real-time formal languages," Ph.D. thesis, Université Joseph Fourier, Grenoble, France (2001).

[12] Dima, C., *Timed regular expressions with colored parentheses* (2002), submitted.

[13] Herrmann, P., *Renaming is necessary in timed regular expressions*, in: *Proceedings of FST&TCS'99*, LNCS **1738**, 1999, pp. 47–59.

[14] Hopcroft, J. and J. Ullman, "Introduction to Automata Theory, Languages and Computation," Addison-Wesley/Narosa Publishing House, 1992.

[15] Moy, M., "Spécifications des comportements temporisés," Master's thesis, Institut National Polytechnique de Grenoble (2002).

[16] Sheng Yu, "Regular Languages," Handbook of Formal Languages **1**, Springer Verlag, 1997.

A Appendix. Finiteness of the maximal nesting degree for balanced regular expressions

We will show that the semantics of each balanced regular expression is composed of words in which, at each "decomposition point", the number of "unbalanced" parentheses has an upper bound which does not depend upon w.

Proposition A.1 *For each balanced regular expression E there exist two n-tuples of naturals $\overline{M}_l, \overline{M}_r \in \mathbb{N}^n$ such that for all words $w \in |E|$, and for any decomposition $w = w_1 \cdot w_2$ we have*

$$l(w_1) \leq \overline{M}_l \text{ and } r(w_2) \leq \overline{M}_r$$

Here, the order is the usual extension of $\leq$ from the naturals to tuples of naturals, i.e., for all $\overline{a}, \overline{\in}\mathbb{N}^n$, $\overline{a} \leq \overline{b}$ iff $a_i \leq b_i$ for all $1 \leq i \leq n$.

Proof. We will actually prove that for each regular expression with the following property:

$$\mathsf{Ir}_e(E) \subseteq \left\{(\overline{a}, \overline{a}) \mid \overline{a} \in \mathbb{N}^n\right\} \tag{A.1}$$

we may get two n-tuples $\overline{M}_l, \overline{M}_r$ with the properties from the statement of this proposition. The proof of this fact runs by structural induction on E.

Before starting this proof by structural induction, let us observe that we also need that Property A.1 be preserved by structural induction, that is, whenever E, E_1, E_2 satisfy A.1, then so do $E_1 + E_2$, $E_1 \cdot E_2$ and E^*.

But this property is a corollary of the following observation: for each $\overline{a}, \overline{b} \in \mathbb{N}^n$, $(\overline{a}, \overline{a}) \ominus (\overline{b}, \overline{b}) = (\overline{c}, \overline{c})$, where $\overline{c}_i = \overline{a}_i$ iff $\overline{a}_i \geq \overline{b}_i$ and $\overline{c}_i = \overline{b}_i$ otherwise.

We may then do our structural induction as follows: observe first that the basic cases and the case $E = E_1 + E_2$ are trivial. Consider then the case $E = E_1 \cdot E_2$, and suppose we have proved the property for E_1 and E_2. Hence, by hypothesis we have four n-tuples of naturals $\overline{M}_l^1, \overline{M}_r^1, \overline{M}_l^2, \overline{M}_r^2$ that assure the uniform bound for the number of "unbalanced" parentheses at each decomposition point in words from $|E_1|$, resp. $|E_2|$.

We will prove that

$$\overline{M}_l = \overline{M}_l^1 + \overline{M}_l^2, \text{ resp. } \overline{M}_r = \overline{M}_r^1 + \overline{M}_r^2$$

are uniform bounds for words in $|E|$:

Take some $w \in E_1 \cdot E_2$ and consider a decomposition of it, $w = w_1 \cdot w_2$. Two symmetric situations occur then:

(i) $w_1 = w_1' \cdot w_1''$ with $w_1' \in |E_1|$ and $w_1'' \cdot w_2 \in |E_2|$, or
(ii) $w_2 = w_2' \cdot w_2''$ with $w_2' \in |E_2|$ and $w_1 \cdot w_2'' \in |E_1|$.

Therefore, we will only prove the result for the first situation: a first straightforward conclusion is that $r(w_2) \leq \overline{M}_r^2 \leq \overline{M}$, by the hypothesis on $\overline{M}_2^r$. What for $l(w_1)$, we have the following sequence of inequalities:

$$l(w_1) = l(w_1') \dot{-} r(w_1'') + l(w_1'') \leq l(w_1') + l(w_1'') \leq \overline{M}_l^1 + \overline{M}_l^2$$

again by the hypotheses on $\overline{M}_l^1$ and $\overline{M}_l^2$.

Consider now the case $E = E_1^*$, and suppose we have proved the result for E_1. Therefore, by hypothesis we have two n-tuples $\overline{M}_l^1, \overline{M}_r^1$ which bind the number of unbalanced parentheses at each decomposition point of a word in $|E_1|$. We will prove that

$$\overline{M}_l = \max\left\{l(w) \mid w \in |E_1|\right\} + \overline{M}_l^1 \text{ resp. } \overline{M}_r = \max\left\{r(w) \mid w \in |E_1|\right\} + \overline{M}_r^1$$

are uniform bounds for words in $|E|$:

Take some $w \in |E|$, that is, $w = w_1 \cdot \ldots w_k$ with $w_j \in |E_1|$, for all $1 \leq j \leq k$. Let us first observe that property (A.1) implies that for each $1 \leq j \leq k$,

$$l(w_1 \cdot \ldots \cdot w_j) = \max\{l(w_i) \mid 1 \leq i \leq j\}. \tag{A.2}$$

and similarly, $r(w_1 \cdot \ldots \cdot w_j) = \max\{r(w_i) \mid 1 \leq i \leq j\}$.

Consider now a decomposition $w = w' \cdot w''$ of w. Suppose the "decomposition point" falls inside the word w_j, for some $1 \leq j \leq k$. That is,

$$w' = w_1 \cdot \ldots \cdot w_{j-1} \cdot \omega, \text{ and } w'' = \omega' \cdot w_{j+1} \cdot \ldots \cdot w_k$$

where $\omega \cdot \omega' = w_j$.

But then, by identity (A.2), we have

$$l(w') = l(w_1 \cdot \ldots \cdot w_{j-1}) \dot{-} r(\omega) + l(\omega) \leq \max\{l(w_i) \mid 1 \leq i \leq j-1\} + \overline{M}_l^1$$

and similarly for $r(w'')$. □

An Integrated Approach for the Specification and Analysis of Stochastic Real-Time Systems

Mario Bravetti [1]

Dipartimento di Scienze dell'Informazione,
University of Bologna, Mura Anteo Zamboni 7, 40127 Bologna, Italy

Abstract

A formal approach for the specification and analysis of concurrent systems is proposed which integrates two different orthogonal aspects of time: (*i*) real-time, concerning the expression of time constraints and the verification of exact time properties, and (*ii*) probabilistic-time, concerning the probabilistic quantification of durations of system activities via exponential probability distributions and the evaluation of system performance. We show that these two aspects, that led to different specification paradigms called timed automata and Markovian process algebras, respectively, can be expressed in an integrated way by a single language: a process algebra capable of expressing activities with generally distributed durations. In particular, we consider the calculus of Interactive Generalized Semi-Markov Processes (IGSMPs) and we present formal techniques for compositionally deriving, from an IGSMP specification, (*i*) a pure real-time model (called Interactive Timed Automaton), by considering the support of general distributions, and (*ii*) a pure probabilistic-time model (called Interactive Weighted Markov Chain), by approximating general distributions with phase-type distributions.

1 Introduction

The importance of considering the behavior of concurrent systems with respect to time during their design process has been widely recognized (see e.g. [15,3,8,2,18,19]). In particular two approaches for expressing and analyzing time properties of systems have been developed which are based on formal description paradigms: (*i*) the real-time approach (see e.g. [2,18,19]), mainly concerned with the expression of time constraints and the verification of exact time properties, and (*ii*) the probabilistic-time approach (see e.g. [15,3,13]), mainly concerned with the probabilistic quantification of durations of system

[1] Email: bravetti@cs.unibo.it

This is a preliminary version and considered for the final proceedings, to be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

activities via exponential probability distributions and the evaluation of system performance.

The different aspects of time expressed by the Stochastic Time and Real-Time approaches can be seen as being *orthogonal*. According to the probabilistic-time approach the possible values for the duration of an activity are quantified through probabilistic (exponential) distributions, but no time constraint is expressible: all duration values are possible with probability greater than zero. According to the real-time approach some interval of time is definable for doing something, but the actual time the system spends in-between interval bounds is expressed non-deterministically. A specification paradigm capable of expressing both aspects of time should be able of expressing both time constraints and a probabilistic quantification for the possible durations which satisfy such constraints. We can obtain such an expressive power by considering stochastic models capable of expressing *general probability distributions* for the duration of activities. In this way time constraints are expressible via probability distribution functions that associate probability greater than zero only to time values that are possible according to the constraints. Technically, the set of possible time values for the duration of an activity is given by the *support* of the associated duration distribution. This idea of deriving real-time constraints from distribution supports, that we have introduced in [6], was subsequently applied also in [9] and [11].

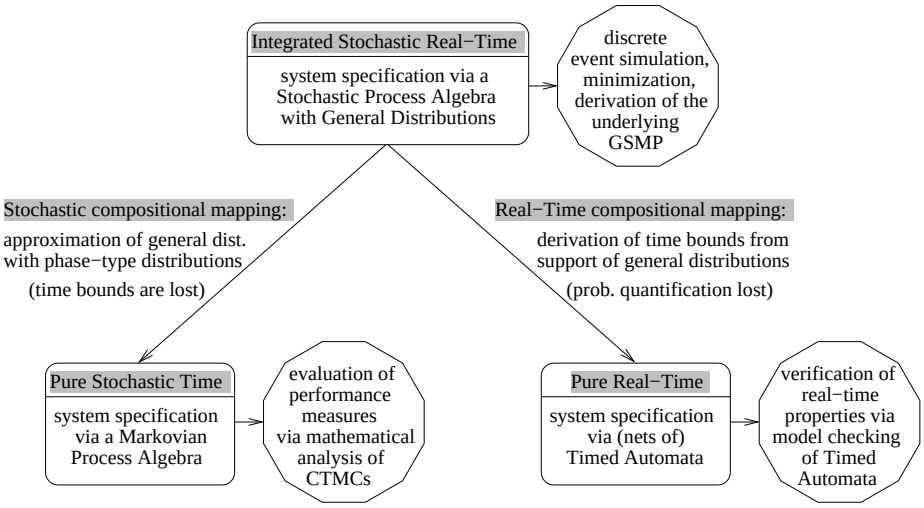

Fig. 1. Stochastic Real-Time Integrated Approach

Representing the real-time and probabilistic-time in a single specification paradigm allows us to model a concurrent system more precisely by expressing and analyzing the relationships between the two aspects of time. Moreover, the capability of expressing general distributions gives the possibility of producing much more realistic specifications of systems. System activities which have an uncertain duration could be represented probabilistically by more adequate distributions than exponential ones (e.g. Gaussian distributions or experimentally determined distributions).

In Fig. 1 we show how process algebra with generally distributed time can offer the possibility of an integrated approach for the modeling and analysis of Stochastic Real-Time concurrent/distributed systems. In particular we consider the calculus of Interactive Generalized Semi-Markov Processes introduced in [8,7,5]. IGSMP specifications can be directly analyzed through standard discrete event simulation (see e.g. [12]) and by means of the techniques introduced in [8]: minimization via a notion of bisimulation based congruence which abstracts from internal system activities, and derivation of the underlying performance model in the form of a GSMP for IGSMPs which are complete both from the interactive and from the performance viewpoints.

Besides the possibility of performing combined analysis, here we introduce formal techniques for compositionally deriving, from an IGSMP specification:

(i) A pure stochastic time (Markovian) specification in the form of a term of the calculus of *Interactive Weighted Markov Chains* (IWMCs) – basically an extension of Interactive Markov Chains (IMCs) of [13] with the capability of representing probabilistic choices through transitions labeled with *weights* [21] – by approximating general distributions with combinations of exponential distributions (the so called *phase-type* distributions). A consequence of this transformation is that all duration values for delays get probability greater than 0. Hence the information about time constraints (related to the real-time behavior of the system) is lost.

(ii) A pure real-time specification in the form of a net (a parallel composition) of *Interactive Timed Automata* (ITA) – a variant of Timed Automata [2,19] where action executions, events enabled on the basis of clock constraints and clock reset events are expressed by means of separate transitions – by considering the support of general distributions, i.e. the set of time values that are given probability (density) greater than 0, and by turning probabilistic choices into non-deterministic choices. As a consequence the information related to the probabilistic-time behavior of the system is lost.

Deriving a pure Markovian representation (the IWMC) and a pure real-time representation (the ITA) is very important from a practical viewpoint in that gives the opportunity of reusing existing techniques and tools alredy developed for performance evaluation and model-checking of non-probabilistic real-time properties. Moreover, the advantage of deriving an IWMC and an ITA from the same initial IGSMP specification (w.r.t. generating them independently) is that they are guaranteed to be consistent, in that they represent different aspects of the same initial system specification.

The technique leading to the derivation of the IWMC is particularly significant in that: (*i*) it shows process algebra to provide exactly the machinery necessary for approximating GSMPs with CTMCs through phase-type distributions, and (*ii*) it confirms ST semantics to be the adequate semantics for generally distributed time (as claimed e.g. in [8,5]) in that approxima-

tion of activity durations with phase-type distributions is a form of action refinement. From the practical viewpoint the approximation of general distributions with phase-type distributions will cause an approximation on the obtained performance measures. In particular the measures obtained will tend to the exact measures as the approximating phase-type durations tend to the exact duration distributions (by increasing the number of phases considered in the approximating phase-types). The problem of evaluating the error introduced in the measures depending on the level of approximation is a very difficult and known problem of statistics (see e.g. [4]) whose solution is somehow orthogonal to the results presented in this paper. Moreover note that, the better the approximation is, the greater the state space explosion caused by phase-type expansion is. Obviously this may become a problem if we want to reach certain levels of precision. Again solutions of this well-known problem are somehow orthogonal to the contents of this paper, e.g. we could adopt the technique introduced in [20] where the state-space is represented with Kronecker matrix expressions. On the other hand in spite of its inconveniences, for most systems with general distributions, approximation with phase-type is the only practical way to do performance analysis not based on simulation.

As far as the mapping into ITA is concerned, it just turns probability distributions into set of possible values for clocks by using distribution supports (defined by adopting the technical shrewdness introduced in [11]) without modifying the "structure" of the transition system. Therefore it has the desirable property of not increasing the number of states of the IGSMP when translating it into an ITA. Such a simple technique, which cannot be correctly applied to the Stochastic Automata model of [10] (see [11]), is convenient w.r.t. the more complex one introduced in [11] in that it avoids a blow up in the number of states which is exponential in the number of clocks used in the initial specification (see Sect. 4.2 for the details). As discussed in more details in [11], the mappings based on supports like this one guarantee that each behavior of the IGSMP which was executable with probability greater than zero becomes a possible behavior of the resulting ITA, but in general the converse cannot be stated. Hence at least non-probabilistic real-time safety properties of the IGSMP can be checked in the resulting ITA. As far as liveness properties are concerned only some kind of them (e.g. those related to possible action behaviors and not to particular time values) can be shown to hold in the initial IGSMP.

Unfortunately, in order not to make presentation too long, we do not include in this paper the definition of IGSMPs and their semantics, which can be found in [7,5]. The same holds for the calculus of IGSMPs, which is simply a variant of the calculus of IMC [13] where prefixes $<f, w>$ (representing delays whose duration has general distribution f and are chosen according to weight w) are used instead of λ prefixes (representing exponentially timed delays of rate λ), and its semantics (which maps algebraic terms into IGSMPs) which are defined in [8,5].

The paper is organized as follows. In Sect. 2 we introduce the calculus of IWMCs, which constitutes the first extension of IMC [13] with probabilistic choices endowed with a complete axiomatization for weak bisimultion. Then, in Sect. 3 we introduce ITA, which consitute the first variant of timed automata [2,19] endowed with a weak version of (structural) bisimulation equivalence and a compositional semantics. Finally, in Sect. 4 we present the two formal mappings from IGSMP specifications to IWMC and ITA specifications and we show that: (*i*) the IGSMP - IWMC mapping preserves performance measures once we replace generally distributed durations with the approximating phase-type durations in the initial IGSMP, (*ii*) the IGSMP - ITA mapping is such that the traces of "supported" behaviors (originating from time values in the support of distributions) starting in a state of the IGSMP are the same as the traces of possible behaviors starting in the corresponding state of the ITA (as in [11]), and (*iii*) both mappings are compositional and preserve weak bisimulation equivalence. In Appendix A we show an axiomatization for weak bisimulation which is complete over finite state IWMC terms, while in Appendix B we present the semantics of ITA and we show that it is compositional and preserves equivalence.

Proofs of theorems can be found in [5] Chapters 4,5 and 8.

2 Interactive Weighted Markov Chains

Interactive Weighted Markov Chains are an extension of Continuous Time Markov Chains with *action transitions*, representing the ability of the process to interact with other processes, and *probabilistic transitions*, representing probabilistic choices internally performed by the process. In particular Interactive Weighted Markov Chains basically extend Interactive Markov Chains of [13] with the capability of representing probabilistic choices through probabilistic transitions labeled with *weights* [21].

Extending IMCs in this way is very convenient in that it significantly simplifies the task of modeling real systems (in that alternative system behaviors can be expressed via probabilistic choices) without increasing the "complexity" of the underlying class of stochastic processes. This because probabilistic choices just give rise to vanishing states which can be eliminated via a simple procedure (see [3] Chapter 4) when evaluating performance.

Similarly to [13], in IWMCs the interrelation between standard action transitions and performance related transitions (probabilistic and exponentially timed transitions) is governed by the so-called *maximal progress assumption* [18]: the possibility of executing τ transitions prevents the execution performance related transitions, thus expressing that the system cannot wait if it has something internal to do. But differently from [13], where such a priority is captured in the definition of equivalence among IMCs, we prefer to express priority by cutting transitions which cannot be performed when defining and composing IWMCs (a solution also hinted in [14]). This allows

us to obtain smaller system models and to define a notion of bisimulation among IWMCs more simply, without having to discard any transitions when establishing equivalence.

As for IMCs [13], we will compose in parallel several IWMCs via a CSP-like synchronization policy. Alternative τ transitions in an IWMC represent internal non-deterministic choices which are performed in zero time and can never be "resolved" through synchronization with other system components. On the contrary, visible actions a in an IWMC are seen as incomplete actions which wait for a synchronization with other system components (they represent potential interaction with the environment). Therefore the choice of such actions in any IWMC state is governed by an external form of non-determinism, as their execution completely depends on the environment. We will also make use of an hiding operator which turns (incomplete) visible action transitions of an IWMC into (complete) τ transitions.

2.1 Definition of Interactive Weighted Markov Chain

In an IWMC we have four different kinds of state:

- *silent states*, enabling invisible action transitions τ and (possibly) visible action transitions a only. In such states the IWMC just performs a non-deterministic choice among the τ transitions in zero time and may potentially interact with the environment through one of the visible actions.
- *probabilistic states*, enabling probabilistic transitions and (possibly) visible action transitions a only. In such states (also called vanishing states) the IWMC just performs a probabilistic choice among the probabilistic transitions in zero time (proportionally to the weights labeling the transitions) and may potentially interact with the environment through one of the visible actions.
- *timed states*, enabling exponentially timed transitions and (possibly) visible action transitions a only. The IWMC sojourns in these states (also called tangible states) until one of the exponential delays terminates and the corresponding transition is performed. While the IWMC sojourns in the state, it may (at any time) potentially interact with the environment through one of the outgoing visible action transitions.
- *waiting states*, enabling standard visible actions only or no transition at all. In such states the IWMC remains indefinitely. It may, at any time, potentially interact with the environment through one of the outgoing visible action transitions.

In the following we present the formal definition of Interactive Weighted Markovian Transition System (IWMTS), then we will define interactive weighted Markov chains as IWMTSs possessing an initial state. Formally, rates, belonging to $\mathbb{R}^+$, are ranged over by $\lambda, \lambda', \ldots$ while weights, belonging to $\mathbb{R}^+$, are ranged over by $w, w', \ldots$. We use $\theta, \theta', \ldots$ to range over both rates and weights.

Moreover, we denote the set of standard action types used in a IWMTS by Act, ranged over by $\alpha, \alpha', \ldots$. As usual Act includes the special type τ denoting internal actions. The set $Act - \{\tau\}$ is ranged over by $a, b, \ldots$. The set of states of an IWMTS is denoted by Σ, ranged over by $s, s', \ldots$. We assume the following abbreviations that will make the definition of IWMTSs easier. Let us suppose that $T \subseteq (\Sigma \times Labels \times \Sigma)$ is a transition relation, where $Labels$ is a set of transition labels, ranged over by l. In the remainder we use $s \xrightarrow{l} s'$ to stand for $(s, l, s') \in T$, $s \xrightarrow{l}$ to stand for $\exists s' : s \xrightarrow{l} s'$, and $s \not\xrightarrow{l}$ to stand for $\nexists s' : s \xrightarrow{l} s'$.

Definition 2.1 An Interactive Weighted Markovian Transition System (IWMTS) is a tuple $\mathcal{M} = (\Sigma, Act, T_w, T_e, T_a)$ with

- Σ a set of states,
- Act a set of standard actions,
- $T_w \subseteq (\Sigma \times \mathbb{R}^+ \times \Sigma)$, $T_e \subseteq (\Sigma \times \mathbb{R}^+ \times \Sigma)$, and $T_a \subseteq (\Sigma \times Act \times \Sigma)$ three transition relations, containing probabilistic, exponentially timed and action transitions, respectively, such that: [2]

(i) $\forall s \in \Sigma.\ s \xrightarrow{\tau} \implies \nexists \theta.\, s \xrightarrow{\theta}$

(ii) $\forall s \in \Sigma.\ \exists w.\, s \xrightarrow{w} \implies \nexists \lambda.\, s \xrightarrow{\lambda}$

An Interactive Weighted Markov Chain (IWMC) is a tuple $\mathcal{M} = (\Sigma, Act, T_w, T_e, T_a, s_0)$, where $s_0 \in \Sigma$ is the initial state of the IWMC and $(\Sigma, Act, T_w, T_e, T_a)$ is an IWMTS. ■

The constraints over transition relations T_w, T_e and T_a guarantee that each state of the IWMC belongs to one of the four kind of states above. In particular, the first requirement says that if a state can perform internal τ actions then it cannot perform exponentially timed or probabilistic transitions. Such a property derives from the assumption of *maximal progress*: the possibility of performing internal actions prevents the execution of delays. The second requirement says that if a state can perform probabilistic transitions then it cannot perform exponentially timed transitions. Such a property derives from the assumption of *urgency of choices*: probabilistic choices cannot be delayed but must be performed immediately, hence they prevent the execution of exponentially timed delays.

2.2 The Calculus of IWMCs

Let Var be a set of process variables ranged over by X, Y, Z. Let $ARFun = \{\varphi : Act \longrightarrow Act \mid \varphi(\tau) = \tau \wedge \varphi(Act - \{\tau\}) \subseteq Act - \{\tau\}\}$ be a set of *action relabeling functions*, ranged over by φ.

[2] For the sake of readability here and in the rest of the paper we assume the following operator precedence when writing constraints for transition relations: existential quantifier > "and" operator > implication.

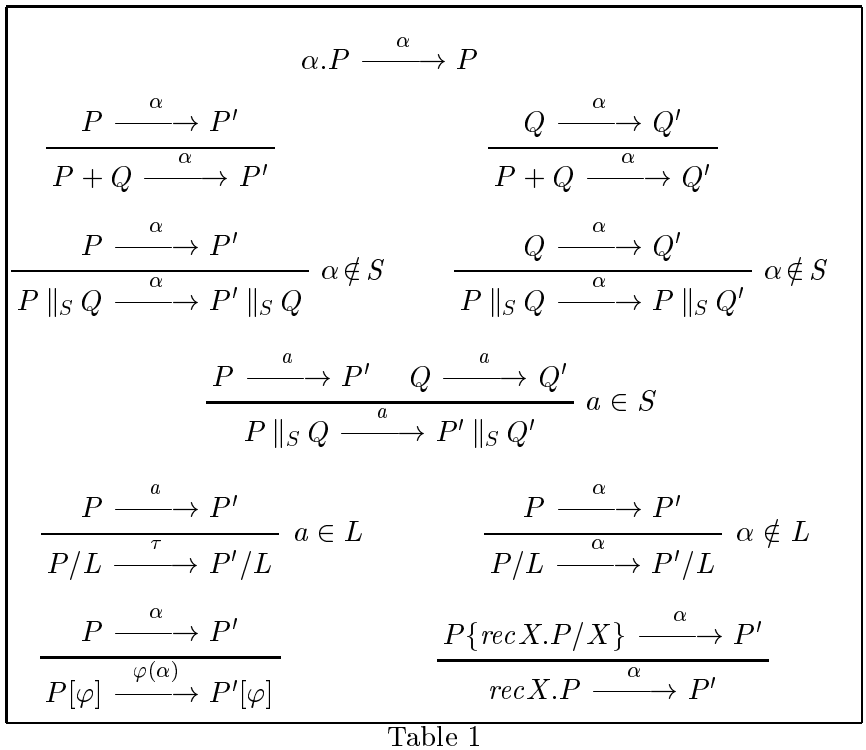

Table 1
Standard Rules

Definition 2.2 We define the language $IWMC$ as the set of terms generated by the following syntax

$$P ::= \underline{0} \mid X \mid w.P \mid \lambda.P \mid \alpha.P \mid P + P \mid P/L \mid P[\varphi] \mid P \parallel_S P \mid recX.P$$

where $L, S \subseteq Act - \{\tau\}$. An $IWMC$ process is a closed term of $IWMC$. We denote by $IWMC_g$ the set of strongly guarded terms of $IWMC$. [3] ■

"$\underline{0}$" denotes a process that cannot move. The operators "." and "+" are the CCS prefix and choice. "$/L$" is the hiding operator which turns into τ the actions in L, "$[\varphi]$" is the relabeling operator which relabels visible actions according to φ. "$\|_S$" is the CSP parallel operator, where synchronization over actions in S is required. Finally "$recX$" denotes recursion in the usual way.

The semantics of IWMC terms produces a transition system labeled by actions in Act, weights in $\mathbb{R}^+$ and rates in $\mathbb{R}^+$. We use $\gamma, \gamma', \ldots$ to range over transition labels. Such a transition system is defined as being the $IWMTS$ $\mathcal{M} = (IWMC_g, Act, T_w, T_e, T_a)$, where: T_a is the least subset of $IWMC_g \times Act \times IWMC_g$ satisfying the standard operational rules of Table 1, T_w is obtained from the least multiset over $IWMC_g \times \mathbb{R}^+ \times IWMC_g$ satisfying the operational rules of Table 2 (similarly to [15,13], we consider a transition to have arity m if and only if it can be derived in m possible ways from the operational rules) by summing the weights of the multiple occurrences of the same transition, and T_e is obtained from the least multiset over $IWMC_g \times \mathbb{R}^+ \times IWMC_g$ satisfying the operational rules of Table 3 by summing the rates of the multiple occurrences

[3] We consider w and λ prefixes as being guards in the definition of strong guardedness.

$$w.P \xrightarrow{w} P$$

$$\frac{P \xrightarrow{w} P' \wedge Q \xrightarrow{\tau} \not\rightarrow}{P + Q \xrightarrow{w} P'} \qquad \frac{Q \xrightarrow{w} Q' \wedge P \xrightarrow{\tau} \not\rightarrow}{P + Q \xrightarrow{w} Q'}$$

$$\frac{P \xrightarrow{w} P' \wedge Q \xrightarrow{\tau} \not\rightarrow}{P \parallel_S Q \xrightarrow{w} P' \parallel_S Q} \qquad \frac{Q \xrightarrow{w} Q' \wedge P \xrightarrow{\tau} \not\rightarrow}{P \parallel_S Q \xrightarrow{w} P \parallel_S Q'}$$

$$\frac{P \xrightarrow{w} P' \wedge \nexists a \in L.\, P \xrightarrow{a}}{P/L \xrightarrow{w} P'/L} \qquad \frac{P \xrightarrow{w} P'}{P[\varphi] \xrightarrow{w} P'[\varphi]}$$

$$\frac{P\{recX.P/X\} \xrightarrow{w} P'}{recX.P \xrightarrow{w} P'}$$

Table 2
Rules for Probabilistic Moves

$$\lambda.P \xrightarrow{\lambda} P$$

$$\frac{P \xrightarrow{\lambda} P' \wedge Q \xrightarrow{\tau} \not\rightarrow \wedge\, Q \xrightarrow{w} \not\rightarrow}{P + Q \xrightarrow{\lambda} P'} \qquad \frac{Q \xrightarrow{\lambda} Q' \wedge P \xrightarrow{\tau} \not\rightarrow \wedge\, P \xrightarrow{w} \not\rightarrow}{P + Q \xrightarrow{\lambda} Q'}$$

$$\frac{P \xrightarrow{\lambda} P' \wedge Q \xrightarrow{\tau} \not\rightarrow \wedge\, Q \xrightarrow{w} \not\rightarrow}{P \parallel_S Q \xrightarrow{\lambda} P' \parallel_S Q} \qquad \frac{Q \xrightarrow{\lambda} Q' \wedge P \xrightarrow{\tau} \not\rightarrow \wedge\, P \xrightarrow{w} \not\rightarrow}{P \parallel_S Q \xrightarrow{\lambda} P \parallel_S Q'}$$

$$\frac{P \xrightarrow{\lambda} P' \wedge \nexists a \in L.\, P \xrightarrow{a}}{P/L \xrightarrow{\lambda} P'/L} \qquad \frac{P \xrightarrow{\lambda} P'}{P[\varphi] \xrightarrow{\lambda} P'[\varphi]}$$

$$\frac{P\{recX.P/X\} \xrightarrow{\lambda} P'}{recX.P \xrightarrow{\lambda} P'}$$

Table 3
Rules for Exponentially Timed Moves

of the same transition. In Tables 2 and 3 we use $P \xrightarrow{a}$ to stand for $\exists P' : P \xrightarrow{a} P'$, $P \xrightarrow{\tau} \not\rightarrow$ to stand for $\nexists Q : P \xrightarrow{\tau} Q$ and $P \xrightarrow{w} \not\rightarrow$ to stand for $\nexists w, Q : P \xrightarrow{w} Q$.

The rules of Table 2 define probabilistic transitions, by taking into account the priority of "τ" actions over weights. Note that we consider a "global" kind

of weights which are applied also across the parallel operator. Moreover we can just interleave parallel weight transitions because they are executed in zero time.

Definition 2.3 The semantic model $\mathcal{M}[\![P]\!]$ of $P \in IWMC_g$ is the *IWMC* defined by $\mathcal{M}[\![P]\!] = (S_P, Act, T_{w,P}, T_{e,P}, T_{a,P}, P)$, where: S_P is the least subset of $IWMC_g$ such that $P \in S_P$ and, if $P' \in S_P$ and $P' \xrightarrow{\gamma} P''$, then $P'' \in S_P$; moreover $T_{w,P}, T_{e,P}$ and $T_{a,P}$ are the restriction of T_w, T_e and T_a to $S_P \times Act \times S_P$, $S_P \times \mathbb{R}^+ \times S_P$ and $S_P \times \mathbb{R}^+ \times S_P$.

2.3 Observational Congruence for IWMCs

Observational congruence over IWMCs deals with exponentially timed choices according to Markovian bisimulation [15], deals with probabilistic choices according to probabilistic bisimulation [16], and abstracts from standard τ actions as in [17].

In our context we express cumulative probabilities and cumulative exponential times by aggregating weights and rates, respectively. In particular, if I is a set of states, $TW(s, I)$ represents the cumulative weight of probabilistic transitions leaving s and going into a state of I. Similarly, $TR(s, I)$ represents the cumulative rate of exponentially timed transitions from s to I.

The definition of weak bisimilarity is an adaptation of that presented in [13] to our context.

Let $\stackrel{\alpha}{\Longrightarrow}$ denote $(\stackrel{\tau}{\longrightarrow})^* \stackrel{\alpha}{\longrightarrow} (\stackrel{\tau}{\longrightarrow})^*$, i.e. a sequence of transitions including a single α transition and any number of τ transitions. Moreover we define $\stackrel{\hat{\alpha}}{\Longrightarrow} = \stackrel{\alpha}{\Longrightarrow}$ if $\alpha \neq \tau$ and $\stackrel{\hat{\tau}}{\Longrightarrow} = (\stackrel{\tau}{\longrightarrow})^*$, i.e. a possibly empty sequence of τ transitions.

Definition 2.4 Let $\mathcal{M} = (\alpha, Act, T_w, T_e, T_a)$ be a IWMTS. An equivalence relation β on α is a *weak bisimulation* iff $s_1 \; \beta \; s_2$ implies

- for every $\alpha \in Act$ and $s_1' \in \Sigma$,
$$s_1 \stackrel{\alpha}{\longrightarrow} s_1' \text{ implies } s_2 \stackrel{\hat{\alpha}}{\Longrightarrow} s_2' \text{ for some } s_2' \text{ with } s_1' \; \beta \; s_2',$$
- $s_2 \stackrel{\hat{\tau}}{\Longrightarrow} s_2'$ for some s_2' such that, for every equivalence class I of β,
$$TW(s_1, I) = TW(s_2', I) \text{ and } TR(s_1, I) = TR(s_2', I)$$

Two states s_1 and s_2 are weakly bisimilar, denoted by $s_1 \approx_{IWMC} s_2$, iff (s_1, s_2) is included in some weak bisimulation. ■

Differently from [13], for the sake of simplicity, we do not express conditions about the stability of bisimilar states because we are interested in obtaining a congruence result only for strongly guarded processes of our calculus. Such processes cannot produce an IWMC which is forced in a τ loop, hence we do not have to recognize this situation.

Definition 2.5 Two closed terms P, Q of $IWMC_g$ are observational congruent, written $P \simeq_{IWMC} Q$, iff:

- for every $\alpha \in Act$ and $P' \in IWMC_g$,
  $P \xrightarrow{\alpha} P'$ implies $Q \stackrel{\alpha}{\Longrightarrow} Q'$ for some Q' with $P' \approx_{IWMC} Q'$,
- for every $\alpha \in Act$ and $Q' \in IWMC_g$,
  $Q \xrightarrow{\alpha} Q'$ implies $P \stackrel{\alpha}{\Longrightarrow} P'$ for some P' with $P' \approx_{IWMC} Q'$,
- for every equivalence class I of β,
  $TW(P, I) = TW(Q, I)$ and $TR(P, I) = TR(Q, I)$ ■

Theorem 2.6 *$\simeq_{IWMC}$ is a congruence over terms of $IWMC_g$ w.r.t. all the operators of IWMC, including recursion.*

It is easy to to produce an axiomatization for $\simeq_{IWMC}$ which is complete over finite-state $IWMC_g$ terms (due to lack of space we refer to Appendix A for the details).

3 Interactive Timed Automata

Interactive Timed Automata are a variant of classical Timed Automata [2,19], where action executions, events enabled on the basis of clock constraints and clock reset events are expressed by means of separate transitions. The advantage of ITA with respect to existing timed automata, where usually we have one single kind of transition expressing all these features in a combined fashion, is that action transitions can be dealt with separately from time-related transitions, hence making it easy to define, e.g., a notion of weak bisimulation as a simple extension of the standard notion of [17]. Therefore, with respect to the existing equivalence notions for timed automata, abstracting from τ transitions, improves the capability of minimizing the state space of specified systems. ITA can be straightforwardly mapped into existing timed automata (e.g. those defined in [19]), hence previous decidability results and software tools can be exploited for analysing real-time properties in ITA specifications.

Time delays are modeled in ITA by means of clocks C_n which are set to zero and count upwards while time passes. An ITA represents the behavior of a system component by employing both clock reset transitions and clock bound transitions, representing the timed behavior of the component and standard action transitions, representing the interactive behavior of the component. Clock reset transitions are labeled with a clock name C_n and represent the event of reset of the clock (which is set to zero). After such event, C_n counts upwards while time passes and states are traversed by the automaton. When several clock reset transitions are enabled in an ITA state, the choice among them is just non-deterministic. Clock bound transitions are labeled with a clock constraint ϕ (an expression built from bounds on the clock values) and they can be executed only when the status of clocks satisfies such a constraint. A system is allowed to stay in a state enabling several clock bound transitions

as long as *all* clock constraints labeling the transitions can be satisfied at present time or in the future. The role and the meaning of visible and invisible action transitions, related to composition of ITA via a CSP-like parallel composition and hiding, is exactly the same as for IWMCs.

3.1 Definition of Interactive Timed Automaton

In an ITA we have four different kinds of state:

- *silent states*, enabling invisible action transitions τ and (possibly) visible action transitions a only. The meaning of such states is exactly as in IWMCs.
- *reset states*, enabling reset transitions C_n and (possibly) visible action transitions a only. In such states the ITA just performs a choice among the clock reset transitions in zero time and may potentially interact with the environment through one of the visible actions.
- *timed states*, enabling clock bound transitions ϕ and (possibly) visible action transitions a only. In such states all the clocks of the ITA count upwards as time passes. The system is allowed to sojourn in the state as long as *all* clock constraints labeling its outgoing transitions can be satisfied at the present time or in the future. Moreover, it can non-deterministically leave the state at any time through a bound transition ϕ whose constraint ϕ is (at present time) satisfied. Moreover, while the ITA sojourns in the state, it may (at any time) potentially interact with the environment through one of the outgoing visible action transitions.
- *waiting states*, enabling standard visible actions only or no transition at all. In such states the ITA remains indefinitely. It may, at any time, potentially interact with the environment through one of the outgoing visible action transitions.

In the following we present the formal definition of Interactive Timed Automaton Transition System (ITATS), then we will define Interactive Timed Automata as ITATSs possessing an initial state. Formally, we use $T, T', \ldots$, representing sets of time values, to range over subsets of $\mathbb{R}^+ \cup \{0\}$. Moreover, we denote the set of standard action types used in an ITATS by Act, ranged over by $\alpha, \alpha', \ldots$. As usual Act includes the special type τ denoting internal actions. The set $Act - \{\tau\}$ is ranged over by $a, b, \ldots$. The set of clocks of an ITATS is denoted by $\mathcal{C} = \{C_n \mid n \in \mathcal{C}Names\}$, where $\mathcal{C}Names$ is a set of clock names. Given a set $\mathcal{C}$, we denote with $\mathcal{C}^\Phi$, ranged over by $\phi, \phi', \ldots$, the set of constraints over clocks of $\mathcal{C}$ (the labels of clock bound transitions), which is defined as the set of terms generated by the following syntax:

$$\phi ::= C_n \in T \mid \phi \wedge \phi$$

Moreover, let $\mathcal{C} \cup \mathcal{C}^\Phi$ be ranged over by $\theta, \theta', \ldots$. The set of states of an ITATS is denoted by Σ, ranged over by $s, s', \ldots$. We assume the following abbreviations that will make the definition of ITATSs easier.

Definition 3.1 An Interactive Timed Automata Transition System (ITATS) is a tuple $\mathcal{T} = (\Sigma, \mathcal{C}, Act, T_r, T_b, T_a)$ with

- Σ a set of states,
- $\mathcal{C}$ a set of clocks,
- Act a set of standard actions,
- $T_r \subseteq (\Sigma \times \mathcal{C} \times \Sigma)$, $T_b \subseteq (\Sigma \times \mathcal{C}^\Phi \times \Sigma)$, and $T_a \subseteq (\Sigma \times Act \times \Sigma)$ three transition relations representing clock reset and clock bound events and action execution, respectively, such that:

(i) $\forall s \in \Sigma.\quad s \xrightarrow{\tau} \implies \nexists \theta.\, s \xrightarrow{\theta}$

(ii) $\forall s \in \Sigma.\quad \exists C_n.\, s \xrightarrow{C_n} \implies \nexists \phi.\, s \xrightarrow{\phi}$

An Interactive Timed Automata (ITA) is a tuple $\mathcal{T} = (\Sigma, \mathcal{C}, Act, T_r, T_b, T_a, s_0)$, where $s_0 \in \Sigma$ is the initial state of the ITA and $(\Sigma, \mathcal{C}, Act, T_r, T_b, T_a)$ is an ITATS. ■

The constraints over transition relations T_r, T_b and T_a guarantee that each state of the ITA belongs to one of the four kind of states above. In particular, the first requirement says that if a state can perform internal τ actions then it cannot perform clock reset transitions or clock bound transitions. Such a property derives from the assumption of *maximal progress*: the possibility of performing internal actions prevents the execution of time-related activity. The second requirement says that if a state can perform clock reset transitions then it cannot perform clock bound transitions. Such a property derives from an assumption of *urgency of clock resets*: clock reset transitions cannot be delayed but must be performed immediately and they are just assumed to prevent the execution of clock bound transitions.

3.2 Composing ITA

In the following we present the formal definitions of parallel composition and hiding of ITA. It can be easily shown that the transition system obtained by the composition is still an ITA (see [5]) due to the fact that maximal progress and urgency of clock resets assumptions are enforced when composing ITA.

Given a clock renaming function $ren : \mathcal{C} \longrightarrow \mathcal{C}$, we assume $ren(\phi)$ to be the constraint ϕ' obtained from ϕ by renaming clocks in ϕ according to function ren. In particular we define the renaming function $l : \mathcal{C} \longrightarrow \mathcal{C}$ by $\{(C_n, C_{n,l}) \mid C_n \in \mathcal{C}\}$ and, similarly, function $r : \mathcal{C} \longrightarrow \mathcal{C}$ by $\{(C_n, C_{n,r}) \mid C_n \in \mathcal{C}\}$.

Definition 3.2 The parallel composition $\mathcal{T}_1 \|_S \mathcal{T}_2$ of two ITA $\mathcal{T}_1 = (\Sigma_1, \mathcal{C}_1, Act, T_{r,1}, T_{b,1}, T_{a,1}, s_{0,1})$ and $\mathcal{T}_2 = (\Sigma_2, \mathcal{C}_2, Act, T_{r,2}, T_{b,2}, T_{a,2}, s_{0,2})$, with $S \subset Act - \{\tau\}$ being the synchronization set, is the tuple $(\Sigma, \mathcal{C}, Act, T_r, T_b, T_a, (s_{0,1}, s_{0,2}))$ with

- $\Sigma = \Sigma_1 \times \Sigma_2 \times \mathcal{M}$ the set of states,
- $\mathcal{C} = \{C_{n,l} \mid C_n \in \mathcal{C}_1\} \cup \{C_{n,r} \mid C_n \in \mathcal{C}_2\}$

- $T_r \subseteq (\Sigma \times \mathcal{C} \times \Sigma)$, $T_b \subseteq (\Sigma \times \mathcal{C}^{\Phi} \times \Sigma)$, and $T_a \subseteq (\Sigma \times Act \times \Sigma)$ are the least transition relations, such that $\forall (s_1, s_2) \in \Sigma$.

  1l $s_1 \xrightarrow{\alpha} s_1',\ \alpha \notin S \implies (s_1, s_2) \xrightarrow{\alpha} (s_1', s_2)$

  2 $s_1 \xrightarrow{a} s_1' \wedge s_2 \xrightarrow{a} s_2',\ a \in S \implies (s_1, s_2) \xrightarrow{a} (s_1', s_2')$

  3l $s_1 \xrightarrow{C_n} s_1' \wedge s_2 \xrightarrow{\tau} \not\implies (s_1, s_2) \xrightarrow{C_{n,l}} (s_1', s_2)$

  4l $s_1 \xrightarrow{\phi} s_1' \wedge s_2 \xrightarrow{\tau} \not\implies \wedge\ \nexists C_n.\, s_2 \xrightarrow{C_n} \implies (s_1, s_2) \xrightarrow{l(\phi)} (s_1', s_2)$

  and also the symmetric rules $\mathbf{1_r}, \mathbf{3_r}, \mathbf{4_r}$ referring to the local transitions of $\mathcal{T}_2$, which are obtained from the rules $\mathbf{1_l}, \mathbf{3_l}, \mathbf{4_l}$ by exchanging the roles of states s_1 (s_1') and s_2 (s_2'), by turning l into r in the subscripts of clocks, and by turning the renaming function l into r, hold true.
- $(s_{0,1}, s_{0,2}) \in \Sigma$ the initial state ■

Each state $s \in \Sigma$ of the composed model is represented by a pair of states ($s_1 \in \Sigma_1$ and $s_2 \in \Sigma_2$). Moreover we rename clocks of both ITA $\mathcal{T}_1$ and $\mathcal{T}_2$ so to avoid a name conflict whenever two clocks with the same name C_n are simultaneously in execution in both ITA. Rules **1** (**2**) describe the behavior of the composed model in the case of a standard action α performed by one (or both, via a synchronization) ITA, when $\alpha \notin S$ ($\alpha \in S$). Rules **3** and **4** define the behavior of the composed model in the case of clock reset and clock bound transitions, respectively, locally performed by components. Note that the negative clauses in the premises enforce the maximal progress and the urgency of clock resets assumptions.

Definition 3.3 The hiding $\mathcal{T}/L$ of a ITA $\mathcal{T} = (\Sigma, \mathcal{C}, Act, T_{r,1}, T_{b,1}, T_{a,1}, s_0)$ with $L \subset Act - \{\tau\}$ being the set of visible actions to be hidden is the tuple $(\Sigma, \mathcal{C}, Act, T_r, T_b, T_a, s_0)$ where $T_r \subseteq (\Sigma \times \mathcal{C} \times \Sigma)$, $T_b \subseteq (\Sigma \times \mathcal{C}^{\Phi} \times \Sigma)$, and $T_a \subseteq (\Sigma \times Act \times \Sigma)$ are the least set of transitions, such that $\forall s \in \Sigma$. [4]

1 $s \xrightarrow{\alpha}_1 s',\ \alpha \notin L \implies s \xrightarrow{\alpha} s'$

2 $s \xrightarrow{a}_1 s',\ a \in L \implies s \xrightarrow{\tau} s'$

3 $s \xrightarrow{\theta}_1 s' \wedge\ \nexists a \in L.\, s \xrightarrow{a}_1 \implies s \xrightarrow{\theta} s'$ ■

Rules **1** and **2** are standard. Rule **3** says that the effect of the hiding operator over states of $\mathcal{T}$ which enable standard actions in L is to preempt all clock related transitions according to the maximal progress assumption.

3.3 Weak bisimulation for ITA

Now we will introduce a notion of weak bisimulation over ITA which matches the clock related transitions as in [1] and abstracts from standard τ actions similarly to [17].

[4] In order to distinguish transition of $T_{r,1}$, $T_{b,1}$ and $T_{a,1}$ from transitions of T_r, T_b and T_a we denote the former with "$\longrightarrow_1$" and the latter simply with "$\longrightarrow$".

Given an ITATS $\mathcal{T} = (\Sigma, \mathcal{C}, Act, T_r, T_b, T_a)$, weak bisimulation over states is defined by associating clock names as in [1] so that equivalence does not depend on the particular names used for clocks. We use H to range over association histories of clock names, i.e. partial bijections from $\mathcal{C}$ to $\mathcal{C}$. We denote by $\mathcal{H}$ the set of all association histories.

We now present weak bisimulation for ITA which is defined by means of a family of bisimulations β_H, each indexed by an association history. First of all, let us say that a $\mathcal{H}$-indexed family of binary relations $\{\beta_H \mid H \in \mathcal{H}\}$ over Σ is *symmetric* if and only if $(s_1, s_2) \in \beta_H$ implies $(s_2, s_1) \in \beta_{\overline{H}}$, where $\overline{H} = \{(C_{n'}, C_n) \mid (C_n, C_{n'}) \in H\}$. Moreover, we use $H \leftarrow (C_n, C_{n'})$ to denote the association history H' obtained from H by adding the pair $(C_n, C_{n'})$ and removing old associations $(C_n, C_{n''})$ and $(C_{n'''}, C_{n'})$, for some $C_{n''}$ and $C_{n'''}$, already contained in H, thus preserving the structure of bijection from $\mathcal{C}$ to $\mathcal{C}$. We use γ to range over transition labels, i.e. $Act \cup \mathcal{C} \cup \mathcal{C}^\Phi$.

Definition 3.4 Let $\mathcal{T} = (\Sigma, \mathcal{C}, Act, T_r, T_b, T_a)$ be a ITATS. A symmetric $\mathcal{H}$-indexed family $\mathbf{B} = \{\beta_H \subseteq \Sigma \times \Sigma \mid H \in \mathcal{H}\}$ of binary relations over Σ is a *weak bisimulation family* iff $s_1 \; \beta_H \; s_2$ implies

- for every $\alpha \in Act$ and $s_1' \in \Sigma$,
$$s_1 \xrightarrow{\alpha} s_1' \text{ implies } s_2 \stackrel{\hat{\alpha}}{\Longrightarrow} s_2' \text{ for some } s_2' \text{ with } s_1' \; \beta_H \; s_2'$$
- for every $C_n \in \mathcal{C}$ and $s_1' \in \Sigma$,
$$s_1 \xrightarrow{C_n} s_1' \text{ implies } s_2 \stackrel{C_{n'}}{\Longrightarrow} s_2' \text{ for some } s_2', C_{n'} \text{ with } s_1' \; \beta_{H \leftarrow (C_n, C_{n'})} \; s_2'$$
- for every $\phi \in \mathcal{C}^\Phi$ and $s_1' \in \Sigma$,
$$s_1 \xrightarrow{\phi} s_1' \text{ implies } \phi \in dom(H) \text{ and } s_2 \stackrel{H(\phi)}{\Longrightarrow} s_2' \text{ for some } s_2' \text{ with } s_1' \; \beta_H \; s_2'$$

Two states s_1 and s_2 are weakly bisimilar with respect to association history $H \in \mathcal{H}$, denoted by $s_1 \approx_{ITA,H} s_2$, iff there exist some weak bisimulation family $\mathbf{B} = \{\beta_H \mid H \in \mathcal{H}\}$ such that $(s_1, s_2) \in \beta_H$. Two ITA $(\mathcal{T}_1, s_{0,1})$ and $(\mathcal{T}_2, s_{0,2})$ are weakly bisimilar, denoted by $(\mathcal{T}_1, s_{0,1}) \approx_{ITA} (\mathcal{T}_2, s_{0,2})$ if their initial states $s_{0,1}$ and $s_{0,2}$ are such that $s_{0,1} \approx_{ITA,\emptyset} s_{0,2}$ in the ITATS obtained with the disjoint union of $\mathcal{T}_1$ and $\mathcal{T}_2$. ■

3.4 Semantics of ITA

ITA are endowed with a semantics which maps an ITA onto a transition system where: (i) the passage of time is explicitely represented by transitions labeled with numeric time delays $t \in \mathbb{R}^+ \cup 0$ and (ii) clock reset transitions and clock bound transitions are turned into prioritized transitions reflecting the precedence of clock reset transitions over clock bound transitions. Differently from existing approaches, we express semantic models of ITA by means of "interactive" timed transition systems which can be themselves composed and for which we define a notion of weak bisimulation. This allows us to develop a semantic mapping which is compositional with respect to parallel composition

and hiding and preserves equivalence, similarly to what is done in [7,5] for IGSMPs. Due to lack of space we refer the reader to Appendix B for a complete presentation of the semantics of ITA.

4 Mapping IGSMPs onto Pure Markovian and Real-Time Processes

In this section we present the two formal mappings from IGSMPs, representing the stochastic and real-time behavior of a concurrent system in an integrated way, into IWMCs, representing the pure stochastic (Markovian) behavior of the system, and into ITA, representing the pure real-time behavior of the system. The former mapping is obtained by approximating generally distributed durations with phase-type durations. Technically, such mapping is performed compositionally at the algebraic level by replacing each delay prefix $<f, w>$ occurring in an algebraic term of an IGSMP specification with an IWMC term $w.P$, where P is the algebraic representation of a phase-type distribution approximating f. In this way we map a term of the calculus of IGSMPs into a term of IWMC. The latter mapping is obtained by abstracting from probability related information. Such mapping is still performed compositionally, but at the level of models (not at the level of algebraic terms). In particular we define how to derive an ITA from an IGSMP by turning probabilistic choices into non-deterministic choices and by considering the support of the distribution of a clock, i.e. the set of time values that may happen with probability (density) greater than 0, as the set of possible values for its duration. Moreover we show that such mapping is compositional, i.e. is preserved by CSP parallel composition and hiding. If every distribution used in the GSMP has a support which is a finite collection of intervals, then the derived ITA is analyzable with existing techniques and tools.

4.1 Deriving the Pure Markovian Process

Given an IGSMP term $P \in IGSMP_g$ (see [8] or [5] Chapter 7), we derive an IWMC term $Q \in IWMCg$ by approximating general distribution with phase-type distributions.

Since phase-type distributions can be seen as the time to absorption of a continuous time Markov chain, any phase-type distribution *pht* can be represented by some term P_{pht} of IWMC, made up of only weighted prefixes "$w._-$", exponentially timed prefixes "$\lambda._-$", choice operators "$\_ + \_$" and occurrences of a variable X representing absorbing states.

Given a function $approx : PDF^+ \multimap PhT$, which associates with each general distribution f occurring in an IGSMP specification P its approximating phase-type distribution *pht*, term $Q \in IWMCg$ is obtained as follows. Denoted with $R[R'/X]$ the term obtained from a term R by replacing R' for X inside R, we just replace each occurrence of a subterm $<f, w>.P'$ in P with

$w.(P_{approx(f)}[P'/X])$.

Definition 4.1 Given $P \in IGSMP_g$ and a function $approx : PDF^+ \multimap PhT$, which associates with each general distribution occurring in P an approximating phase-type distribution, we define $\mathcal{M}[\![P, approx]\!] \in IWMCg$ to be the term obtained by replacing each occurrence of a subterm $<f, w>.P'$ in P with $w.(P_{approx(f)}[P'/X])$. ■

The following theorem, where we denote by $approx(P)$ the term of $IGSMP_g$ obtained from $P \in IGSMP_g$ by replacing distributions f in prefixes $<f, w>$ according to $approx$, shows the correctness of the mapping from IGSMP to IWMC terms (performance measures are preserved).

Theorem 4.2 *Given $P \in IGSMP_g$ and $approx : PDF^+ \multimap PhT$, we have that, for every fixed adversary resolving non-deterministic choices, the stochastic process underlying $approx(P)$ is the same as that underlying $\mathcal{M}[\![P, approx]\!]$ (provided that in $\mathcal{M}[\![P, approx]\!]$ we only consider states which do not enable derivatives of terms $P_{approx(f)}$, for any f, as states of the underlying stochastic process).* ■

The following theorem shows that, thanks to the fact that the semantics of IGSMP delays are defined by means of an ST semantics, observational equivalence is preserved when delays are refined by means of phase-type distributions. We denote with $\simeq_{IGSMP}$ observational equivalence over IGSMP terms (defined in [8] or [5] Chapter 7).

Theorem 4.3 *Given $P, Q \in IGSMP_g$ and a function $approx : PDF^+ \multimap PhT$, we have that $P \simeq_{IGSMP} Q$ implies $\mathcal{M}[\![P, approx]\!] \simeq_{IWMC} \mathcal{M}[\![Q, approx]\!]$.*

The simple mapping above from IGSMP terms into IWMC terms is significant from a pure performance viewpoint in that it shows process algebra to provide exactly the machinery necessary for approximating GSMPs with CTMCs through phase-type distributions. This because, while directly transforming at the model level a GSMP into a CTMC via phase-type approximation is really cumbersome due to the interleaving of the exponential phases, when using process algebra we just have to approximate general distributions at the term level and then the parallel operator automatically computes the interleaving of exponential phases for us. Finally, such a mapping confirms ST semantics to be the adequate semantics for generally distributed time in that approximation of activity durations with phase-type distributions is a form of action refinement.

4.2 Deriving the Pure Real-Time Process

Given an IGSMP $\mathcal{G} = (\Sigma, \mathcal{C}, D, Act, T_+, T_-, T_a, s_0)$ (see [7] or [5] Chapter 6), we derive an ITA $\mathcal{T} = (\Sigma, \mathcal{C}, Act, T_r, T_b, T_a, s_0)$, by turning probabilistic choices into non-deterministic choices and by considering the support of the distribution of a clock as the set of possible values for its duration. In particular, clock

start transitions C_i^+ are turned into reset transitions C_i, while clock termination transitions C_i^- are turned into clock bound transitions $C_i \in T$, where T is the support of the distribution $D(C_i)$. Note that a technique like this, which is based on the idea that we introduced in [6] of considering support of distributions as constraints over clocks, was also used in [9] for deriving timed automata from the stochastic automata model of [10]. Subsequently, in [11] it was shown that a more complex technique, which generates new states for each interval composing the domain of the support of the probability distribution of clocks, is actually needed for correctly deriving timed automata from the model of [10]. This because it can be seen that in such a model the direct transformation of clock termination transitions into transitions requiring clocks to assume values in the support of their distributions causes timed automata which behave differently from the original system to be derived. This is due to the fact that in the model of [10] it may happen that a clock termination transition is executed some time after the clock the transition refers to actually terminates. Since such a phenomenon cannot happen in IGSMPs, our simple technique which does not increase the system state space, can be correctly applied.

Now we present the precise definition of support of a probability distribution that we need for the translation. We follow the idea of [11] of defining the support (therein called "useful domain") in such a way that, if a time value is in the support set, then either it has non-zero measure, or it is internal, i.e. it belongs to an open interval which is all included in the support set (and which must have non-zero measure). This avoids considering traces containing action orderings which in the original IGSMP occur with zero probability (see [11]).

Definition 4.4 Given a probability distribution f over $\mathbb{R}$, the support of f, denoted by $supp(f)$, is the set obtained from the least closed subset of $\mathbb{R}$ with measure 1 by eliminating non-internal values with measure 0. ■

It is trivial to verify that for each probability distribution f, $supp(f)$ has measure 1 (hence that the definition is correct).

Definition 4.5 Given an IGSMP $\mathcal{G} = (\Sigma, \mathcal{C}, D, Act, T_+, T_-, T_a, s_0)$, we define $\mathcal{T}[\![\mathcal{G}]\!]$ to be the ITA $(\Sigma, \mathcal{C}, Act, T_r, T_b, T_a, s_0)$, where T_r and T_b are given by

- $T_r = \{(s, C_i, s') \mid (s, C_i^+, s') \in T_+\}$
- $T_b = \{(s, C_i \in T, s') \mid (s, C_i^-, s') \in T_- \wedge T = supp(D(C_i))\}$ ■

In order to show the correctness of the mapping from IGSMP to ITA, we assume the following. Given a state s of an IGSMP and a valuation function v assigning a time value to each of its clocks, we call a "supported execution of an IGSMP starting in (s, v)" a finite sequence of timed transitions $t \in \mathbb{R}^+ \cup 0$ and actions transitions $\alpha \in Act$ executable by the IGSMP according to its semantics (see [7,5]) when it starts in the state s with initial valuation v and when we consider as possible values sampled for a clock with distribution f

the time values in $supp(f)$ only. Similarly a "possible execution of an ITA starting in (s, v)" is a finite sequence of timed transitions $t \in \mathbb{R}^+ \cup 0$ and actions transitions $\alpha \in Act$ executable by the ITA according to its semantics (see Appendix B) when it starts in the state s with initial valuation v.

Theorem 4.6 *Given an IGSMP $\mathcal{G} = (\Sigma, \mathcal{C}, D, Act, T_+, T_-, T_a, s_0)$, we have that for each state s and valuation function v associating a time value to the clocks of $\mathcal{G}$ (belonging to $\mathcal{C}$) the set of all supported executions of $\mathcal{G}$ starting in (s, v) is equal to the set of all possible executions of the $\mathcal{T}[\![\mathcal{G}]\!]$ starting in (s, v).* ■

The following theorem shows that weak bisimulation equivalence is preserved when well-named IGSMPs are mapped into ITA. We denote with $\approx_{IGSMP}$ weak bisimulation over well-named IGSMPs (defined in [8,7] or [5] Chapter 6).

Theorem 4.7 *Given two well-named IGSMPs $\mathcal{G}'$ and $\mathcal{G}''$, we have that $\mathcal{G}' \approx_{IGSMP} \mathcal{G}''$ implies $\mathcal{T}[\![\mathcal{G}']\!] \approx_{ITA} \mathcal{T}[\![\mathcal{G}'']\!]$. Moreover, for each $S, L \subseteq Act - \{\tau\}$, we have $\mathcal{T}[\![\mathcal{G}']\!] \|_S \mathcal{T}[\![\mathcal{G}'']\!] \approx_{ITA} \mathcal{T}[\![\mathcal{G}' \|_S \mathcal{G}'']\!]$ and $\mathcal{T}[\![\mathcal{G}]\!]/L \approx_{ITA} \mathcal{T}[\![\mathcal{G}/L]\!]$.*

References

[1] L. Aceto, M.C.B. Hennessy, *"Adding Action Refinement to a Finite Process Algebra"*, in Information and Computation 115:179-247, 1994

[2] R. Alur, C. Courcoubetis, D. Dill *"Model-Checking in Dense Real-Time"*, in Information and Computation 104:2-34, 1993

[3] M. Bernardo, *"Theory and Application of Extended Markovian Process Algebra"*, Ph.D. Thesis, University of Bologna (Italy), 1999

[4] A. Bobbio, A. Horváth, M. Telek *"The Scale Factor: A New Degree of Freedom in Phase Type Approximation"*, to appear in Proc. of the *Int. Performance & Dependability Symposium (IPDS '02)*, Washington (DC), 2002

[5] M. Bravetti, *"Specification and Analysis of Stochastic Real-Time Systems"*, Ph.D. Thesis, University of Bologna (Italy), 2002. Available at `http://www.cs.unibo.it/~bravetti`

[6] M. Bravetti, *"Towards the Integration of Real-Time and Probabilistic-Time Process Algebras"*, in Proc. of the *3rd European Research Seminar on Advances in Distributed Systems (ERSADS '99)*, Madeira Island (Portugal), April 1999

[7] M. Bravetti, A. Aldini, *"Non-Determinism in Probabilistic Timed Systems with General Distributions"*, in Proc. of the *2nd Int. Workshop on Models for Time-Critical Systems (MTCS 2001)*, ENTCS 52.3, Aalborg (Denmark), August 2001

[8] M. Bravetti, R. Gorrieri, *"The Theory of Interactive Generalized Semi-Markov Processes"*, to appear in Theoretical Computer Science

[9] J. Bryans, J. Derrick, *"Stochastic Specification and Verification"*, In Proc. of the *3rd Irish Workshop in Formal Methods*, Electronic Workshops in Computing, July 1999

[10] P.R. D'Argenio, *"Algebras and Automata for Timed and Stochastic Systems"*, Ph.D. Thesis, Univ. Twente, 1997

[11] P.R. D'Argenio, *"A Compositional Translation of Stochastic Automata into Timed Automata"*, Technical Report CTIT 00-08, Univ. Twente, May 2000

[12] P.W. Glynn, *"A GSMP formalism for discrete event simulation"*, in Proc. of the IEEE, 77(1): 14-23, 1989

[13] H. Hermanns, *"Interactive Markov Chains"*, Ph.D. Thesis, Universität Erlangen-Nürnberg (Germany), 1998

[14] H. Hermanns, *"An Operator for Symmetry Representation and Exploitation in Stochastic Process Algebras"*, in Proc. of the *5th Workshop on Process Algebras and Performance Modeling*, pp. 55-70, Twente (The Netherlands), 1997

[15] J. Hillston, *"A Compositional Approach to Performance Modelling"*, Cambridge University Press, 1996

[16] K.G. Larsen, A. Skou, *"Bisimulation through Probabilistic Testing"*, in Information and Computation 94:1-28, 1991

[17] R. Milner, *"Communication and Concurrency"*, Prentice Hall, 1989

[18] X. Nicollin, J. Sifakis, *"An Overview and Synthesis on Timed Process Algebras"*, in Real-Time: Theory in Practice, LNCS 600, 1991

[19] X. Nicollin, J. Sifakis, S. Yovine, *"Compiling Real-Time Specifications into Extended Automata"*, in IEEE Trans. on Software Engineering, 18(9):794-804, 1992

[20] M. Scarpa, A. Bobbio, *"Kronecker representation of Stochastic Petri nets with discrete PH distributions"*, in Proc. of the *3rd Int. Performance & Dependability Symposium (IPDS '98)*, pp. 52-61, Durham (NC), 1998

[21] C.M.N. Tofts, *"Processes with Probabilities, Priority and Time"*, in Formal Aspects of Computing 6:536-564, 1994

A A Complete Axiomatization for finite state IWMC terms

In this section we present an axiom system which is complete for $\simeq_{IWMC}$ on finite state $IWMC_g$ terms.

$(A1)$	$P + Q = Q + P$	$(A2)$	$(P + Q) + R = P + (Q + R)$
$(A3)$	$\alpha.P + \alpha.P = \alpha.P$	$(A4)$	$P + \underline{0} = P$
$(Tau1)$	$\gamma.\tau.P = \gamma.P$	$(Tau2)$	$P + \tau.P = \tau.P$
$(Tau3)$	$\alpha.(P + \tau.Q) + \alpha.Q = \alpha.(P + \tau.Q)$		
$(Prob)$	$w.P + w'.P = (w + w').P$		
$(ExpT)$	$\lambda.P + \lambda'.P = (\lambda + \lambda').P$		
$(Pri1)$	$\tau.P + \theta.Q = \tau.P$	$(Pri2)$	$w.P + \lambda.Q = w.P$
$(Hi1)$	$\underline{0}/L = \underline{0}$	$(Hi2)$	$(\gamma.P)/L = \gamma.(P/L) \quad \gamma \notin L$
$(Hi3)$	$(a.P)/L = \tau.(P/L) \quad a \in L$	$(Hi4)$	$(P + Q)/L = P/L + Q/L$
$(Rel1)$	$\underline{0}[\varphi] = \underline{0}$	$(Rel2)$	$(\alpha.P)[\varphi] = \varphi(\alpha).(P[\varphi])$
$(Rel3)$	$(\theta.P)[\varphi] = \theta.(P[\varphi])$	$(Rel4)$	$(P + Q)[\varphi] = P[\varphi] + Q[\varphi]$
(Par)	$P \parallel_S Q = P \between_S Q + Q \between_S P + P \mid_S Q$		
$(LM1)$	$\underline{0} \between_S P = \underline{0}$		
$(LM2)$	$(a.P) \between_S Q = \underline{0} \quad a \in S$		
$(LM3)$	$(\gamma.P) \between_S Q = \gamma.(P \parallel_S Q) \quad \gamma \notin S$		
$(LM4)$	$(P + Q) \between_S R = P \between_S R + Q \between_S R$		
$(SM1)$	$P \mid_S Q = Q \mid_S P$		
$(SM2)$	$\underline{0} \mid_S P = \underline{0}$		
$(SM3)$	$(\gamma.P) \mid_S (\gamma'.Q) = \underline{0} \quad (\gamma \notin S \vee \gamma \neq \gamma') \wedge \tau \notin \{\gamma, \gamma'\}$		
$(SM4)$	$(\tau.P) \mid_S Q = P \mid_S Q$		
$(SM5)$	$(a.P) \mid_S (a.Q) = a.(P \parallel_S Q) \quad a \in S$		
$(SM6)$	$(P + Q) \mid_S R = P \mid_S R + Q \mid_S R$		
$(Rec1)$	$recX.P = recY.(P\{Y/X\})$ provided that Y is not free in recX.P		
$(Rec2)$	$recX.P = P\{recX.P/X\}$		
$(Rec3)$	$Q = P\{Q/X\} \Rightarrow Q = recX.P$ provided that X is strongly guarded in P		

Fig. A.1. Axiomatization for IWMC

The axiom system $\mathcal{A}_{IWMC}$ for $\simeq_{IWMC}$ on $IWMC_g$ terms is formed by the axioms presented in Fig. A.1. In this figure "$\between$" and "$\mid$" denote, respectively, the left merge and synchronization merge operators. We recall from Sect. 2 that θ ranges over weights in $\mathbb{R}^+$ and rates in $\mathbb{R}^+$, while $\gamma, \gamma', \ldots$ range over actions in Act, weights and rates.

The axioms $(Pri1)$ and $(Pri2)$ express the two kinds of priorities of $IWMC$, respectively, priority of τ actions over weights and rates and priority of weights over rates. The axiom (Par) is the standard one which expresses parallel composition in terms of left and synchronization merge. The axioms $(Rec1-3)$ handle strongly guarded recursion in the standard way.

If we consider the obvious operational rules for "$\|_S$" and "$|_S$" that derive from those we presented for the parallel operator [5] then the axioms of $\mathcal{A}_{IWMC}$ are sound.

A sequential state is defined to be one which includes "$\underline{0}$", "X" and operators ".", "+", "$recX$" only; leading to the following theorem.

Theorem A.1 *If an $IWMC_g$ process P is finite state, then $\exists P' : \mathcal{A}_{IWMC} \vdash P = P'$ with P' sequential state.*

For sequential states the axioms of $\mathcal{A}_{IWMC}$ involved are just standard axioms plus the axioms for priority and probabilistic and exponentially timed choice, hence we have the following.

Theorem A.2 *$\mathcal{A}_{IWMC}$ is complete for $\simeq_{IWMC}$ over finite state $IWMC_g$ processes.*

B A Semantics for Interactive Timed Automata

In Sect. B.1 we introduce Interactive Prioritized Timed Transition Systems (IPTTSs) that will be used in Sect. B.2 to define a semantics for ITA.

B.1 Interactive Prioritized Timed Transition Systems

In this section we formally introduce Interactive Prioritized Timed Transition Systems (IPTTS) which essentially include three type of transitions: *standard action transitions*, representing the interactive behavior of a system component, *prioritized transitions*, representing behaviors of the system component executed according to a certain priority level, and *numeric time transitions* representing a fixed temporal delay.

As far as standard action transitions are concerned they have exactly the same behavior as in ITA. Prioritized transitions are labeled with a certain priority level $p \in \mathbb{N}^+$ and, where transitions with a higher priority level take priority (e.g. when composing two IPTTSs in parallel) over prioritized transitions with a lower priority level. Moreover, we assume standard τ transitions to take priority over prioritized transitions, no matter which is the priority level of such transitions (due to the maximal progress assumption). Given a time domain $TD \subseteq \mathbb{R}^+$, numeric time transitions are labeled with a certain delay $t \in TD$ representing the passage of t time units. As usual in the real

[5] The definition of the operational rule for "$|_S$" must allow for actions "τ" to be skipped, as reflected by axiom $(SM4)$.

time literature (see e.g. [19]), several timed transition leaving a state offer the possibility to the observer to choose the amount of time after which he wants to observe the status of the system.

In IPTTS we have two different kinds of state:

- *silent states* which are exactly like in ITA.
- *non-silent states* enabling numeric timed transitions and/or prioritized transitions all with the same priority level and (possibly) visible action transitions a, only. In such states numeric timed transitions (which cause the amount of time labeling the transition to pass) and prioritized transitions are chosen by means of a non-deterministic choice. Moreover the IPTTS may potentially interact with the environment through one of its visible actions.

In the following we present the formal definition of Interactive Prioritized Timed Transition System (IPTTS), then we will define Rooted Interactive Prioritized Timed Transition Systems as IPTTSs possessing an initial state. Formally, given a time domain $TD \subseteq \mathbb{R}^+$, we use $t, t', \ldots$, representing time values, to range over TD. Moreover we use $p, p', \ldots$, representing priority levels, to range over $\mathbb{N}^+$. Finally we use θ to range over time values in TD and priorities in $\mathbb{N}^+$.

Definition B.1 An Interactive Prioritized Timed Transition System (IPTTS) is a tuple $\mathcal{D} = (\Sigma, TD, Act, T_p, T_t, T_a)$ with

- Σ a set of possibly infinite states,
- TD a time domain, i.e. the set of possible values over which the labels of the numeric timed transitions range.

- Act a set of standard actions,
- $T_p \subseteq (\Sigma \times \mathbb{N}^+ \times \Sigma)$ and $T_t \subseteq (\Sigma \times \mathbb{R}^+ \times \Sigma)$ and $T_a \subseteq (\Sigma \times Act \times \Sigma)$ three transition relations representing prioritized behaviors, time passage and action execution, respectively. T_p, T_t and T_a must be such that $\forall s \in \Sigma$.
  - $s \xrightarrow{\tau} \implies \nexists t.s \xrightarrow{t} \wedge \nexists p.s \xrightarrow{p}$
  - $s \xrightarrow{p} \implies \nexists p' < p.s \xrightarrow{p'}$
  - $s \xrightarrow{\tau} \vee \exists t.s \xrightarrow{t} \vee \exists p.s \xrightarrow{p}$ ■

Definition B.2 A Rooted Interactive Prioritized Timed Transition System (RIPTTS) is a tuple $\mathcal{D} = (\Sigma, TD, Act, T_p, T_t, T_a, s_0)$, where $s_0 \in \Sigma$ is the initial state and $(\Sigma, TD, Act, T_p, T_t, T_a)$ is an IPTTS. ■

The meaning of the constraints over transition relations is the following. The first requirement says that (similarly as in ITA) if a state that can perform internal τ actions then it cannot perform time-related transitions (*maximal progress* assumption). The second requirement says that if a state can perform prioritized transitions with a certain priority level then it cannot perform

prioritized transitions with a lower priority level. The third requirement says that (similarly as in ITA) we cannot have states where time is not allowed to pass (time deadlocks).

B.1.1 Parallel of Rooted IPTTSs

Now we define, similarly as for ITA, the parallel composition à la CSP of RIPTTSs.

In such a parallel composition the discrete timed transitions of the composed RIPTTSs are constrained to synchronize, so that the same amount of time passes for both systems, i.e. when time advances for one RIPTTS it must also advance for the other RIPTTS.

Definition B.3 The parallel composition $\mathcal{D}_1 \parallel_S \mathcal{D}_2$ of two RIPTTSs $\mathcal{D}_1 = (\Sigma_1, TD, Act, T_{p,1}, T_{t,1}, T_{a,1}, s_{0,1})$ and $\mathcal{D}_2 = (\Sigma_2, TD, Act, T_{p,2}, T_{t,2}, T_{a,2}, s_{0,2})$, with $S \subset Act - \{\tau\}$ being the synchronization set, is the tuple $(\Sigma, TD, Act, T_p, T_t, T_a, (s_{0,1}, s_{0,2}))$ with:

- $\Sigma = \Sigma_1 \times \Sigma_2$ the set of states
- $T_p \subseteq (\Sigma \times \mathbb{N}^+ \times \Sigma)$, $T_t \subseteq (\Sigma \times TD \times \Sigma)$ and $T_a \subseteq (\Sigma \times Act \times \Sigma)$ the least transition relations, such that

$$\mathbf{1_l}\ \ s_1 \xrightarrow{\alpha} s_1',\ \alpha \notin S \implies (s_1, s_2) \xrightarrow{\alpha} (s_1', s_2)$$

$$\mathbf{1_r}\ \ s_2 \xrightarrow{\alpha} s_2',\ \alpha \notin S \implies (s_1, s_2) \xrightarrow{\alpha} (s_1, s_2')$$

$$\mathbf{2}\ \ s_1 \xrightarrow{a} s_1' \ \wedge\ s_2 \xrightarrow{a} s_2',\ a \in S \implies (s_1, s_2) \xrightarrow{a} (s_1', s_2')$$

$$\mathbf{3_l}\ \ s_1 \xrightarrow{p} s_1' \ \wedge\ s_2 \xrightarrow{\tau} \not\ \wedge\ \nexists p' > p.s_2 \xrightarrow{p'} \implies (s_1, s_2) \xrightarrow{p} (s_1', s_2)$$

$$\mathbf{3_r}\ \ s_2 \xrightarrow{p} s_2' \ \wedge\ s_1 \xrightarrow{\tau} \not\ \wedge\ \nexists p' > p.s_1 \xrightarrow{p'} \implies (s_1, s_2) \xrightarrow{p} (s_1, s_2')$$

$$\mathbf{4}\ \ s_1 \xrightarrow{t} s_1' \ \wedge\ s_2 \xrightarrow{t} s_2' \implies (s_1, s_2) \xrightarrow{t} (s_1', s_2')$$

- $(s_{0,1}, s_{0,2}) \in \Sigma$ the initial state. ■

When evaluating action transitions we just make use of standard rules. Prioritized transitions are determined by taking into account priorities according to a "global" notion of priority where priorities are applied across the parallel operator. Finally timed transitions are evaluated by just requiring them to synchronize.

Theorem B.4 *Let $\mathcal{D}_1$ and $\mathcal{D}_2$ be two RIPTTSs. Then for each $S \subseteq Act - \{\tau\}$, $\mathcal{D}_1 \parallel_S \mathcal{D}_2$ is a RIPTTS.* ■

B.1.2 Hiding of Rooted IPTTSs

Now we define, similarly as for ITA, the hiding of RIPTTSs.

Definition B.5 The hiding $\mathcal{D}/L$ of a RIPTTS $\mathcal{D}_1 = (\Sigma, TD, Act, P_1, T_{p,1}, T_{t,1}, T_{a,1}, s_0)$, with $L \subset Act - \{\tau\}$ being the set of visible actions to be hidden, is

the tuple $(\Sigma, TD, Act, P, T_p, T_t, T_a, s_0)$, with:

- P the partial function obtained from P_1 by removing from its domain those states (and the associated probability spaces) which enable at least one transition labeled with an action in L
- $T_p \subseteq (\Sigma \times \mathbb{N}^+ \times \Sigma)$, $T_t \subseteq (\Sigma \times TD \times \Sigma)$ and $T_a \subseteq (\Sigma \times Act \times \Sigma)$ the least transition relations, such that $\forall s \in \Sigma$. [6]

1 $s \xrightarrow{\alpha}_1 s',\ \alpha \notin L \implies s \xrightarrow{\alpha} s'$

2 $s \xrightarrow{a}_1 s',\ a \in L \implies s \xrightarrow{\tau} s'$

3 $s \xrightarrow{\theta}_1 \wedge\ \nexists a \in L.\, s \xrightarrow{a}_1 \implies s \xrightarrow{\theta}$

■

Similarly as for ITA, in the definition of the hiding operator in addition to standard rules we make use of rules which enforce the maximal progress assumption.

Theorem B.6 *Let $\mathcal{D}$ be a RIPTTS. Then for each $L \subseteq Act - \{\tau\}$, $\mathcal{D}/L$ is a RIPTTS.* ■

B.1.3 Equivalence of Rooted IPTTSs

Now we introduce a notion of weak bisimulation for RIPTTSs which matches prioritized and timed transitions according to strong bisimulation and abstracts from standard τ actions similarly as in [17].

Definition B.7 Let $\mathcal{D} = (\Sigma, TD, Act, T_p, T_t, T_a)$ be an IPTTS. An equivalence relation β on Σ is a *weak bisimulation* iff $s_1\ \beta\ s_2$ implies

- for every $\alpha \in Act$,
$$s_1 \xrightarrow{\alpha} s_1' \text{ implies } s_2 \stackrel{\hat{\alpha}}{\Longrightarrow} s_2' \text{ for some } s_2' \text{ with } s_1'\ \beta\ s_2',$$
- for every $\theta \in \mathbb{N}^+ \cup TD$,
$$s_1 \xrightarrow{\theta} s_1' \text{ implies } s_2 \xrightarrow{\theta} s_2' \text{ for some } s_2' \text{ with } s_1'\ \beta\ s_2',$$

Two states s_1 and s_2 are weakly bisimilar, denoted by $s_1 \approx s_2$, iff (s_1, s_2) is included in some weak bisimulation. Two RIPTTSs $(\mathcal{D}_1, s_{0,1})$ and $(\mathcal{D}_2, s_{0,2})$ are weakly bisimilar, if their initial states $s_{0,1}$ and $s_{0,2}$ are weakly bisimilar in the IPTTS obtained with the disjoint union of $\mathcal{D}_1$ and $\mathcal{D}_2$. ■

B.2 Definition of the Semantics for ITA

In this section we present a semantics for interactive timed automata which maps them onto interactive prioritized timed transition systems. Such a semantics explicitely represents the passage of time by means of transitions labeled with numeric time delays and turns clock reset transitions into prior-

[6] In order to distinguish transition of $T_{p,1}$, $T_{t,1}$ and $T_{a,1}$ from transitions of T_p, T_t and T_a we denote the former with "$\longrightarrow_1$" and the latter simply with "$\longrightarrow$".

itized transitions with priority level 2 and clock bound transitions into prioritized transitions with priority level 1.

$$(P1)\ \frac{s \xrightarrow{\phi} s' \ \wedge\ v \vdash \phi}{\langle s, v\rangle \xrightarrow{1} \langle s', v\rangle} \qquad (P2)\ \frac{s \xrightarrow{C_n} s'}{\langle s, v\rangle \xrightarrow{2} \langle s', v \leftarrow (C_n, 0)\,\rangle}$$

$$(T)\ \frac{\exists t' \geq t : v + t' \vdash \bigwedge\{\phi \mid s \xrightarrow{\phi}\}}{\langle s, v\rangle \xrightarrow{t} \langle s, v + t\rangle}$$

$$(A)\ \frac{s \xrightarrow{\alpha} s'}{\langle s, v\rangle \xrightarrow{\alpha} \langle s', v\rangle}$$

Table B.1
Semantic rules for ITA

We now formally define the semantics of an ITA.

Definition B.8 The semantics of an ITA $\mathcal{T} = (\Sigma, \mathcal{C}, Act, T_r, T_b, T_a, s_0)$ is the RIPTTS $[\![\mathcal{T}]\!] = (\Sigma', \mathbb{R}^+ \cup \{0\}, Act, T_p, T_t, T'_a, s'_0)$ where:

- $\Sigma' = (\Sigma \times \mathit{Spent})$ is the set of states of the RIPTTS, where *Spent*, ranged over by v, is the set of functions from $\mathcal{C}$ to $\mathbb{R}^+ \cup \{0\}$, expressing the time already spent in execution by the clocks of the ITA from the last reset event
- $\mathbb{R}^+ \cup \{0\}$ is the time domain: we consider continuous time.
- Act is the set of standard actions considered in the ITA.
- T_p is the set of prioritized transitions which are defined as the least relation over $\Sigma' \times \mathbb{N}^+ \times \Sigma'$ satisfying the operational rules in the first part of Table B.1.
- T_t is the set of timed transitions which are defined as the least relation over $\Sigma' \times (\mathbb{R}^+ \cup \{0\}) \times \Sigma'$ satisfying the operational rules in the second part of Table B.1.
- T'_a is the set of action transitions which are defined as the least relation over $\Sigma' \times Act \times \Sigma'$ satisfying the operational rules in the third part of Table B.1.
- $s'_0 = \langle s_0, \mathbf{0}\rangle$, with $\mathbf{0} = \{(C_n, 0) \mid C_n \in \mathcal{C}\}$ is the initial state of the RIPTTS, where the ITA is in the initial state and all clocks start from zero. ■

In Table B.1 we make use of the following notation. $v \vdash \phi$ holds true if and only if the formula obtained from ϕ by replacing clocks with time values

according to v is true. Moreover we define $v \leftarrow (C_n, t)$ to be the function obtained from v by replacing the pair (C_n, t') already contained in v with the new pair (C_n, t). Finally, we define $v + t$, with $t \in \mathbb{R}^+ \cup 0$, to be the function obtained from v by adding t to the time value associated with each clock in v.

Theorem B.9 *Let $\mathcal{T}'$, $\mathcal{T}''$ be two ITA. If $\mathcal{T}' \approx \mathcal{T}''$ then $[\![\mathcal{T}']\!] \approx [\![\mathcal{T}'']\!]$.* ■

The following theorems show that the semantics of ITA is indeed compositional.

Theorem B.10 *Let $\mathcal{T}'$, $\mathcal{T}''$ be two ITA. For each $S \subseteq Act - \{\tau\}$ we have $[\![\mathcal{T}']\!] \|_S [\![\mathcal{T}'']\!] \approx [\![\mathcal{T}' \|_S \mathcal{T}'']\!]$.* ■

Theorem B.11 *Let $\mathcal{T}$ be an ITA. For each $L \subseteq Act - \{\tau\}$ we have $[\![\mathcal{T}]\!]/L \approx [\![\mathcal{T}/L]\!]$.* ■

Revisiting Interactive Markov Chains

Mario Bravetti [1]

Dipartimento di Scienze dell'Informazione,
University of Bologna, Mura Anteo Zamboni 7, 40127 Bologna, Italy

Abstract

The usage of process algebras for the performance modeling and evaluation of concurrent systems turned out to be convenient due to their feature of compositionality. A particularly simple and elegant solution in this field is the calculus of Interactive Markov Chains (IMCs), where the behavior of processes is just represented by Continuous Time Markov Chains extended with action transitions representing process interaction. The main advantage of IMCs with respect to other existing approaches is that a notion of bisimulation which abstracts from τ transitions ("complete" interactions) can be defined which is a congruence. However in the original definition of the calculus of IMCs the high potentiality of compositionally minimizing the system state space given by the usage of a "weak" notion of equivalence and the elegance of the approach is somehow limited by the fact that the equivalence adopted over action transitions is a finer variant of Milner's observational congruence that distinguishes τ-divergent "Zeno" processes from non-divergent ones. In this paper we show that it is possible to reformulate the calculus of IMCs in such a way that we can just rely on simple standard observational congruence. Moreover we show that the new calculus is the first Markovian process algebra allowing for a new notion of Markovian bisimulation equivalence which is coarser than the standard one.

1 Introduction

The advantages of using process algebras for the performance modeling and evaluation of concurrent systems due to their feature of compositionality have been widely recognized (see [12,2,18,9,5,3] and the references therein). Particularly simple and successful has been the extension of standard process algebras with time delays whose duration follows an exponential probability distribution, called Markovian process algebras (see e.g. [12,2,18,9]). The "timed" behavior of systems specified with a Markovian process algebra can represented by a continuous time Markov chain (CTMC), i.e. a simple continuous time stochastic process where in each time point the future behavior of

[1] Email: bravetti@cs.unibo.it

This is a preliminary version and considered for the final proceedings, to be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

the process is completely independent on its past behavior. Due to their simplicity CTMCs can be analyzed with standard mathematical techniques and software tools (see e.g. [19]) for deriving performance measures of systems.

1.1 Interactive Markov Chains

In [9] specifying concurrent systems as the parallel composition of interacting subsystems described by CTMCs is made possible simply by extending CTMCs with standard action transitions, thus giving rise to Interactive Markov Chains (IMCs). An IMC represents the behavior of a component by employing both *standard action transitions*, representing the interactive behavior of the component, and *exponentially timed transitions*, representing the timed probabilistic behavior of the component. Action transitions are just standard CCS/CSP [14,13] transitions labeled with an action "α", which can be either an internal τ action or a visible action "a". They are executed in zero time: when several action transitions are enabled in an IMC state, the choice among them is just performed non-deterministically and when IMCs are composed in parallel they synchronize following the CSP [13] approach, where the actions belonging to a given set S are required to synchronize. Exponentially timed transitions are, instead, labeled with a rate λ (the parameter of the exponential distribution) and represent timed choices performed according to a "race" between exponential delays. The interrelation between standard action transitions and exponentially timed transitions is governed by the so-called *maximal progress assumption* [17]: the possibility of executing τ transitions prevents the execution of exponentially timed transitions, thus expressing that the system cannot wait if it has something internal to do. Visible a transitions are, instead, interpreted as representing a "potential" interaction with the environment, hence their execution can be indefinitely delayed. Therefore in the IMC obtained from the specification of "complete concurrent system" no visible action transition occurs. In [9] a process algebra (called calculus of IMCs) is defined, which is just a simple extension of a standard process algebra (containing CCS [14] prefix "$\alpha.P$" and choice "$P + Q$" and CSP [13] parallel composition "$P \|_S Q$" and hiding "P/L") with a new form of prefix "$\lambda.P$" representing an exponential time delay. The semantics of the calculus of IMC derives IMCs from algebraic terms by using the standard CCS/CSP semantics for action transitions and by essentially using an interleaving semantics for "λ" delay prefixes (this is correct due to the memoryless property of exponential delays).

The notion of weak bisimulation for IMCs that is presented in [9] essentially matches exponentially timed transitions according to Markovian bisimulation [12] and abstracts from standard τ similarly to [14]. Since such an equivalence is shown to be a congruence for the calculus, it makes it possible to significantly and efficiently minimize the state-space of complete systems by abstracting from process interaction in a compositional way.

However the high potentiality and the elegance of the approach of [9] is somehow limited by the fact that the equivalence adopted over action transitions is a finer variant of Milner's observational congruence that distinguishes τ-divergent "Zeno" processes from non-divergent ones. In particular, similarly to [8], the additional requirement is introduced that two bisimilar terms must have the same opportunity to silently become stable terms, i.e. terms that cannot perform τ actions.

In [11] it is claimed that, since the maximal progress assumption generates a priority mechanism, it is somehow necessary to have such a τ-divergence sensitive equivalence. In particular [11] shows how to adapt the standard Milner's sound and complete axiomatization of observational congruence for a basic algebra with prefix, choice and recursion, when exponential delay prefixes are introduced (in such a way that the corresponding operators of the calculus of IMCs are obtained) and the τ-divergence sensitive equivalence of [9] is considered.

1.2 Simplifying the Notion of Weak Equivalence

In [6] we made a first step in the direction of eliminating the condition about stability from the equivalences of [8,9] in the context of interactive timed processes. We showed that maximal progress and Milner's standard notion of observational congruence are indeed compatible: it is possible to obtain a complete axiomatization for the basic interactive timed algebra of [11] even if the equivalence considered is not sensible to τ divergence.

Moreover, it is worth noting that in [6] we express priority arising from maximal progress by cutting transitions which cannot be performed directly in the operational semantics, instead of capturing such priority in the definition of equivalence as done in [9] (a solution also hinted in [10]). This technique allows us to obtain smaller system models and to further simplify the notion of equivalence considered in [9].

Unfortunately the results obtained in [6] for the basic interactive timed algebra do not scale to the full calculus of IMC [9]. This because the equivalence, being it not sensible to τ divergence, would not be a congruence for the parallel operator. The problem with congruence is that, e.g., while $\tau.\underline{0} \simeq recX.\tau.X$, since the parallel operator behaves in such a way that the *presence* of a τ action within the actions immediately executable by a process pre-empts the other process from executing a timed action λ (global pre-emption [7]) we have that $\tau.\underline{0} \parallel_\emptyset \lambda.\underline{0} \not\simeq recX.\tau.X \parallel_\emptyset \lambda.\underline{0}$. [2] This because the semantics of $\tau.\underline{0} \parallel_\emptyset \lambda.\underline{0}$ is that of $\tau.\lambda.\underline{0}$, while the semantics of $recX.\tau.X \parallel_\emptyset \lambda.\underline{0}$ is that of $recX.\tau.X$ (where no λ action can be executed). Note that this problem arises both in the case we capture priority in the notion of equivalence as done in [9] and in the case we enforce it in the definition of the operational semantics of the

[2] Here and in the rest of the paper we assume the following operator binding precedence: prefix > recursion > parallel composition > choice.

parallel operator with the technique of [6]. In the following we will suppose priority to be captured in the semantics of operators and equivalence to be "neutral" with respect to priority.

Conceptually, the problem above derives from the fact that the parallel operator deals with the terminated process $\underline{0}$ (and in general with processes which cannot execute neither τ actions nor λ actions) as if it let time pass. For example $\underline{0} \|_\emptyset \lambda$ may execute λ and become $\underline{0} \|_\emptyset \underline{0}$. This is obviously in contrast with the fact that $\underline{0}$ is weakly bisimilar to $recX.\tau.X$, which is clearly a process that does not let time pass (it represents a so-called time deadlock).

1.3 A New Markovian Calculus

As a consequence of the previous discussion, a very clean solution is to consider as processes which can let time pass only processes which can actually execute λ actions. In this way $\underline{0}$ is interpreted not as a terminated process which may let time pass, but as a time deadlock. As a consequence the definition of the parallel operator changes. In particular the parallel operator must be defined, similarly as in [8], in such a way that the *absence* of λ actions within the actions executable by a process (which means that the process cannot let time pass) pre-empts the other process from executing a timed action λ. Pre-emption caused by the *absence* of λ actions differs from pre-emption caused by the *presence* of τ actions exactly for the class of processes that were misinterpreted, i.e. processes which cannot execute neither τ actions nor λ actions. The new interpretation of such processes (as in [8]) is that, consistently with weak bisimilarity, either they immediately execute a visible action or they cause a time deadlock.

Based on this idea, in this paper we will define the new calculus of "Revisited" IMCs (RIMCs). In particular, the difference between IMCs and RIMCs at the transition system level is just in the meaning of states which cannot execute neither τ actions nor λ actions: RIMCs do not allow time to elapse in such states as, instead, IMCs do. Note that, as for IMCs, we can derive a CTMC from a complete system specification only if the derived RIMC cannot incur time deadlocks, i.e. states executing infinite sequences of τ transitions (as for IMCs) or equivalently states with no outgoing transitions (for RIMCs only).

As already explained, in the calculus we will define the rules for the parallel operator in such a way that, when we derive an exponentially timed move of $P \|_S Q$ from a corresponding move of P we require that also Q may perform an exponentially timed move, instead of requiring that Q must not perform a τ move. Note that, differently from [8], even if we require that Q may perform an exponentially timed move, we do not actually perform it because of the memoryless property of exponential delays.

Moreover, w.r.t the calculus of IMCs, in the calculus of RIMCs it is important (for "modeling convenience" and for the reasons that we will explain

in Sect. 1.4) to also modify prefix and choice by considering operators similar to those of [8]:

- A new prefix operator $\gamma; P$ which is defined: as $\gamma.P$ if γ is τ or λ, as $recX.(\gamma.P + \tilde{\lambda}.X)$, for some $\tilde{\lambda}$, otherwise (where "$recX$" denotes recursion). Such a prefix, which allows visible actions to be delayed as in the calculus of IMCs (hence is suitable for specifying systems), becomes the unique prefix operator in the new calculus, while we will use the "basic" prefix $\gamma.P$ as an auxiliary operator to be used just for building an axiomatization.
- A new choice operator $P \diamond Q$ which, similarly as for the new parallel operator, allows one of P and Q to let time pass only if the other one may let time pass and is defined in such a way that delay execution does not resolve the choice. Such a choice operator, which allows new prefixes $a; P$ (where a is a visible action) to be used without causing the delays λ preceding the execution of the a to solve the choice (hence is suitable for specifying systems), becomes the unique choice operator in the new calculus, while we will use the "basic" choice $P + Q$ as an auxiliary operator to be used just for building an axiomatization.

Finally, we also include in the calculus of RIMC a new symbol "$\underline{1}$" representing a terminated process which may let time elapse (as for "$\underline{0}$" in the calculus of IMCs) defined as $recX.\tilde{\lambda}.X$, for some $\tilde{\lambda}$.

It is worth noting that, from the modeling viewpoint, we can mimic the behavior of the choice operator of the calculus of IMCs, where $\lambda.P + \mu.Q$ represents a choice between P and Q decided by a "race" between the λ and μ delays, by means of term $\lambda; \tau; P \diamond \mu; \tau; Q$ of the calculus of RIMC.

1.4 A New Notion of Markovian Equivalence

As we will see, the calculus of RIMCs, based on the ideas presented in the previous section, also allows for a new notion of Markovian bisimulation equivalence which is coarser than the standard one of [12]. The new version of Markovian equivalence is based on the new idea of "observability" of exponential delays.

As explained in the previous section, the behavior of the new prefix "$\gamma; P$" and of the new symbol "$\underline{1}$" is defined in terms of a generic rate $\tilde{\lambda}$ whose particular value is not important. In particular, $\tilde{\lambda}$ is the rate of an exponentially timed transition leading back to the state in which it is executed (a "selfloop"). Such a definition is consistent from the probabilistic viewpoint because the (transient) behavior of a CTMC (hence also its steady state behavior), defined as the probability of being in a certain state at a certain time, does not depend on the presence of exponentially timed selfloops in states (hence on the particular values for the rate labeling such selfloops). Intuitively, *as long as we consider the firing of exponentially timed transitions to be unobservable* as in CTMCs, it is easy to see that the particular values chosen for selfloop rates in a state of a RIMC do not change its behavior (hence

that of derived CTMCs). We distinguish the following two cases. If the state has other outgoing exponentially timed transitions (which are not selfloops) then, the behavior of the RIMC in the state will be just as if selfloops are not present. This because, in the case a selfloop fires before one of the outgoing exponentially timed transitions causes the RIMC to leave the state, when the state is re-entered we can consider, thanks to the memoryless property, outgoing exponentially timed transitions to *continue* from the accomplishment level they reached before such event. Otherwise, if the state does not have other outgoing exponentially timed transitions, the RIMC will stay in the state forever, independently of the particular values of selfloop rates.

Even if in principle considering exponential delays as being "unobservable" could be done for every Markovian specification paradigm, to the best of our knowledge the calculus of RIMCs is the first Markovian process algebra to be compatible with unobservable exponential delays. This because, while all Markovian process algebras previously developed in the literature (see [12,2,18,9] and the references therein) make use of a "$P+Q$" choice operator such that an exponential move of P or Q resolves the choice (hence such a move is indeed "observed by the operator"), all the operators of the calculus of RIMCs (excluding the auxiliary ones to be used in the axiomatization) intuitively do not observe individual exponential firings, but just the global time to the occurrence of the next standard α action.

More precisely, supposing exponential delays are unobservable, we can modify the standard definition of Markovian bisimulation equivalence [12] as follows. Instead of requiring that *every* bisimulation equivalence class must be reached with the same aggregated rate by bisimilar terms, we can just require that this must hold for all equivalence classes *apart from the class including the terms themselves*. We will show that the new notion of Markovian bisimulation equivalence, which preserves the behavior of the underlying CTMC since it just adds insensitivity to rate of selfloops, is a congruence for the calculus of RIMC. On the contrary, such a notion is not a congruence for all existing Markovian process algebras, due to the presence of the "observing" choice operator "$P+Q$".

The notion of observational equivalence that we consider for the calculus of RIMCs is a combination of standard observational congruence and the new notion of Markovian bisimulation equivalence above. In spite of the problem with congruence arising with unobservable delays, the prefix $\theta.P$ and choice "$P+Q$" operators of the calculus of IMC [9] (which are also part of the basic interactive timed calculus for which we have developed a complete axiomatization of observational congruence in [6]) will play a fundamental role in building an axiomatization of such an equivalence. In particular we will build the axiom system by extending the calculus of RIMC with the "observable" exponential delays of [9], denoted by λ^o, and by considering standard Markovian bisimulation equivalence over such delays. In this way, by supposing that θ can be an observable delay λ^o and that "$P+Q$" only works with delays

which are observable, we do not break the congruence property.

1.5 Contents of the Paper

In Sect. 2 we define RIMCs and the syntax and semantics of the calculus of RIMCs which contains the "$\gamma; P$", the "$P \diamond Q$", the "$P \parallel_S Q$" and the "P/L" operators and the symbols "$\underline{1}$" and "$\underline{0}$". Moreover we define observational congruence over RIMC terms simply as a combination of our "improved" notion of Markovian bisimulation and the standard notion of observational congruence of [14] and we show that it is indeed a congruence for the new calculus.

In Sect. 3 we present a sound axiomatization for our notion of observational congruence which is complete for strongly guarded finite-state processes of the new calculus. Such an axiomatization is built by: (i) introducing transition systems extending RIMCs with the "observable" exponential delay transitions used in IMCs, (ii) by consequently extending our notion of observational congruence so that standard Markovian bisimulation [12] is used over "observable" exponential delays, and (iii) by introducing some auxiliary operators: the prefix "$\theta.P$" (extended to sequences) and the choice "$P + Q$" operators of the calculus of IMCs [9]; the new operator "$\mathcal{H}(P)$" which "hides" exponential delays by turning each "observable" λ^o into an "unobservable" λ; the operator $pri(P)$ introduced in [6] (where we show it to be necessary also for axiomatizing unguarded recursion) that eliminates non-prioritized behaviors; the operators "$P \lfloor\!\lfloor_S Q$" and "$P \lfloor_S Q$" that are simple variants of the left merge and synchronization merge operators of [1], and finally the operator "$P \triangleleft Q$" which is a sort of left merge operator used for axiomatizing choice "$P \diamond Q$". We present the semantics of all auxiliary operators and we show that they preserve the congruence property.

Note that, since, to the best of our knowledge, developing an axiomatization of observational congruence for finite-state processes with unguarded recursion in the presence of "static" operators (like e.g. parallel composition) is an open problem also for standard CCS/CSP, obtaining an axiomatization for strongly guarded finite-state processes is the best that can be done without solving such an open problem. On the other hand in [6] we have already shown how to axiomatize unguarded recursion by means of the $pri(P)$ operator for interactive timed processes without static operators (the adaptation of the axiomatization of [6] to exponential delays is trivial), similarly to what Milner did for CCS in [16].

2 Calculus of Revisited Interactive Markov Chains

2.1 Revisited Interactive Markov Chains

In the following we present the formal definition of Interactive Markovian Transition System (IMTS). Interactive Markov Chains are IMTSs possessing

an initial state. Formally, we denote the set of rates by $Exp = \mathbb{R}^+$, ranged over by λ, μ. Moreover, we denote the set of standard action types used in a IMTS by Act, ranged over by $\alpha, \alpha', \ldots$. As usual Act includes the special type τ denoting internal actions. The set $Act - \{\tau\}$ is ranged over by $a, b, \ldots$. We use $\gamma, \gamma', \ldots$ to range over $Act \cup Exp$, i.e. labels of IMTS transitions. The set of states of an IMTS is denoted by Σ, ranged over by $s, s', \ldots$. In the rest of the paper we will assume the following abbreviations. Let us suppose that $T \subseteq (\Sigma \times Labels \times \Sigma)$ is a transition relation, where $Labels$ is a set of transition labels, ranged over by l. In the remainder we use $s \xrightarrow{l} s'$ to stand for $(s, l, s') \in T$; $s \xrightarrow{l}$ to stand for $\exists s' \in \Sigma : s \xrightarrow{l} s'$; and $s \xrightarrow{Set}$, where $Set \subseteq Labels$, to stand for $\exists s' \in \Sigma, l \in Set : s \xrightarrow{l} s'$. $s \not\xrightarrow{l}$ and $s \not\xrightarrow{Set}$, where $Set \subseteq Labels$, denote the negations of $s \xrightarrow{l}$ and $s \xrightarrow{Set}$, respectively.

Definition 2.1 An Interactive Markovian Transition System (IMTS) is a tuple (Σ, Act, T_e, T_a) with

- Σ a set of states,
- Act a set of standard actions,
- $T_e \subseteq (\Sigma \times Exp \times \Sigma)$ and $T_a \subseteq (\Sigma \times Act \times \Sigma)$ two transition relations, containing exponentially timed and action transitions, respectively, such that $\forall s \in \Sigma$:

$$s \xrightarrow{\tau} \text{ implies } s \not\xrightarrow{Exp}$$

■

2.2 Syntax and Semantics of the Calculus of RIMCs

Let Var be a set of process variables ranged over by X, Y, Z. Let $ARFun = \{\varphi : Act \longrightarrow Act \mid \varphi(\tau) = \tau \wedge \varphi(Act - \{\tau\}) \subseteq Act - \{\tau\}\}$ be a set of *action relabeling functions*, ranged over by φ.

Definition 2.2 We define the language $RIMC$ as the set of terms generated by the following syntax

$$P ::= \underline{1} \mid \underline{0} \mid X \mid \gamma; P \mid P \diamond P \mid P/L \mid P[\varphi] \mid P \parallel_S P \mid recX.P$$

where $L, S \subseteq Act - \{\tau\}$. A $RIMC$ process is a closed term of $RIMC$. We denote by $RIMC_c$ the set of $RIMC$ processes and by $RIMC_{cg}$ the set of strongly guarded $RIMC$ processes. [3] ■

"$\underline{1}$" denotes a terminated process which allows for the passage of time. "$\underline{0}$" denotes a time deadlock. "$\gamma; P$" is the prefix operator. Similarly as in [8], if γ is τ or a delay λ then it is immediately executed, otherwise (it is a visible action

[3] We consider λ prefixes as being guards in the definition of strong guardedness. Moreover we consider the notion of strong guardedness to account for relabeling and hiding operators: e.g. $(recX.a.X)/\{a\}$ is not strongly guarded (see e.g. [3] Appendix A for a precise definition).

$$\alpha; P \xrightarrow{\alpha} P$$

$$\frac{P \xrightarrow{\alpha} P'}{P \diamond Q \xrightarrow{\alpha} P'} \qquad \frac{Q \xrightarrow{\alpha} Q'}{P \diamond Q \xrightarrow{\alpha} Q'}$$

$$\frac{P \xrightarrow{\alpha} P'}{P \parallel_S Q \xrightarrow{\alpha} P' \parallel_S Q}\ \alpha \notin S \qquad \frac{Q \xrightarrow{\alpha} Q'}{P \parallel_S Q \xrightarrow{\alpha} P \parallel_S Q'}\ \alpha \notin S$$

$$\frac{P \xrightarrow{a} P' \wedge Q \xrightarrow{a} Q'}{P \parallel_S Q \xrightarrow{a} P' \parallel_S Q'}\ a \in S$$

$$\frac{P \xrightarrow{a} P'}{P/L \xrightarrow{\tau} P'/L}\ a \in L \qquad \frac{P \xrightarrow{\alpha} P'}{P/L \xrightarrow{\alpha} P'/L}\ a \notin L$$

$$\frac{P \xrightarrow{\alpha} P'}{P[\varphi] \xrightarrow{\varphi(\alpha)} P'[\varphi]} \qquad \frac{P\{recX.P/X\} \xrightarrow{\alpha} P'}{recX.P \xrightarrow{\alpha} P'}$$

Table 1
Standard Rules

a) it can be arbitrarily delayed. "$P \diamond Q$" is the choice operator. Similarly as in [8], as long as P or Q execute delays λ then they just evolve internally and the choice remains (as if they were in parallel). In particular time is allowed to advance for one process only if the same holds for the other one. The first between P and Q which executes an action α resolves the choice. "P/L" is the hiding operator which turns the actions in L into τ actions by consequently cutting alternative delay transitions, "$P[\varphi]$" is the relabeling operator which relabels visible actions according to φ. "$P \parallel_S Q$" is the CSP parallel operator, where synchronization over actions in S is required and where, similarly as in [8], time is allowed to advance for one process only if the same holds for the other one. Finally "$recX.P$" denotes recursion in the usual way.

The semantics of RIMC terms is defined as being the $RIMTS$ ($RIMC_c$, Act, T_e, T_a), where: T_a is the least subset of $RIMC_c \times Act \times RIMC_c$ satisfying the standard operational rules of Table 1 and T_e is obtained from the least multiset over $RIMC_c \times Exp \times RIMC_c$ satisfying the operational rules of Table 2 (similarly to [12,9], we consider a transition to have arity m if and only if it can be derived in m possible ways from the operational rules) by summing the rates of the multiple occurrences of the same transition. As already explained in Sect. 1.4, any value can be chosen for the rate $\tilde{\lambda}$ occurring in Table 2 (different values give rise to equivalent $RIMTSes$).

$$\lambda; P \xrightarrow{\lambda} P \qquad a; P \xrightarrow{\tilde{\lambda}} a; P \qquad \underline{1} \xrightarrow{\tilde{\lambda}} \underline{1}$$

$$\frac{P \xrightarrow{\lambda} P' \wedge Q \xrightarrow{Exp}}{P \diamond Q \xrightarrow{\lambda} P' \diamond Q} \qquad \frac{Q \xrightarrow{\lambda} Q' \wedge P \xrightarrow{Exp}}{P \diamond Q \xrightarrow{\lambda} P \diamond Q'}$$

$$\frac{P \xrightarrow{\lambda} P' \wedge Q \xrightarrow{Exp}}{P \parallel_S Q \xrightarrow{\lambda} P' \parallel_S Q} \qquad \frac{Q \xrightarrow{\lambda} Q' \wedge P \xrightarrow{Exp}}{P \parallel_S Q \xrightarrow{\lambda} P \parallel_S Q'}$$

$$\frac{P \xrightarrow{\lambda} P' \wedge \nexists a \in L. P \xrightarrow{a}}{P/L \xrightarrow{\lambda} P'/L} \qquad \frac{P \xrightarrow{\lambda} P'}{P[\varphi] \xrightarrow{\lambda} P'[\varphi]}$$

$$\frac{P\{recX.P/X\} \xrightarrow{\lambda} P'}{recX.P \xrightarrow{\lambda} P'}$$

Table 2
Rules for Exponentially Timed Moves

2.3 Observational Congruence for RIMCs

As explained in Sect. 1.4, the notion of observational congruence over RIMCs: (*i*) deals with exponentially timed choices according to a coarser variant of Markovian bisimulation [12] which abstracts from selfloops, and (*ii*) abstracts from standard τ actions as in observational congruence [14].

Given a RIMTS (Σ, Act, T_e, T_a), a state $s \in \Sigma$ and a set of states $C \subseteq \Sigma$, in the following we denote the total rate of exponentially timed transitions from s to C by $TR(s, C) = \sum\{\!|\, \lambda \mid \exists s' \in C : s \xrightarrow{\lambda} s' \,|\!\}$. [4] Moreover we use $\stackrel{\alpha}{\Longrightarrow}$ to denote $(\xrightarrow{\tau})^* \xrightarrow{\alpha} (\xrightarrow{\tau})^*$, i.e. a sequence of transitions including a single α transition and any number of τ transitions. We also define $\stackrel{\hat{\alpha}}{\Longrightarrow} = \stackrel{\alpha}{\Longrightarrow}$ if $\alpha \neq \tau$ and $\stackrel{\hat{\tau}}{\Longrightarrow} = (\xrightarrow{\tau})^*$, i.e. a possibly empty sequence of τ transitions.

Definition 2.3 Let (Σ, Act, T_e, T_a) be a RIMTS. An equivalence relation β on Σ is a *weak bisimulation* iff $s_1\ \beta\ s_2$ implies:

(i) for every $\alpha \in Act$ and $s_1' \in \Sigma$,
$$s_1 \xrightarrow{\alpha} s_1' \text{ implies } s_2 \stackrel{\hat{\alpha}}{\Longrightarrow} s_2' \text{ for some } s_2' \text{ with } s_1'\ \beta\ s_2'$$

(ii) $s_1 \xrightarrow{Exp}$ implies: $s_2 \stackrel{\hat{\tau}}{\Longrightarrow} s_2'$ for some s_2' such that $s_2' \xrightarrow{Exp}$ and for every $C \in \Sigma/\beta$ with $C \neq [s_1]_\beta$,
$$TR(s_1, C) = TR(s_2', C)$$ [5]

[4] We use "{|" and "|}" as brackets for multisets. Moreover we assume summation over the empty multiset to yield 0.

[5] We use "Σ/β" to denote the set of the equivalence classes of β defined over Σ.

$s_1, s_2 \in \Sigma$ are weakly bisimilar, denoted by $s_1 \approx s_2$, iff (s_1, s_2) is included in some weak bisimulation. ■

Note that for a state s_2' satisfying condition (ii) it must be that $s_1 \ \beta \ s_2'$ (otherwise it would not be possible that $s_1 \ \beta \ s_2$ since $s_1 \xrightarrow{Exp}$ implies that $s_1 \stackrel{\tau}{\not\longrightarrow}$), hence $[s_1]_\beta = [s_2']_\beta$, i.e. for both s_1 and s_2' we do not consider exponential transitions leading to their own equivalence class. Moreover note that, as shown in [11], trying to "weaken" any further the notion of weak bisimulation above, e.g. by allowing τ transitions to be executed after exponential delays to reach an equivalence class, does not generate a coarser notion of equivalence.

Definition 2.4 Let (Σ, Act, T_e, T_a) be a RIMTS. An equivalence relation β on Σ is an *observational bisimulation* iff $s_1 \ \beta \ s_2$ implies:

(i) for every $\alpha \in Act$ and $s_1' \in \Sigma$,
$$s_1 \xrightarrow{\alpha} s_1' \text{ implies } s_2 \stackrel{\alpha}{\Longrightarrow} s_2' \text{ for some } s_2' \text{ with } s_1' \approx s_2'$$

(ii) $s_1 \xrightarrow{Exp}$ implies: $s_2 \xrightarrow{Exp}$ and
$$\text{for every } C \in \Sigma/\beta \text{ with } C \neq [s_1]_\beta,\quad TR(s_1, C) = TR(s_2, C)$$

$s_1, s_2 \in \Sigma$ are observationally congruent, denoted by $s_1 \simeq s_2$, iff (s_1, s_2) is included in some observational bisimulation. ■

Note that, since $[s_1]_\beta = [s_2]_\beta$, again, in condition (ii) for both s_1 and s_2 we do not consider exponential transitions leading to their own equivalence class.

We consider $\simeq$ as being defined also on the open terms of *RIMC* by extending observational congruence with the standard approach of [14].

Theorem 2.5 *$\simeq$ is a congruence for the calculus of RIMCs w.r.t. all its operators, including recursion.*

Proof. *Let us start from the choice operator "$P \diamond Q$". It suffices to show that $\beta = \{(P_1 \diamond Q, P_2 \diamond Q) \mid P_1, P_2, Q \in RIMC_c \wedge P_1 \simeq P_2\} \cup ID_{RIMC_c}$ (where ID_{RIMC_c} is the identity relation over $RIMC_c$) is an observational bisimulation. Given $(R_1, R_2) \in \beta$, either $(R_1, R_2) \in ID_{RIMC_c}$ and the proof is trivial, or $R_1 \equiv P_1 \diamond Q$ and $R_2 \equiv P_1 \diamond Q$ for some P_1, P_2 and Q. In the latter case:*

- *If $P_1 \diamond Q$ perfoms a standard action α then $P_2 \diamond Q$ may perform a corresponding move by resorting to standard machinery [14].*
- *If $P_1 \diamond Q \xrightarrow{Exp}$ then $P_1 \xrightarrow{Exp}$ and $Q \xrightarrow{Exp}$. Since $P_1 \simeq P_2$, we have $P_2 \xrightarrow{Exp}$ and for every $C \in RIMC_c/\simeq$ with $C \neq [P_1]_\simeq$, $TR(P_1, C) = TR(P_2, C)$. Therefore $P_2 \diamond Q \xrightarrow{Exp}$ and for every $C \in RIMC_c/\beta$ with $C \neq [P_1 \diamond Q]_\beta$, we have:*
  - *either $C = \{R\}$ for some term R whose outermost operator is not "$\diamond$" and $TR(P_1 \diamond Q, C) = TR(P_2 \diamond Q, C) = 0$,*

· *or there exists* $C' \in RIMC_c/\simeq$ *and* $Q' \in RIMC_c$ *such that* $C = \{P \diamond Q' \mid P \in C'\}$. *In this case:*
if $C' \neq [P_1]_\simeq$ *and* $Q = Q'$ *then* $TR(P_1 \diamond Q, C) = TR(P_1, C') = TR(P_2, C') = TR(P_2 \diamond Q, C)$;
if $C' = [P_1]_\simeq$ *and* $Q \neq Q'$ *then* $TR(P_1 \diamond Q, C) = TR(Q, \{Q'\}) = TR(P_2 \diamond Q, C)$;
if $C' \neq [P_1]_\simeq$ *and* $Q \neq Q'$ *then* $TR(P_1 \diamond Q, C) = 0 = TR(P_2 \diamond Q, C)$;

As far as the parallel operator "$P \parallel_S Q$*" is concerned, we preliminarily show that "*$P \parallel_S Q$*" is a congruence w.r.t. weak bisimulation, i.e. that, for a given set* S, $\beta = \{(P_1 \parallel_S Q, P_2 \parallel_S Q) \mid P_1, P_2, Q \in RIMC_c \wedge P_1 \simeq P_2\} \cup ID_{RIMC_c}$ *is a weak bisimulation. Given* $(R_1, R_2) \in \beta$, *either* $(R_1, R_2) \in ID_{RIMC_c}$ *and the proof is trivial, or* $R_1 \equiv P_1 \parallel_S Q$ *and* $R_2 \equiv P_1 \parallel_S Q$ *for some* P_1, P_2 *and* Q. *In the latter case:*

- *If* $P_1 \parallel_S Q$ *perfoms a standard action* α *then* $P_2 \parallel_S Q$ *may perform a corresponding move by resorting to standard machinery [14].*
- *If* $P_1 \parallel_S Q \xrightarrow{Exp}$ *then* $P_1 \xrightarrow{Exp}$ *and* $Q \xrightarrow{Exp}$. *Since* $P_1 \approx P_2$, *we have* $P_2 \stackrel{\hat{\tau}}{\Longrightarrow} P_2'$ *and* $P_2' \xrightarrow{Exp}$ *and for every* $C \in RIMC_c/\approx$ *with* $C \neq [P_1]_\approx$, $TR(P_1, C) = TR(P_2, C)$. *Therefore* $P_2 \parallel_S Q \stackrel{\hat{\tau}}{\Longrightarrow} P_2' \parallel_S Q$ *and* $P_2' \parallel_S Q \xrightarrow{Exp}$ *and for every* $C \in RIMC_c/\beta$ *with* $C \neq [P_1 \parallel_S Q]_\beta$, *we have:*
  · *either* $C = \{R\}$ *for some term* R *whose outermost operator is not "*$\parallel_S$*" and* $TR(P_1 \parallel_S Q, C) = TR(P_2 \parallel_S Q, C) = 0$,
  · *or there exists* $C' \in RIMC_c/\approx$ *and* $Q' \in RIMC_c$ *such that* $C = \{P \parallel_S Q' \mid P \in C'\}$. *In this case:*
  if $C' \neq [P_1]_\approx$ *and* $Q = Q'$ *then* $TR(P_1 \parallel_S Q, C) = TR(P_1, C') = TR(P_2', C') = TR(P_2' \parallel_S Q, C)$;
  if $C' = [P_1]_\approx$ *and* $Q \neq Q'$ *then* $TR(P_1 \parallel_S Q, C) = TR(Q, \{Q'\}) = TR(P_2' \parallel_S Q, C)$;
  if $C' \neq [P_1]_\approx$ *and* $Q \neq Q'$ *then* $TR(P_1 \parallel_S Q, C) = 0 = TR(P_2' \parallel_S Q, C)$;

Now it suffices to show that, for a given set S, $\beta = \{(P_1 \parallel_S Q, P_2 \parallel_S Q) \mid P_1, P_2, Q \in RIMC_c \wedge P_1 \simeq P_2\} \cup ID_{RIMC_c}$ *is an observational bisimulation. The proof of this fact is identical to the proof above for weak bisimulation (with "*$\simeq$*" replacing "*$\approx$*"), apart from the case of a standard action* α *performed by* $P_1 \parallel_S Q$. *In paricular, we derive* $P_1' \parallel_S Q \approx P_2' \parallel_S Q$, *where* P_1' *and* P_2' *are the terms reached by* P_1 *and* P_2 *respectively, from* $P_1' \approx P_2'$ *by exploiting the result above about congruence of weak bisimulation w.r.t. parallel.*

As far as the hiding operator "P/L*" is concerned, we preliminarily show that "*P/L*" is a congruence w.r.t. weak bisimulation, i.e. that, for a given set* L, $\beta = \{(P_1/L, P_2/L) \mid P_1, P_2 \in RIMC_c \wedge P_1 \simeq P_2\} \cup ID_{RIMC_c}$ *is a weak bisimulation. Given* $(R_1, R_2) \in \beta$, *either* $(R_1, R_2) \in ID_{RIMC_c}$ *and the proof is trivial, or* $R_1 \equiv P_1/L$ *and* $R_2 \equiv P_1/L$ *for some* P_1, P_2. *In the latter case:*

- *If* P_1/L *perfoms a standard action* α *then* P_2/L *may perform a corresponding move by resorting to standard machinery [14].*

- *If* $P_1/L \xrightarrow{Exp}$ *then* $P_1 \xrightarrow{Exp}$ *and* $\nexists a \in L : P_1 \xrightarrow{a}$. *Since* $P_1 \approx P_2$, *we have* $P_2 \stackrel{\hat{\tau}}{\Longrightarrow} P_2'$ *and* $P_2' \xrightarrow{Exp}$ *and for every* $C \in RIMC_c/\approx$ *with* $C \neq [P_1]_\approx$, $TR(P_1, C) = TR(P_2, C)$. *Therefore* $P_2/L \stackrel{\hat{\tau}}{\Longrightarrow} P_2'/L$ *and (since* $P_2' \approx P_1$, *hence* $\nexists a \in L : P_2' \xrightarrow{a}$*)* $P_2'/L \xrightarrow{Exp}$ *and for every* $C \in RIMC_c/\beta$ *with* $C \neq [P_1/L]_\beta$, *we have:*
  - *either* $C = \{R\}$ *for some term* R *whose outermost operator is not "*$/L$*" and* $TR(P_1/L, C) = TR(P_2/L, C) = 0$,
  - *or there exists* $C' \in RIMC_c/\approx$, *with* $C' \neq [P_1]_\approx$, *such that* $C = \{P/L \mid P \in C'\}$. *In this case* $TR(P_1/L, C) = TR(P_1, C') = TR(P_2', C') = TR(P_2'/L, C)$.

Now it suffices to show that, for a given set L, $\beta = \{(P_1/L, P_2/L) \mid P_1, P_2 \in RIMC_c \wedge P_1 \simeq P_2\} \cup ID_{RIMC_c}$ *is an observational bisimulation. The proof of this fact is identical to the proof above for weak bisimulation (with "*$\simeq$*" replacing "*$\approx$*"), apart from the case of a standard action* α *performed by* P_1/L. *In paricular, we derive* $P_1'/L \approx P_2'/L$, *where* P_1' *and* P_2' *are the terms reached by* P_1 *and* P_2 *respectively, from* $P_1' \approx P_2'$ *by exploiting the result above about congruence of weak bisimulation w.r.t. hiding.*

The proof of congruence w.r.t. prefix "$\gamma; P$*" and relabeling "*$P[\varphi]$*" is trivial.*

As far as recursion "$recX.P$*" is concerned, we apply the technique we introduced in [4]. We have to show that, for all* $P_1, P_2 \in RIMC_c$ *containing at most the variable* X *free, we have that* $P_1 \simeq P_2$ *implies* $recX.P_1 \simeq recX.P_2$. *We do this by showing that the relation* $\beta = \{(Q\{recX.P_1/X\}, Q\{recX.P_2/X\}) \mid Q \in RIMC_c\}$ *is such that, given* $\beta' = \beta \cup \beta^{-1}$, *whenever* $R_1 \; \beta \; R_2$ *we have:*

(i) *for every* $\alpha \in Act$ *and* $R_1' \in RIMC_c$,
$$R_1 \xrightarrow{\alpha} R_1' \text{ implies } R_2 \stackrel{\alpha}{\Longrightarrow} R_2' \text{ for some } R_2' \text{ with } R_1' \approx \cup\beta \; R_2'$$

(ii) $R_1 \xrightarrow{Exp}$ *implies:* $R_2 \xrightarrow{Exp}$ *and*
$$\text{for every } C \in RIMC_c/(\simeq \cup\beta')^+ \text{ with } C \neq [R_1]_{(\simeq\cup\beta')^+}, \; TR(R_1, C) = TR(R_2, C)$$

In particular we induce on the maximum depth of the inference of the transitions leaving term R_1 *and we show* $TR(R_1, C) \leq TR(R_2, C)$ *only. The converse is obtained by a symmetrical argument on the moves of* R_2. *In such an induction, the only significant novelty w.r.t. the proof of [4] is the exclusion of selfloops when evaluating total rates. However such an exclusion is "compatible" with the proof because when an equivalence class* C *considered at maximum depth* d *is expressed in terms of the corresponding ones* $C_i, i \in I$ *considered at maximum depth* $d - 1$, *we have that if* C *does not constitute a selfloop none of the classes* $C_i, i \in I$ *constitutes a selfloop. Intuitively a recursion* $recX.P$ *cannot unfold a selfloop already present in* P *(thus making the total rate of the selfloop "observable"). Note that from the statement above it is immediate to conclude that* $(\approx \cup\beta')^+$ *is a weak bisimulation and, then, that* $(\simeq \cup\beta')^+$ *is an observational bisimulation. Therefore, by taking* $Q \equiv X$ *in* β,

we are done. □

3 Axiomatizing Revisited Interactive Markov Chains

In this section we present an axiom system which is complete for $\simeq$ on strongly guarded finite-state *RIMC* processes.

In order to build the axiomatization we need to extend RIMTSes and our notion of observational equivalence with the "observable" exponential delays of [9] and to introduce some auxiliary operators. Formally, we denote the set of rates of observable delays by $Exp^o = \mathbb{R}^+$, ranged over by $\lambda^o, \mu^o, \ldots$. We use $\theta, \theta', \ldots$ to range over $Act \cup Exp \cup Exp^o$. Moreover we use $\omega, \omega', \ldots$ to range over $(Act \cup Exp \cup Exp^o)^+$, i.e. non-empty finite sequences over $Act \cup Exp \cup Exp^o$, and $\rho, \rho', \ldots$ to range over $(Exp)^+$.

Definition 3.1 An Extended Interactive Markovian Transition System (EIMTS) is a tuple $(\Sigma, Act, T_o, T_e, T_a)$ with

- Σ a set of states,
- Act a set of standard actions,
- $T_o \subseteq (\Sigma \times Exp^o \times \Sigma)$, $T_e \subseteq (\Sigma \times Exp \times \Sigma)$, and $T_a \subseteq (\Sigma \times Act \times \Sigma)$ three transition relations, containing observable exponentially timed, unobservable exponentially timed, and action transitions, respectively, such that $\forall\, s \in \Sigma$:

(i) $s \xrightarrow{Exp^o}$ implies $s \xrightarrow{Exp} \not\rightarrow$ (or equivalently $s \xrightarrow{Exp}$ implies $s \xrightarrow{Exp^o} \not\rightarrow$)

(ii) $s \xrightarrow{\tau}$ implies: $s \xrightarrow{Exp^o} \not\rightarrow$ and $s \xrightarrow{Exp} \not\rightarrow$ ■

Now we formally introduce the auxiliary operators needed to build the axiomatization, whose semantics is presented in Table 3. The operators "$\omega.P$" and "$P + Q$" are those of the calculus of IMCs [9] (apart from extension of prefix to sequences); in particular "$P + Q$" works on observable delays only. The new operator "$\mathcal{H}(P)$", which "hides" exponential delays by turning "observable" λ^o immediately executable by P into "unobservable" λ (and "restricts" unobservable delays previously executable by P), will play a fundamental role in the axiomatization. In particular it will allow us to express the operators of the calculus of RIMCs in terms of the prefix and choice operators of the calculus of IMCs [9]. The operators "$P \,|\!|_S\, Q$" and "$P \,|_S\, Q$" are simple variants of the left merge and synchronization merge operators of [1], while "$P \triangleleft Q$" is a sort of left merge operator used for axiomatizing choice "$P \diamond Q$". Note that "$P \,|\!|_S\, Q$" and "$P \triangleleft Q$" are defined in such a way that: (i) since they have to be used as arguments of a "$P + Q$" operator, they require delays immediately executable (by P) to be observable; and (ii) they can execute exponential delays of P also in the case Q can execute τ transitions (and, e.g., not delay transitions), so that the axioms $(LC3)$ and $(LM4)$ of Table 1 are sound. Moreover, the definition of the operational rule for "$P \,|_S\, Q$" allows for actions "τ" to be skipped so to get a congruence [1]. Finally, the operator "$pri(P)$",

$$(\theta\omega).P \xrightarrow{\theta} \omega.P \qquad \theta.P \xrightarrow{\theta} P$$

$$\frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'} \qquad \frac{Q \xrightarrow{\alpha} Q'}{P + Q \xrightarrow{\alpha} Q'}$$

$$\frac{P \xrightarrow{\lambda^o} P' \wedge Q \not\xrightarrow{\tau}}{P + Q \xrightarrow{\lambda^o} P'} \qquad \frac{Q \xrightarrow{\lambda^o} Q' \wedge P \not\xrightarrow{\tau}}{P + Q \xrightarrow{\lambda^o} Q'}$$

$$\frac{P \xrightarrow{\alpha} P'}{\mathcal{H}(P) \xrightarrow{\alpha} P'} \qquad \frac{P \xrightarrow{\lambda^o} P'}{\mathcal{H}(P) \xrightarrow{\lambda} P'}$$

$$\frac{P \xrightarrow{\alpha} P'}{P \triangleleft Q \xrightarrow{\alpha} P'} \qquad \frac{P \xrightarrow{\lambda^o} P' \wedge (Q \xrightarrow{Exp} \vee\, Q \xrightarrow{\tau})}{P \triangleleft Q \xrightarrow{\lambda^o} P' \diamond Q}$$

$$\frac{P \xrightarrow{\alpha} P'}{P \,\|_S\, Q \xrightarrow{\alpha} P' \,\|_S\, Q}\ \alpha \notin S \qquad \frac{P \xrightarrow{\lambda^o} P' \wedge (Q \xrightarrow{Exp} \vee\, Q \xrightarrow{\tau})}{P \,\|_S\, Q \xrightarrow{\lambda^o} P' \,\|_S\, Q}$$

$$\frac{P \stackrel{a}{\Longrightarrow} P' \wedge Q \stackrel{a}{\Longrightarrow} Q'}{P \,|_S\, Q \xrightarrow{a} P' \,\|_S\, Q'}\ a \in S \qquad \frac{P \xrightarrow{\alpha} P'}{pri(P) \xrightarrow{\alpha} P'}$$

Table 3
Rules for Auxiliary Operators

which we introduced in [6] for axiomatizing unguarded recursion, eliminates non-prioritized behaviors (those immediately starting in P with an observable or unobservable exponential delay). Such an operator will play a important role in the axiomatization of parallel composition and choice "$P \diamond Q$" in that it allows us to check for the absence of executable exponential delays (see axioms $(LC4)$ and $(LM5)$ of Table 1).

We define the language EIMC to be the set of terms obtained by extending the calculus of RIMCs with the auxiliary operators above (we denote the set of closed EIMC terms by EIMC$_c$). Moreover we define the semantics of EIMC terms to be the EIMTS $(EIMC_c, Act, T_o, T_e, T_a)$ obtained from the operational rules of Table 1, Table 2 and Table 3 plus an additional rule for both the hiding "P/L" and the relabeling "$P[\varphi]$" operators which is obtained from that of Table 2 by replacing λ^o transitions for λ transitions. Note that "P/L" and "$P[\varphi]$" are conservatively extended.

The notions of weak bisimulation and observational congruence for EIMC are conservative extensions of those for RIMC. In the following, we denote the total rate of observable exponentially timed transitions from s to I by $TR^o(s, I)$, which is defined similarly as for "unobservable" delays.

Definition 3.2 Let $(\Sigma, Act, T_o, T_e, T_a)$ be an EIMTS. An equivalence relation β on Σ is a *weak bisimulation* iff $s_1\ \beta\ s_2$ implies: the 2 conditions of Definition 2.3 and the additional condition

(iii) $s_1 \xrightarrow{Exp^o}$ implies: $s_2 \stackrel{\hat{\tau}}{\Longrightarrow} s_2'$ for some s_2' such that $s_2' \xrightarrow{Exp^o}$ and
for every $C \in \Sigma/\beta$, $TR^o(s_1, C) = TR^o(s_2', C)$

$s_1, s_2 \in \Sigma$ are weakly bisimilar, denoted by $s_1 \approx s_2$, iff (s_1, s_2) is included in some weak bisimulation. ■

Definition 3.3 Let $(\Sigma, Act, T_o, T_e, T_a)$ be an EIMTS. An equivalence relation β on Σ is an *observational bisimulation* iff $s_1\ \beta\ s_2$ implies: the 2 conditions of Definition 2.4 and the additional condition

(iii) $s_1 \xrightarrow{Exp^o}$ implies: $s_2 \xrightarrow{Exp^o}$ and
for every $C \in \Sigma/\beta$, $TR^o(s_1, C) = TR^o(s_2, C)$

$s_1, s_2 \in \Sigma$ are observationally congruent, denoted by $s_1 \simeq s_2$, iff (s_1, s_2) is included in some observational bisimulation. ■

The extension of the calculus of *RIMCs* preserves the congruence property.

Theorem 3.4 *$\simeq$ is a congruence for the calculus of EIMCs w.r.t. all its operators, including recursion.*

Proof. *Given $P \simeq Q$, for each operator "op" it just sufficies to show that the (symmetric and transitive closure of the) relation obtained by adding $(op(P), op(Q))$ to $\simeq$ is an observational bisimulation, by exploiting the congruence property of observational congruence w.r.t. both parallel "$\|_S$" and choice "$\diamond$", and the congruence property of weak bisimulation w.r.t. parallel "$\|_S$". Note that for obtaining congruence w.r.t. the operator "$\triangleleft$" it is essential that in item 3 of Definition 3.3 we consider equivalence classes w.r.t. relation β instead of just considering relation $\approx$.* □

We are now in a position to present the axiom system $\mathcal{A}_{EIMC}$ for $\simeq$ on *EIMC* terms, which is formed by the axioms presented in Fig. 1. The axioms $(Ter) - (SM6)$, with the help of axioms $(Pri1), (Pri2), (A4)$ and $(Rec1) - (Rec3)$, are used to transform $RIMC_{cg}$ processes into normal form.

Definition 3.5 A process $P \in RIMC_c$ is in normal form if it is either of the form "$\mathcal{H}(\sum_{i \in I} \theta_i.P_i)$" [6] or "$recX.\mathcal{H}(\sum_{i \in I} \theta_i.P_i)$" or "$X$", where:

- for each $i \in I$, $\theta_i \in Exp^o \cup Act$,
- if there exists $i \in I$ such that $\theta_i = \tau$ then there is no $i \in I$ such that $\theta_i \in Exp^o$,
- for each $i \in I$, P_i is again in normal form and satisfies the following condition: if $\theta_i \in Exp^o$ then, supposing that the transitions leaving P_i and corresponding derivative terms are described in the normal form by means of θ_j' and P_j', with $j \in I'$, we must have that $\{(\theta_j, P_j) \mid j \in I - \{i\}\} \subseteq \{(\theta_j', P_j') \mid j \in I'\}$ (i.e. exponential delay transitions preserve alternative behaviors). ■

[6] We assume $\sum_{i \in I} \theta_i.P_i$ to be "$\underline{0}$" when $I = \emptyset$.

$(A1)$	$P + Q = Q + P$	$(A2)$	$(P + Q) + R = P + (Q + R)$
$(A3)$	$\alpha.P + \alpha.P = \alpha.P$	$(A4)$	$\mathcal{H}(P + \underline{0}) = \mathcal{H}(P)$
(Seq)	$\theta.\omega.P = (\theta\omega).P$		
$(Tau1)$	$(\alpha\rho).\mathcal{H}(\tau.P) = (\alpha\rho).P$		
$(Tau2)$	$\mathcal{H}(P + \tau.\mathcal{H}(P) + Q) = \mathcal{H}(\tau.\mathcal{H}(P) + Q)$		
$(Tau3)$	$\alpha.\mathcal{H}(P + \tau.Q) + \alpha.Q = \alpha.\mathcal{H}(P + \tau.Q)$		
$(ExpT1)$	$\lambda^o.P + \mu^o.P = (\lambda^o + \mu^o).P$	$(ExpT2)$	$\mathcal{H}(\lambda^o.P) = \lambda.P$
$(MProg)$	$\tau.P + \lambda^o.Q = \tau.P$		
$(Pri1)$	$pri(\alpha.P) = \alpha.P$	$(Pri2)$	$pri(P + Q) = pri(P) + pri(Q)$
(Ter)	$\underline{1} = recX.\mathcal{H}(\lambda^o.X)$		
$(Dead)$	$\underline{0} = \mathcal{H}(\underline{0})$		
$(Pre1)$	$a; P = recX.\mathcal{H}(a.P + \lambda^o.X)$	$(Pre2)$	$\tau; P = \mathcal{H}(\tau.P)$
$(Pre3)$	$\lambda; P = \mathcal{H}(\lambda^o.P)$		
$(Hi1)$	$\mathcal{H}(P)/L = \mathcal{H}(P/L)$	$(Hi2)$	$\underline{0}/L = \underline{0}$
$(Hi3)$	$(\theta.P)/L = \theta.(P/L) \quad \theta \notin L$	$(Hi4)$	$(a.P)/L = \tau.(P/L) \quad a \in L$
$(Hi5)$	$(P + Q)/L = P/L + Q/L$		
$(Rel1)$	$\mathcal{H}(P)[\varphi] = \mathcal{H}(P[\varphi])$	$(Rel2)$	$\underline{0}[\varphi] = \underline{0}$
$(Rel3)$	$(\alpha.P)[\varphi] = \varphi(\alpha).(P[\varphi])$	$(Rel4)$	$(\theta.P)[\varphi] = \theta.(P[\varphi])$
$(Rel5)$	$(P + Q)[\varphi] = P[\varphi] + Q[\varphi]$		
(Ch)	$\mathcal{H}(P) \diamond \mathcal{H}(Q) = \mathcal{H}(P \triangleleft \mathcal{H}(Q) + Q \triangleleft \mathcal{H}(P))$		
$(LC1)$	$\underline{0} \triangleleft P = \underline{0}$		
$(LC2)$	$(\alpha.P) \triangleleft Q = \alpha.P$		
$(LC3)$	$(\lambda^o.P) \triangleleft \mathcal{H}(\mu^o.Q + R) = \lambda^o.(P \diamond \mathcal{H}(\mu^o.Q + R))$		
$(LC4)$	$(\lambda^o.P) \triangleleft \mathcal{H}(pri(Q)) = \underline{0}$		
$(LC5)$	$(P + Q) \triangleleft R = P \triangleleft R + Q \triangleleft R$		
(Par)	$\mathcal{H}(P) \parallel_S \mathcal{H}(Q) = \mathcal{H}(P \between_S \mathcal{H}(Q) + Q \between_S \mathcal{H}(P) + P \mid_S Q)$		
$(LM1)$	$\underline{0} \between_S P = \underline{0}$		
$(LM2)$	$(a.P) \between_S Q = \underline{0}$	$a \in S$	
$(LM3)$	$(\alpha.P) \between_S Q = \alpha.(P \parallel_S Q)$	$\alpha \notin S$	
$(LM4)$	$(\lambda^o.P) \between_S \mathcal{H}(\mu^o.Q + R) = \lambda^o.(P \parallel_S \mathcal{H}(\mu^o.Q + R))$		
$(LM5)$	$(\lambda^o.P) \between_S \mathcal{H}(pri(Q)) = \underline{0}$		
$(LM6)$	$(P + Q) \between_S R = P \between_S R + Q \between_S R$		
$(SM1)$	$P \mid_S Q = Q \mid_S P$		
$(SM2)$	$\underline{0} \mid_S P = \underline{0}$		
$(SM3)$	$(\theta.P) \mid_S (\theta'.Q) = \underline{0}$	$(\theta \notin S \ \vee \ \theta \neq \theta') \wedge \tau \notin \{\theta, \theta'\}$	
$(SM4)$	$(\tau.P) \mid_S Q = P \mid_S Q$		
$(SM5)$	$(a.P) \mid_S (a.Q) = a.(P \parallel_S Q)$	$a \in S$	
$(SM6)$	$(P + Q) \mid_S R = P \mid_S R + Q \mid_S R$		
$(Rec1)$	$recX.P = recY.(P\{Y/X\})$	provided that Y is not free in $recX.P$	
$(Rec2)$	$recX.P = P\{recX.P/X\}$		
$(Rec3)$	$P = Q\{P/X\}$ implies $P = recX.Q$	if X is serial and strongly guarded in Q	
$(ExpRec)$	$recX.\mathcal{H}(\lambda^o.X + \mu^o.P + Q) = recX.\mathcal{H}(\mu^o.P + Q)$		

Fig. 1. Axiomatization for RIMC

The standard axioms $(A1)-(A4), (Tau1)-(Tau3)$ and $(Rec1)-(Rec3)$ (the slight variation of the axiom $(Tau1)$ w.r.t. the standard one reflects the fact that our notion of observational congruence requires an action transition, as opposed to a delay transition, to be performed before weak bisimulation is considered) plus the axiom (Seq), which allows a sequence of prefixes to be "merged" into a single prefix so that $(Tau1)$ can be applied, the axiom $(ExpT1)$, which captures additivity of exponential delays, the axiom $(ExpT2)$, which allows "$\mathcal{H}(\lambda^o.P)$" states to be expressed by "$\lambda.P$" so that axiom $(Tau1)$ can be applied, the axiom $(MProg)$, which captures the maximal progress assumption, and the totally new axiom $(ExpRec)$ which captures the insensitivity to selfloops of exponential delays, are used to equate normal forms which are equivalent according to $\simeq$. In particular note that axiom $(Tau1)$ is sufficient to get completeness over normal forms because delay transitions preserve alternative behaviors. Concerning axiom $(Rec3)$, we define X to be serial in a term if each free occurrence of X in that term is in the scope of $\theta.P$, $P+Q$, $\mathcal{H}(P)$ and $recX.P$ only. Moreover we assume the standard definition of [14] for strong guardedness of serial variables (λ and λ^o prefixes are considered as being guards) and of terms in normal form.

Theorem 3.6 *The axioms of $\mathcal{A}_{EIMC}$ are sound for $\simeq$ over EIMC terms.*

Proof. *For each pair of equated terms it is sufficient to show that there exists an observational bisimulation which includes such a pair.* □

Lemma 3.7 *If a process $P \in RIMC_{cg}$ is finite state, then $\exists P' \in EIMC_c$: $\mathcal{A}_{EIMC} \vdash P = P'$ with P' strongly guarded term in normal form.*

Proof. *Let $P_1 \ldots P_n$ be the states of the RIMC derived from the semantics of P, $P_n \equiv P$. Since P is strongly guarded, each state P_i of the semantics of P is finitely branching. It can be easily seen that (thanks to an inductive usage of axioms $(Ter)-(SM6)$, with the help of axioms $(Pri1), (Pri2), (A4)$ and $(Rec1)-(Rec2)$, on the syntactic structure of states) for each $i \in \{1 \ldots n\}$, there exist $m_i \in \mathbb{N}$, $\{\theta^i_j\}_{j \leq m_i}$, $\{k^i_j\}_{j \leq m_i}$ s.t. $\mathcal{A}_{EIMC} \vdash P_i = \mathcal{H}(\sum_{j \leq m_i} \theta^i_j.P_{k^i_j})$ where:*

- *for each $j \leq m_i$, $\theta^i_j \in Exp^o \cup Act$,*
- *if there exists $j \leq m_i$ such that $\theta^i_j = \tau$ then there is no $j \leq m_i$ such that $\theta^i_j \in Exp^o$,*
- *for each $j \leq m_i$, $P_{k^i_j}$ satisfies the following condition: if $\theta^i_j \in Exp^o$ then, $\{(\theta^i_{j'}, P_{k^i_{j'}}) \mid j' \leq m_i \wedge j' \neq j\} \subseteq \{(\theta^{k^i_j}_{j'}, P_{k^{k^i_j}_{j'}}) \mid j' \leq m_{k^i_j}\}$ (i.e. exponential delay transitions preserve alternative behaviors).*

Hence we can characterize the behavior of P by means of a set of equations similarly to [15]. Moreover, similarly to the unique solution of equations theorem of [15], we have that there is a (strongly guarded) term P in normal form such that $\mathcal{A}_{EIMC} \vdash P = P_n \equiv P$. This can be shown as follows. For each i,

from 1 *to* n*, we do the following. If* i *is such that* $\exists j \leq m_i : k_j^i = i$ *we have, by applying* $(Rec3)$*, that* $P_i = recX.\mathcal{H}(\sum_{j \leq m_i : k_j^i \neq i} \theta_j^i.P_{k_j^i} + \sum_{j \leq m_i : k_j^i = i} \theta_j^i.X)$*. Then we replace each subterm* P_i *occurring in the equations for* $P_{i+1} \ldots P_n$ *with its equivalent term. When, in the equation for* $P_n \equiv P$*, we have replaced* P_{n-1}*, we are done.* □

Lemma 3.8 *If* $P, Q \in EIMC_c$ *are strongly guarded terms in normal form such that* $P \simeq Q$ *then* $\mathcal{A}_{EIMC} \vdash P = Q$

Proof. *The proof is carried out similary to [6] and [9] by using the standard technique based on "guarded equation sets" [16]. In particular, when applying such a technique, we take "standard guarded equation sets" to be guarded equation sets whose structure follows exactly our definition of normal forms for terms. Given that, it is quite simple to verify that each strongly guarded term in normal form satisfies some standard guarded equation set and that, by using axioms* $(A1) - (A4)$*,* (Seq)*,* $(Tau1) - (Tau3)$*,* $(ExpT1), (ExpT2)$*,* $(MProg)$*,* $(Rec1) - (Rec3)$ *and* $(ExpRec)$*, it is possible to build a common standard guarded equation set which is satisfied by both* P *and* Q*, thus obtaining* $\mathcal{A}_{EIMC} \vdash P = Q$ *(see the explanation above of the role of these axioms in proving equality of equivalent normal forms).* □

Theorem 3.9 $\mathcal{A}_{EIMC}$ *is complete for* $\simeq$ *over finite state processes of* $RIMC_{cg}$*.*

Proof. *A direct consequence of Lemmas 3.7 and 3.8* □

4 Conclusion

We would like to observe that our τ-divergence insensitive notion of observational congruence (simplified so that "ticks" replace exponential delays) is a congruence also for the timed algebra of [8] and in this context a much simpler and suitably varied version of the axiom system that we have presented (where the operator "$\mathcal{H}(P)$" is not used and delays synchronize instead of being interleaved) can be used to obtain an axiomatization that is complete over strongly guarded finite-state processes. Moreover we believe that the same "transformation" we performed on the calculus of IMC [9] can be applied also to other interactive timed calculi, as, e.g., the calculus of IWMC (see [3] Chapter 4) and the calculus of IGSMP (see [5] or [3] Chapters 6 and 7) which are basically extensions of the calculus of IMC [9] with probabilistic choices and generally distributed delays.

References

[1] L. Aceto, *"On "Axiomatising Finite Concurrent Processes" "* in SIAM Journal on Computing 23(4):852-863, 1994

[2] M. Bernardo, *"Theory and Application of Extended Markovian Process Algebra"*, Ph.D. Thesis, University of Bologna (Italy), 1999

[3] M. Bravetti, *"Specification and Analysis of Stochastic Real-Time Systems"*, Ph.D. Thesis, University of Bologna (Italy), 2002.

[4] M. Bravetti, M. Bernardo, R. Gorrieri, *"A Note on the Congruence Proof for Recursion in Markovian Bisimulation Equivalence"*, in Proc. of the *6th Int. Workshop on Process Algebras and Performance Modeling (PAPM '98)*, C. Priami editor, pp. 153-164, Nice (France), September 1998

[5] M. Bravetti, R. Gorrieri, *"The Theory of Interactive Generalized Semi-Markov Processes"*, to appear in Theoretical Computer Science

[6] M. Bravetti, R. Gorrieri, *"A Complete Axiomatization for Observational Congruence of Prioritized Finite-State Behaviors"*, in Proc. of the *27th Int. Colloquium on Automata, Languages and Programming (ICALP 2000)*, U. Montanari, J.D.P. Rolim and E. Welzl ed., LNCS 1853:744-755, Geneva (Switzerland), 2000

[7] W.R. Cleaveland, G. Luttgen, V. Natarajan, *"Priority in Process Algebras"*, in Handbook of Process Algebra, Elsevier, pp. 711-765,2001

[8] M. Hennessy, T. Regan, *"A Process Algebra for Timed Systems"*, in Information and Computation, 117(2):221-239, 1995

[9] H. Hermanns, *"Interactive Markov Chains"*, Ph.D. Thesis, Universität Erlangen-Nürnberg (Germany), 1998

[10] H. Hermanns, *"An Operator for Symmetry Representation and Exploitation in Stochastic Process Algebras"*, in Proc. of the *5th Workshop on Process Algebras and Performance Modeling*, pp. 55-70, Twente (The Netherlands), 1997

[11] H. Hermanns, M. Lohrey, *"Priority and Maximal Progress Are Completely Axiomatisable (Extended Abstract)"*, in Proc. of the *9th Int. Conf. on Concurrency Theory (CONCUR '98)*, LNCS 1466:237-252, Nice (France), 1998

[12] J. Hillston, *"A Compositional Approach to Performance Modelling"*, Cambridge University Press, 1996

[13] C.A.R. Hoare, *"Communicating Sequential Processes"*, Prentice Hall, 1985

[14] R. Milner, *"Communication and Concurrency"*, Prentice Hall, 1989

[15] R. Milner, *"A Complete Inference System for a Class of Regular Behaviours"*, in Journal of Computer and System Sciences 28:439-466, 1984

[16] R. Milner, *"A Complete Axiomatization for Observational Congruence of Finite-state Behaviours"*, in Information and Computation 81:227-247, 1989

[17] X. Nicollin, J. Sifakis, *"An Overview and Synthesis on Timed Process Algebras"*, in Real-Time: Theory in Practice, LNCS 600, 1991

[18] C. Priami, *"Stochastic π-Calculus"*, in Computer Journal 38(6):578-589, 1995

[19] W.J. Stewart, *"Introduction to the Numerical Solution of Markov Chains"*, Princeton University Press, 1994

Petri nets with causal time for system verification

C. Bui Thanh [1], H. Klaudel [2] and F. Pommereau [3]

Université Paris 12, LACL
61 avenue du général de Gaulle
94010 Créteil, France.

Abstract

We present a new approach to the modelling of time constrained systems. It is based on *untimed* high-level Petri nets using the concept of *causal time*. With this concept, the progression of time is modelled in the system by the occurrence of a distinguished event, *tick*, which serves as a reference to the rest of the system. In order to validate this approach as suitable for automated verification, a case study is provided and the results obtained using a model-checker on high-level Petri nets are compared with those obtained for timed automata using prominent tools. The comparison is encouraging and shows that the causal time approach is intuitive and modular. It also potentially allows for efficient verification.

1 Introduction

This paper presents a case study in modelling and verification of systems with time constraints. We use an original approach based on *untimed* high-level Petri nets, using a concept of so called *causal time* [17], inspired by [5,18]. This widely differs from the classical approaches where time is introduced in Petri nets in terms of intervals or durations labelling nets elements, as in time or timed Petri nets (see [4] for a survey and a comparison of the different approaches), referring to a progression of time external to the system. The main characteristic of the causal time approach is that the progression of time is modelled *in the system* by a distinguished event, called *tick*. Thus, the occurrences of the other events may depend on the occurrences of *tick*. The time constraints of the kind "at most" or "at least 5 ticks between events t and t'" are realized by counting the appropriate number of ticks between the

[1] Email: bui@univ-paris12.fr
[2] Email: klaudel@univ-paris12.fr
[3] Email: pommereau@univ-paris12.fr

This is a preliminary version and considered for the final proceedings, to be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

occurrences of t and t'. So, the occurrence of t' is causally dependent on those of *tick* and may only occur if the time constraint is satisfied. The modelled system and the counter of ticks are both represented by high-level Petri nets interacting with each other.

We use a model of high-level Petri nets provided with a structure of process algebra, the *algebra of M-nets* [2], in which Petri nets can be composed together with operators like sequential and parallel composition. The model also allows for synchronous communication, as in CCS [16]. In this context, introducing causal time amounts to consider a net expressing a tick counter being able to interact with the system and to produce the required number of ticks between occurrences of transitions (as proposed for instance in [12]).

The main goal of this paper is to show that the causal time approach in this context allows one to model systems in an intuitive and modular way, with the potentiality of efficient verification. For this purpose, we present a comparative case study concerning the railroad crossing problem and give its specification in terms of timed automata as well as in terms of high-level nets with causal time. Various versions of the specification having different properties (for instance the absence or presence of deadlocks) are then verified using model-checkers Kronos [20] and Uppaal [13] for the timed automata, and MARIA [15] for the high-level Petri nets. The results obtained are very promising since in many cases, the causal time approach allows for a more efficient verification. At the end of the paper, we discuss the current limitations concerning the approach and the tools, and we point out some ways which can lead to significant improvements.

Throughout the paper, we assume that the reader has basic knowledge about timed automata [1,9] and coloured Petri nets [10,2].

2 Railroad crossing system (RC)

The railroad crossing system is composed of n_t trains (each of them moving on its own track) and of a pair of gates which prevent cars from crossing the tracks when a train is present.

The trains move independently and, initially, none is present. Each train starts far from the railroad crossing; it triggers a signal *app* when it approaches close enough to the gates. From this point, it reaches the gates in at least a_m and at most a_M time units. Then, it passes inside the gates during at least e_m and at most e_M time units and finally leaves the gates triggering a signal *exit*.

The gates are initially open. They close in at least g_m and at most g_M time units after receiving a signal *down*. They require the same delay for opening after receiving a signal *up*. It may happen that the gates receive the signal *down* when they are already going up; in this case also, the time needed in order to close is in the same boundaries.

A controller receives the signals from the trains and reacts by sending signals to the gates in at least c_m and at most c_M time units. It must ensure

the *safety property* which states that if a train is present at the crossing, then the gates must be closed.

The purpose of this paper is to show the usability of the causal time approach and to compare its performances with timed automata. This, we will use a simplified specification of the railroad crossing problem. For instance, we do not verify the *availability property* (gates are open as much as possible).

3 A modelling of RC with timed automata

We consider here a version of timed automata [1] which allows, in particular, for *state invariants* [9], *integer variables* (in addition to *clocks* which take real values) and *binary synchronisations*. A state invariant is a condition involving clocks and variables which must be true while the automaton stays in this state. Invariants are often used to express deadlines, for instance, $c \leq max_c$ labelling a state s means that the maximal value of the clock c in s is max_c. A transition label contains three parts separated by bars: a condition called a *guard*, a communication action (such as $act!$ or $act?$, expressing respectively a sending and a receiving on a canal act) and an expression specifying the clocks to be reset and the integer variables to be modified. For instance,

$$c \geq min_c \mid act! \mid c := 0; n := n + 1$$

indicates that the transition is possible if c is greater than min_c; if it occurs, signal $act!$ is sent, clock c is reset and the variable n is incremented. Timed automata may be composed using synchronised product inducing the synchronisation of complementary actions (like $act!$ and $act?$).

It is easy to give a modelling of RC with timed automata. The variant presented here is depicted in figure 1. A train is modelled by the automaton *Train* and the gates by the automaton *Gate*. The link between trains and gates is obtained by the automaton *Controller*. The complete specification is the synchronised product of *Gate*, *Controller* and n_t copies of the automaton *Train*.

Initially, the controller is idle, and the variable n is set to 0, the gates are open and all the trains are far from them. If a train approaches, the controller receives a signal *app* and reacts sending *down* to the gates and incrementing n. When a train leaves the crossing, it sends *exit* to the controller which decrements n. If it was equal to 1, then, *up* is sent to the gates, otherwise, no special reaction is needed.

One may notice that when the controller is in state AppDown it cannot receive any signal (*app* or *exit*) and delays their reception until it reaches the state Idle. This is unrealistic since trains cannot be stopped; however, we preferred to use this simplified version since our goal is more a comparison than a complete case study.

The tools used for this work are Kronos [20] and Uppaal [13] because they have the reputation to offer efficient verification. Deadlock freeness and safety

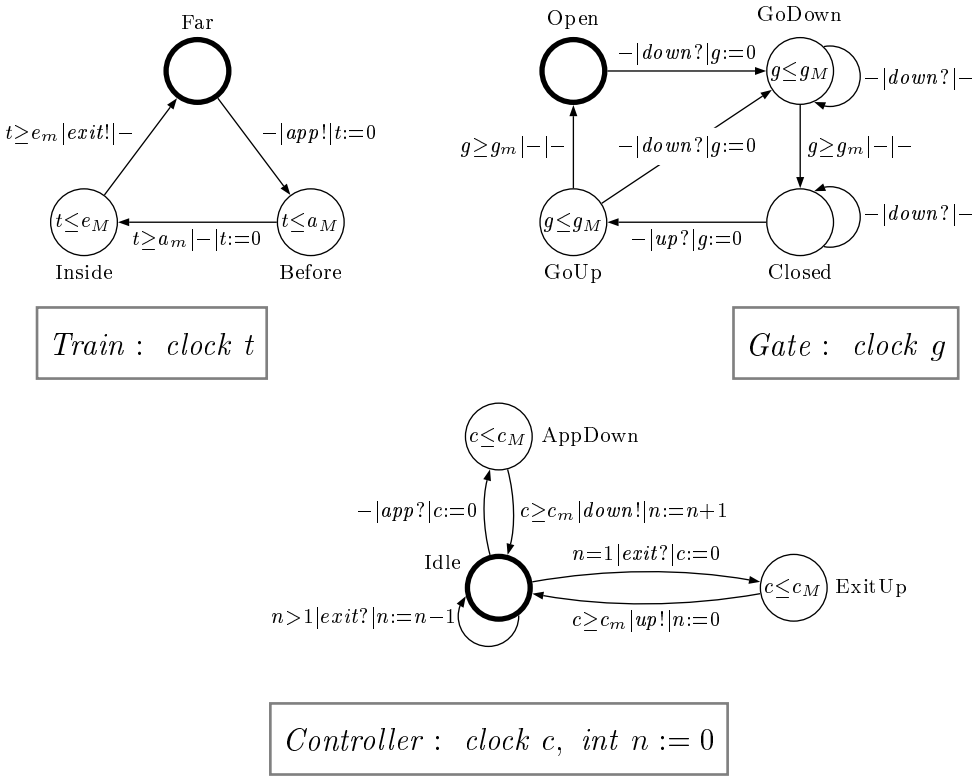

Fig. 1. The timed automata *Train*, *Gate* and *Controller*. The initial states are depicted with bold circles.

properties may be expressed through temporal logic formulas; for instance, with Uppaal, we have:

$$\forall\Box(\neg deadlock) \wedge ((Train_1.\text{Inside} \vee \cdots \vee Train_{n_t}.\text{Inside}) \Rightarrow Gate.\text{Closed}) \quad .$$

The automata presented above are directly usable with Uppaal; a non-trivial translation is necessary in order to adapt them for Kronos. Indeed, this tool uses a lower-level model without integer variables and with multi-way synchronisation. So, the automata used for Kronos are much more complicated than those presented above but they are functionally equivalent. Notice also that while Uppaal have a very nice user-friendly interface, Kronos is much more a low-level tool.

4 Composable high-level Petri nets with causal time

In this paper, we use modular high-level Petri nets, called M-nets [2], which are well suited for specifying large concurrent systems. As usual for high-level nets, their places, transitions and arcs are annotated in a specific way. In the simplest case each place has a *type* which is the set of values (tokens) it can hold; each arc is labelled by a multi-set of expressions (the simplest ones

being just values or variables); and each transition carries a *guard* which is a boolean expression playing the role of an execution condition.

An example of such a marked high-level net is shown in figure 2. This net may evolve by *firing transitions*. During the execution, the variables in the guards and in the arc annotations are *bounded* to values. A transition may fire if its guard is true and if the arcs carry only tokens belonging to the types of adjacent places. A possible execution of the net of figure 2 starts by firing t_1, which consumes the token $\bullet$ from its unique input place and produces a new marking composed of a token $\bullet$ and a token 0, each in the corresponding output place of t_1. Then, the transition t is the only enabled because of the guard of t_2 which is false for the *binding* associating z to 0, denoted $\{z \mapsto 0\}$. The firing of t with the binding $\{x \mapsto 0\}$ consumes 0 and produces 1 instead. One more firing of t is possible producing the marking 2 in its output place. Then, t_2 becomes enabled with the binding $\{z \mapsto 2\}$ and its firing consumes tokens $\bullet$ and 2 from its input places and produces $\bullet$ in the output place. With this marking, the net may not evolve anymore.

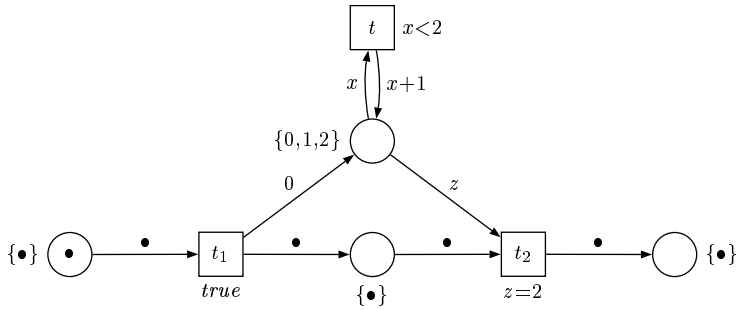

Fig. 2. A high-level Petri net.

The behaviour of an initially marked high-level Petri net may be described by a *reachability graph* whose nodes are the reachable markings and whose arcs correspond to the bounded transitions allowing to produce one marking from another. The set of all paths (starting from the initial marking) in this graph corresponds to an *interleaving semantics* of the Petri net. Several concurrent semantics may be considered, including step [7] or partial order semantics [14,6], however they are not considered in this paper.

These high-level nets may be composed in parallel by simply putting the nets side by side. They may also by synchronised (using the operation called *scoping*) in order to enforce all synchronous communications between transitions. For this purpose, we consider for the high-level nets used in this paper an labelling on transitions allowing for synchronisations. Some examples of such extended initially marked nets are given in figure 4. Their transitions are decorated by additional *labels* (the guards being as before) which are multi-sets of CCS-like communication actions (possibly with arguments which are variables or constants) as, for instance, app, $down$, $clock(x, a, b)$ or $\widehat{app}$, $\widehat{down}$, $\widehat{clock}(z, 2, c)$.

Notice that the following are always omitted in the figures: empty transition labels; guards which are always satisfied; arcs inscriptions and place types of the form $\{\bullet\}$.

The parallel composition of nets N_{Ga}, N_{Tr}, N_{Co} and N_{Cl} is $ParSys = N_{Ga} \| N_{Tr} \| N_{Co} \| N_{Cl}$ represented in figure 4. The scoping (which is a synchronisation followed by a restriction) is illustrated in figure 3; it is applied to a fragment of the net $ParSys$ with transitions t_1, t_0 and t_4 coming from nets N_{Ga}, N_{Cl}, and N_{Tr}, respectively. The synchronisation of $ParSys$ w.r.t. action $clock$ yields new transitions: t_{10} (gluing t_1 and t_0) and t_{04} (gluing t_0 and t_4). These new transitions are obtained in several steps. First, the variables appearing in the surroundings of t_1, t_0 and t_4 are renamed in order to avoid name clashes. This is necessary because, by synchronisation, these surroundings are combined into a single one. Then, a new transition is created for each pair of actions $clock$ and $\widehat{clock}$ if there is an unifier for their arguments. For instance, $\{z \mapsto x, t \mapsto c_1, c_2 \mapsto 0\}$ is a unifier allowing to synchronise t_1 and t_0. Finally, the guard of the new transition is the conjunction of the two constituent substituted guards; its label is the multi-set sum of the two constituent substituted labels, without the matching pair of actions; the arcs are all those of both former transitions (with substituted inscriptions). A restriction of the resulting net w.r.t. $clock$ gives a net in which all transitions whose labels contain at least one action $clock(\cdots)$ or $\widehat{clock}(\cdots)$, together with their surrounding arcs, are deleted, see the right hand side of figure 3 which corresponds also to the scoping of the net w.r.t. $clock$, denoted $ParSys$ sc $clock$.

Scoping may be applied with respect to a set of actions (because synchronisation is commutative and so is restriction [2]). Moreover, scoping w.r.t. action $clock$ is possible even if a transition holds several instances of this action as on t_2 in 4. In such a case, one action, say $clock(y, t, \omega)$, is first chosen for synchronisation, leading to a new transition which still holds the second action (here, $clock(y', t', 0)$). This new transition is then synchronised, yielding a new transition holding without action $clock$ and inheriting the arcs from t_2 and two pairs of arcs from t_0 (one pair for each synchronisation).

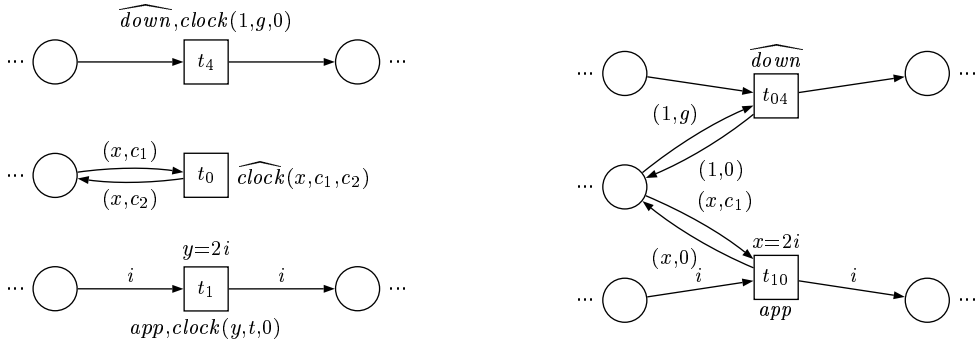

Fig. 3. A fragment of the net $ParSys$ (on the left) and a fragment of $ParSys$ sc $clock$ (on the right).

4.1 Introduction of causal time in high-level Petri nets

The basic idea behind the concept of causal time is to represent the occurrence of successive ticks (modelling the progression of time) in the same way as any other event in the system. In the context of Petri nets, events are represented by occurrences of transitions, and so, a time scale may be built by the firing of some reference transition, called *tick*. For instance, figure 2, represents a time constrained system composed of transitions t_1 and t_2 where two occurrences of t (representing the tick) are enforced between those of t_1 and t_2.

It is possible to temporally constrain a given system in a modular way. The approach consists in considering a particular net modelling a *clock*, being able to generate occurrences of ticks, to evolve in parallel to the system and to synchronise with it in order to enforce some temporal constraints. If the system has more than one independent time constraint, the clock net should be capable to manage several counting requests concurrently.

For the railroad crossing problem, we consider the clock net N_{Cl}, represented on the bottom of figure 4; it manages n_c+1 counting requests. Initially, the place Time carries $n_c + 1$ pairs of the form (j, c) where $j \in \{0, \ldots, n_c\}$ is the number of the request and c is the current value of the corresponding tick counter. Each request j has a fixed maximum value of its tick counter, max_j, which cannot be overtaken. A tick counter $c = \omega$ for some request means that this request is unused. The constant ω is assumed to be equal to $max + 1$, where max is the maximum of all the max_j's (for $0 \leq j \leq n_c$), and we state $\omega + 1 = \omega$. The transition *tick* may occur at any time provided that its guard is true (which is the case if all the temporal constraints are fulfilled and will still be true after the tick, and, in particular, if no max_j is reached). The occurrence of *tick* increments the tick counters of all requests. Initially, all the requests are unused and can be started at any time by the firing of a transition coming from the synchronisation w.r.t. *clock*.

5 A modelling of RC using Petri nets with causal time

RC is modelled by the net:

$$ParSys \ \mathsf{sc} \ \{clock, down, up, app, exit\} \quad .$$

The resulting net has the same places as *ParSys* but different transitions coming from the scoping w.r.t. all the communication actions. The scoping w.r.t. *up* and *down* ensures that the gates move exactly as the controller allows it. Analogously, the scoping w.r.t. *app* and *exit* enforces the communication between the trains and the controller. The scoping w.r.t. *clock* ensures that all counting requests are correctly handled.

The number of tick counters in clock N_{Cl} depends on the number of trains in the system because we use two counters for each train, with the following setting.

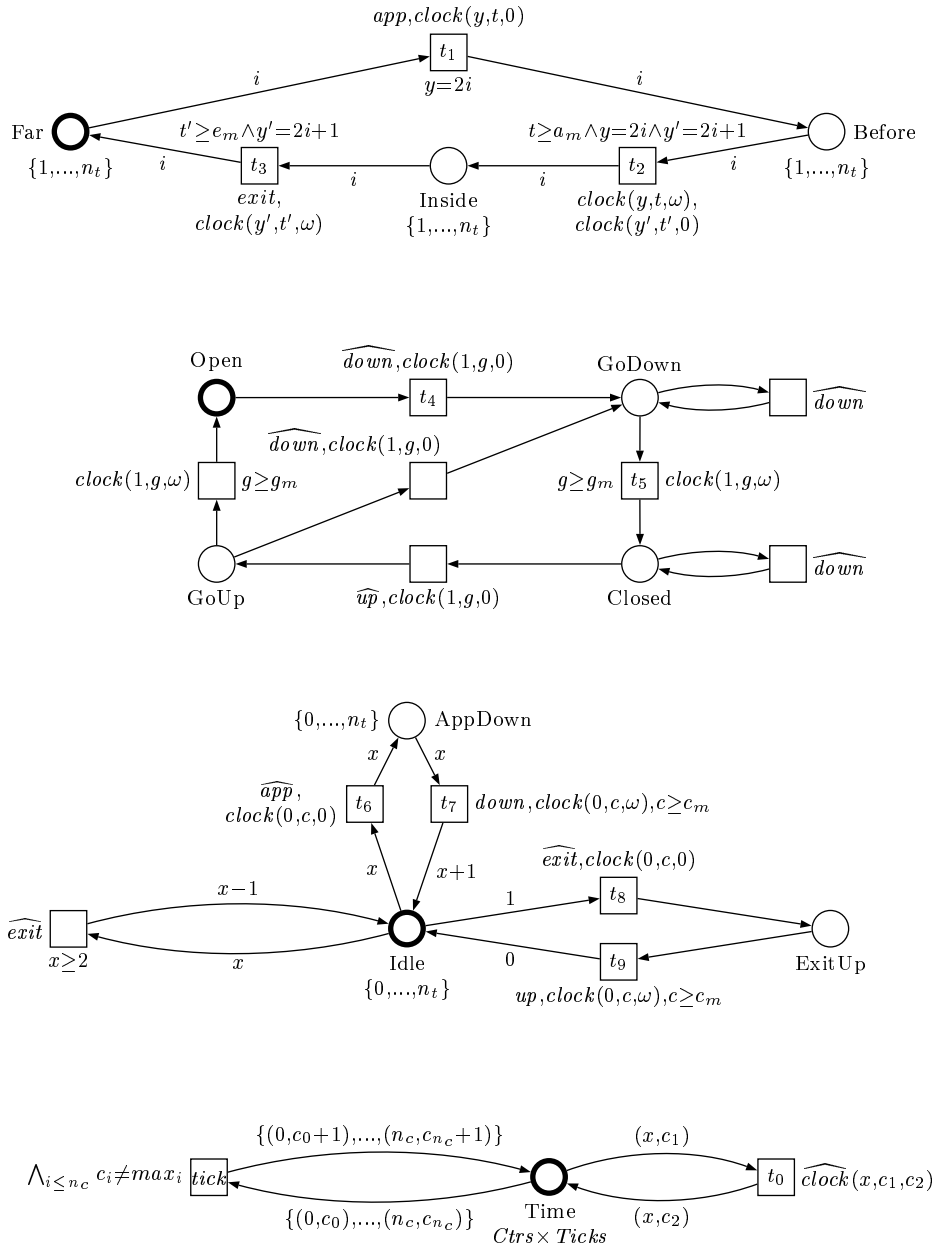

Fig. 4. The nets N_{Tr}, N_{Ga}, N_{Co} and N_{Cl} (from top to bottom, if taken separately), or their parallel composition, *ParSys* (if taken as a single net). In the figure, n_t is the number of trains, $n_c = 2n_t + 1$ is the greatest counting request number, $Ctrs = \{0, \ldots, n_c\}$ is the set of all these numbers, and $Ticks = \{0, \ldots, \omega\}$ is the set of the possible values of tick counters. Places in bold are initially marked as follows: $\{1, \ldots, n_t\}$ for Far; $\{\bullet\}$ for Open; $\{0\}$ for Idle; and $\{(0, \omega), \ldots (n_c, \omega)\}$ for Time.

The counter 0 is reserved to the controller, and its maximal value is $max_0 \stackrel{\text{df}}{=} c_M$ (see section 2). This counter is reset when a train is approaching (see transition t_6) and is used in order to ensure that signal *down* is sent to the gates after at least c_m ticks (see transition t_7). The maximum number of

ticks allowed here, c_M, is enforced in the guard of transition *tick* in the clock. Then, the same counter is used once again (for a different purpose) when the last train leaves the crossing (see transitions t_8 and t_9). Notice that if we have had different constraints in these two cases, we should have used two different counters. (This is not an intrinsic limitation of causal time but rather a limitation of the simple clock we choose to use.)

The counter 1 is reserved to the gates and its maximal value is $max_1 \stackrel{\text{df}}{=} g_M$. It is reset when the gates receive the signal to go down (see transition t_4) and it ensures that the gates are down after at least g_m and at most g_M ticks (see transition t_5 and the guard of *tick*). The same counter is used in order to ensure the opening of the gates under the same time constraints.

For each train i, for $i \in \{1, \ldots, n_t\}$, we use two distinct counters: $2i$ and $2i+1$, with $max_{2i} \stackrel{\text{df}}{=} a_M$ and $max_{2i+1} \stackrel{\text{df}}{=} e_M$, respectively. When a train approaches, at least a_m and at most a_M ticks can occur between the sending of signal *app* and the arriving of the train between the gates. This constraint is ensured by the counter $2i$ (see transitions t_1 and t_2). The counter $2i+1$ ensures that there must be at least e_m and at most e_M ticks between the crossing of the road by a train and its leaving sending the signal *exit* (see transitions t_2 and t_3). In particular, t_2 fires when the train enters the crossing; the counter $2i$ must then indicate a value greater than a_m (thanks to $t \geq a_m$ in the guard t_2) and the counter $2i+1$ is reset.

In this system, the controller holds only one token and is synchronised to all the other nets, and so, most events are interleaved. Moreover, almost all the transitions of the system are synchronised on action *clock* and thus the resulting transitions are in conflict with *tick*. This reduces again the concurrency in the system which is in fact purely sequential (which is suitable for a comparison with timed automata).

5.1 Tools used

We modelled the above specification using PEP toolkit [8] which proposes a lot of tools gathered in a convenient graphical interface. In particular, it allows one to edit high-level nets, to apply scoping on them and to convert the resulting nets into low-level (place/transition) nets which are suitable for verification using one of the model-checkers integrated with PEP. Unfortunately, we were not able to use PEP from the beginning to the end. The reason is mainly the size of the low-level nets equivalent to our high-level specification, which cannot be handled by PEP.

We used instead a high-level tool, MARIA [15], in order to check our specification against deadlock-freeness and safety. Such a tool does not need to produce low-level nets and thus it does not generate more than necessary, contrasting with the transformation from high-level nets to low-level ones which may generate, for instance, many places which will never be marked. This is particularly true in our specification, where place Time in N_{Cl} cannot hold arbitrary combination of tokens because the progression of time is not arbitrary

itself and only a small subset of possible markings are actually reachable. MARIA works on the reachability graph of coloured Petri nets and allows one to check for deadlocks and for the reachability of partial markings (sub-markings) during the generation of the graph. Deadlock freeness and the safety property could be expressed as:

```
deadlock fatal;
reject !(place Inside equals empty)
       && (place Closed equals empty) && fatal;
```

The first line specifies that if a deadlock is found, the computation of the reachability graph must be interrupted and the error reported. The rest specifies states which has to be rejected if reached. It is a C-like boolean expression on the marking of places, with lazy evaluation: if place Inside is marked and then, if place Closed is not marked, then `fatal` is evaluated, leading to abort the computation and to report the encountered rejected state.

The files produced by PEP have been converted to the file format supported by MARIA. Then, these files have been made generic, and so we are able to produce the specification for any number of trains and all kind of time constraints using a simple preprocessing. At the current state of the work, only a preprocessor and MARIA are involved in the generation and the verification of the railroad specification, but PEP was necessary in the first steps in order to produce the scoping of the nets.

6 Results

We report now the performances of the different tools during the deadlock analysis and safety verification of various versions of the specification. All the checks have been performed on a Sun Sparc station at 440Mhz, with 1Gb of physical memory and 1Gb of swap space. We worked in the `/tmp` directory which, thanks to Sun's TMPFS file system [19], is located in the virtual memory so all the work, even file accesses, was actually made in memory with proper swap. When a pre-compilation of some files has been necessary, the time consumed is included into the durations given below. This was the case for Kronos which needs to synchronise timed automata before to check them, and for MARIA which can build the guards of the transitions into libraries being then dynamically loaded by the tool in order to speed-up the evaluation. Finally, we used Unix `time` tool in order to measure the time consumed by each process (the "real" time is the one reported below).

We checked safe and deadlock-free systems for one to six trains with the following values for the different constants: $a_m = 4$, $a_M = 5$, $c_m = 0$, $c_M = 1$, $e_m = 4$, $e_M = 6$, $g_m = 0$ and $g_M = 2$. The times measured for each tool are reported in the top part of figure 5 (see also the left graph on figure 6). After about 12h30m, Uppaal exhausted all the memory and begun to be heavily swapped, using less than 1% of CPU, so we preferred to stop it since the reported time would have been meaningless.

trains	1	2	3	4	5	6
safe systems with no deadlock						
MARIA	0.2s	0.2s	0.9s	12s	4m	1h12m
Kronos	0s	0.1s	1.6s	20s	5m	1h36m
Uppaal	0.3s	0.5s	0.7s	27s	57m	-
unsafe systems with no deadlock						
MARIA	0.1s	0.2s	0.2s	0.2s	0.3s	0.4s
Kronos	0s	0.2s	1.7s	21.4s	6m57s	5h59m
Uppaal	0s	0s	0s	0s	0s	0s
deadlocking safe systems						
MARIA	0.1s	0.2s	0.2s	0.2s	0.3s	0.4s
Kronos	0s	0.2s	1.7s	21.4s	6m55s	6h02m
Uppaal	0s	0s	0s	0s	0s	0.1s

Fig. 5. The performances of the tools for "good" systems (top part), unsafe systems (middle part) and deadlocking ones (bottom part). We used specifications taking into account up to six trains.

Unsafe systems were produced with the same constants values as those used for good systems except for g_M which was here set to 3. Thus, the gates could go down too slowly and a train could cross the road while they are not yet closed. The performances are given in the middle part of figure 5 (see also the right graph on figure 6). Notice that the line for Uppaal is correct: this tool was incredibly fast with wrong systems (*i.e.*, unsafe and deadlocking ones).

Systems with deadlock were produced with the same values as for good ones. We suppressed the capability for the gates to receive a signal *down* when being or going down, by removing two transitions in each specification. Notice that for one train only, this does not produce a deadlock. The performances are reported in the bottom part of figure 5 (see also the right graph on figure 6).

6.1 Causal time w.r.t. dense time and consistency of the results

Using causal time is very natural provided that one has in mind that time constraints are expressed with respect to a time scale built by a causal clock, *i.e.*, by the occurrence of a tick which is not itself directly observable (but its consequence on the marking can be observed). Therefore, the causal time is available through tick counters, which is fairly different from reading values on a dense (or real) time scale. For instance, if c is a tick counter, equation $c = 3$ on a causal time scale means of course that "exactly three ticks occurred", but

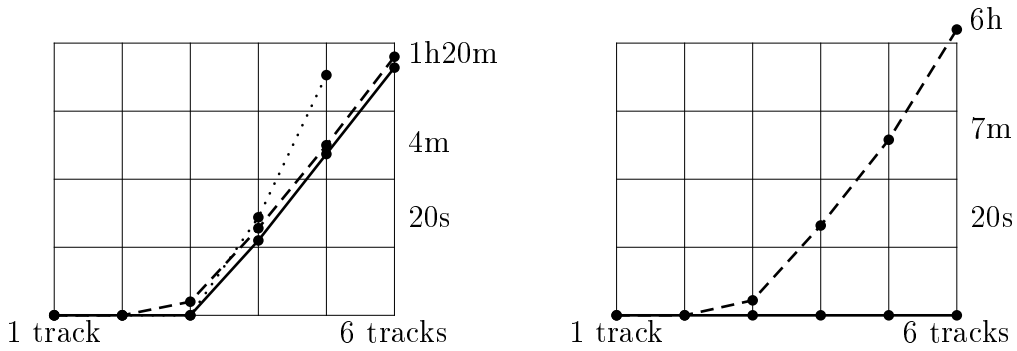

Fig. 6. The graphical representations of the performances measured for MARIA (continuous lines), Kronos (dashed lines) and Uppaal (dotted lines). The left graph is for good systems, the other for deadlocking or unsafe ones. Notice that the vertical scales are logarithmic. On the right graphics, lines for MARIA and Uppaal completely overlap.

this may mean also that a fourth tick is just about to occur. On a dense time scale, this would be expressed as $3 < c < 4$. Notice that we do not have $3 \leq c$ because the third tick has to be counted, and thus must have occurred. (We assume that actions which cannot occur concurrently are not simultaneous, and thus are separated by a non zero delay.)

Another example is the segment $5 < c \leq 6$ on a causal time scale which corresponds to $6 < c < 7$ on a dense time scale. This is not surprising if one remembers that the first constraint has to be read as "strictly more than 5 and at most 6 ticks occurred" which corresponds to "at least 6 ticks and strictly less than 7 occurred".

One can see that causal time differs from real time in many ways. However, in our case study, we used for each specification the more natural expression of time constraints, regardless of the introduced differences. Actually, we conjecture that a wide class of timed automata can be translated this way and that we can have a bisimilarity relation between the automata and the translated M-nets. In order to verify in practice this intuition, and before to obtain theoretical results, we checked many different versions of RC for many different values of the constants and with or without deadlock. These results are not presented here since they do not give more information than what we already provided. But it is worth noting that all the checks were consistent. For instance, if a given set of constants led to an unsafe system with one tool, the same happened for the other tools. Moreover, the tools always reported equivalent counter examples. As an illustration, consider the unsafe system described above for 3 trains. MARIA reports a transition sequence leading to an unsafe state which corresponds to:

(i) The token 1 in place Far (identifying the first train) is moved to place Before while the token in the controller moves from Idle to AppDown.

(ii) One tick.

(iii) The token in the controller moves from AppDown to Idle and the token for the gates from Down to GoDown

(iv) Three ticks.

(v) Train 1 moves from Before to Inside.

With Uppaal (the case of Kronos is similar), we obtain a trace which corresponds to:

(i) Train 1 goes from state Far to state Before while the controller goes from Idle to AppDown.

(ii) Delay of 1 time unit.

(iii) The controller goes from AppDown to Idle while the gates goes from Open to GoDown.

(iv) Delay of 3 time units.

(v) Train 1 goes from Before to Inside.

One may notice that a delay of three ticks with Petri nets corresponds to a delay of exactly three time units with Uppaal (and actually with Kronos also).

6.2 State space explosion

In the results reported above, it happens that the performances obtained with MARIA are generally better than those obtained with the other tools. This optimistic results have to be moderated a little bit. Actually, using MARIA, the causal time approach suffers of the well known *state space explosion* problem: when we increase the constants in the system, the number of reachable markings increases very fast. Since MARIA explicitly generates these markings, its performances become very bad.

One way to alleviate this problem would be to abstract from the net the intermediary states generated by counting ticks between two boundaries. For instance, if counter c is used in order to ensure a constraint $1 \leq c \leq 6$, only values 1 and 6 are interesting for this counter. Removing the intermediary values would reduce the number of states while preserving the interesting behaviour. This would amount for this example to consider three "meta-values": "before 1", "between 1 and 6", and "after 6". With this kind of technique, the preformances of the causal time approach would be still dependent on the number of tick counters, but not on the values of the constants compared to them.

In Petri nets, there are also other techniques trying to provide a solution to the state space explosion problem. They are typically based on the independence of some actions, often relying on the partial order view of concurrent computation. Based on such a view, the entire state space of a system may be represented implicitly, using an acyclic net in order to represent system actions and local states (see MacMillan's finite prefixes of Petri net unfoldings [14,6]). Such techniques are so far limited to low-level models of Petri nets, but recent

researches in this area showed that it is possible to produce high-level prefixes of high-level nets [11]. It is even possible to improve dramatically the efficiency of this analysis by defining an equivalence between markings, which gathers many states in the generated prefixes. This amounts to abstract data from the Petri net when it has no influence on the execution. For instance with our railroad example, place Time would appear in the prefix only when the values it holds lead to a new branch in the execution of the Petri net. Similarly, most occurrences of the transition *tick* would not be present in the prefix.

This kind of new developments will certainly soon lead to alleviate the state space explosion problem presented above. In such a case, it would not only solve this problem, but it would also increase again the performances already measured because working on finite prefixes is most of time much more efficient than the exploration of the reachability graph. This gain of performance would of course depend of the degree of concurrency we can introduce in the specification.

7 Final remarks

We presented a new approach to the modelling of time constrained concurrent systems, and developed a case study illustrating how it can be used for verification. It showed that causally timed Petri nets are easy to use (the obtained specification is similar to that given with timed automata), and offers also a quite efficient verification. This paper is the first attempt to use this model for verification, and we are aware of many improvements which could be provided. In particular, we should alleviate the state space explosion problem and the sensitivity to the constants clocks values are compared to. However, even without these optimisations, performances were quite satisfactory, what is very encouraging for the future.

We already plan further investigations in this way. On the practical side, we would like to make more case studies, in order to better appreciate the kind of problems that causal time can address efficiently. On the theoretical side, we wish to give a characterisation of a class of timed automata that could be translated into Petri nets with causal time. We think that this class may be quite wide and that it will be possible to establish a bisimilarity relation between timed automata and their translation.

A very important point in this paper is that we showed that it was possible to use successfully untimed Petri nets for the modelling of systems incorporating time constraints. Usually, various Petri net extensions were used for this purpose, where time was associated to net components like places, transitions, arcs or tokens, under the form of durations or dates. For one of these extensions, time Petri nets, it was proposed to compute branching processes including tick transitions [3]. Contrasting with this approach, we provide this kind of representation of time at the level of modelling and not only as a interpretation of another notion of time for the purpose of verification.

We plan to provide case studies comparing causal time with tools based on extended Petri net models. However, we would like to use a specification which allows for concurrency (which is not the case in this paper) because sequential systems are often the worst case for many Petri net tools (in particular for those relying on the partial order execution semantics).

Finally, we hope that tools will be developed in order to support the needs of the causal time approach. In particular, we discovered that PEP was unable to generate low-level nets from our high-level specification because of their size. Some work is already in progress in order to solve this problem. Another possibility would be to generate prefixes directly from high-level nets, as proposed in [11].

References

[1] R. Alur and D. Dill. *A theory of timed automata.* Theoretical Computer Science, 126(2). Elsevier, 1994.

[2] E. Best, W. Frączak, R. P. Hopkins, H. Klaudel and E. Pelz. *M-nets: an algebra of high level Petri nets, with an application to the semantics of concurrent programming languages.* Acta Informatica, 35. Springer, 1998.

[3] B. Bieber and H. Fleishhack. *Model checking of timed Petri nets based on partial order semantics.* CONCUR'99, LNCS 1664. Springer, 1999.

[4] A. Cerone and A. Maggiolo-Schettini. *Time-based expressivity of time Petri nets for system specification.* Theoretical Computer Science, 216. Elsevier, 1999.

[5] R. Durchholz. *Causality, time, and deadlines.* Data & Knowledge Engineering, 6. North-Holland, 1991.

[6] J. Esparza. *Model checking using net unfoldings.* Science of Computer Programming, 23. Elsevier, 1994.

[7] H. J. Genrich, K. Lautenbach and P. S. Thiagarajan. *Elements of General Net Theory.* Net Theory and Applications, Advanced Course on General Net Theory of Processes and Systems, LNCS 84. Springer, 1980.

[8] B. Grahlmann. *The PEP Tool.* Computer Aided Verification, LNCS 1254. Springer, 1997.

[9] T. A. Henzinger, X. Nicollin, J. Sifakis and S. Yovine. *Symbolic model checking for real-time systems.* LICS'92. IEEE Computer Society, 1992.

[10] K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use.* Volume 1 of EATCS Monographs on TCS, Springer, 1992.

[11] V. Khomenko, M. Koutny and W. Vogler. *Canonical prefixes of Petri net unfoldings.* CAV'02, LNCS. Springer, 2002 (to appear).

[12] H. Klaudel and F. Pommereau. *Asynchronous links in the PBC and M-nets.* ASIAN'99, LNCS 1742. Springer, 1999.

[13] K. G. Larsen, P. Pettersson and W. Yi. *UPPAAL in a nutshell.* International Journal on Software Tools and Technology Transfer, 1(1-2). Springer, 1997.

[14] K. MacMillan. *A technique of state space search based on unfoldings.* Formal Methods in System Design, 6. Kluwer Academic Publishers, 1995

[15] M. M"akel"a. *MARIA: modular reachability analyser for algebraic system nets.* Online manual, `http://www.tcs.hut.fi/maria`, 1999.

[16] R. Milner. *Calculi for synchrony and asynchrony.* Theoretical Computer Science, 25. Elsevier, 1983.

[17] F. Pommereau. *Modèles composables et conurrents pour le temps-réel.* Ph.D. Thesis. University Paris 12, 2002.

[18] G. Richter. *Counting interfaces for discrete time modeling.* Technical report 26, GMD. Sept. 1998.

[19] P. Snyder. *tmpfs: a virtual memory file system.* White Papers, Sun Microsystems Inc.

[20] S. Yovine. *Kronos: A verification tool for real-time systems.* International Journal of Software Tools for Technology Transfer, 1(1/2). Springer, 1997.

A Contribution to a Classification of Timing Attacks on Privacy

Damas P. Gruska [1,3]

Institute of Informatics, Comenius University, Mlynska dolina, 842 15 Bratislava, Slovakia

Ruggero Lanotte [2,4] and Andrea Maggiolo-Schettini [2,5]

Dipartimento di Informatica, Università di Pisa, Corso Italia 40, 56125 Pisa, Italy

Abstract

We study the problem of privacy in the framework of Timed Automata with an alphabet partitioned in two subsets of symbols representing secret and observable actions. We study two main kinds of timing attacks to privacy. The aim of the paper is to contribute to a classification of timing attacks on privacy.

1 Introduction

Several papers (see, among others, [3,4,5,9,8]) dealing with privacy, consider *two-level* systems, where the *high level* (or *secret*) behavior is distinguished from the *low level* (or *observable*) one. In the mentioned papers, systems respect the property of privacy if there is no leaking of private information, namely there is no *information flow* from the high level to the low level. This means that the secret behavior cannot influence the observable one, or, equivalently, no information on the observable behavior permits to infer information on the secret one. In the present paper we pursue the study of privacy in real-time systems begun in [7]. The framework we assume is that of Timed Automata [1]. When using this formalism, the possible behaviors of a system are described by a set of infinite *timed words*, namely infinite sequences

[1] Research partially supported by the grant VEGA 1/7654/20.

[2] Research partially supported by MURST Progetto Cofinanziato Metodi Formali per la Sicurezza e il Tempo (MEFISTO).

[3] Email: gruska@fmph.uniba.sk

[4] Email: lanotte@di.unipi.it

[5] Email: maggiolo@di.unipi.it

This is a preliminary version and considered for the final proceedings, to be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

of pairs (action performed, time of firing). In describing two-level systems, we distinguish between high-level and low-level actions. In [7] we have formulated in the formalism of automata and studied a timing attack on web privacy proposed in [2].

By a timing attack we mean an attack in which timing of events, and not only their "ordering" is important. We assume that attackers have some knowledge about the internal structure of the system to be attacked. For example, by carefully measuring the amount of time required to perform private key operations, attackers may be able to find fixed Diffie-Hellman exponents, factor RSA keys, and break other cryptosystems (see [6]).

We assume in general that attackers are *passive*, namely that they base their attacks only on observing a given timed sequence of low actions, and deriving from this observation the certainty that a certain secret (high) action has been performed. We consider two types of passive timing attack. We distinguish an attack for which the attacker uses a stopwatch, and one for which the attacker uses a watch. We consider generalizations of the problem tackled in [7] and we give solutions for the two types of attack mentioned.

More precisely, we show that the following problems are decidable for both the types of attack: with a given attack description, can an attacker detect a given private (high level) action h?; is there any timing attack such that an attacker can detect a given private (high level) action h? (the action h is or is not secure); is there any high level action h such that an attacker with a given attack description can detect the action h?; is there any timing attack and a high level action h such that an attacker can detect the action h? (the system is or is not secure). Moreover we show that an attacker with watch is strictly more powerful than that one with stopwatch.

2 LH-Timed Automata

The formalism of LH-Timed Automata [7] is an extension of Alur and Dill's Timed Automata [1] suitable to model two-level systems and to deal with problems of privacy. LH-Timed Automata are compositions of Timed Automata with the alphabet partitioned in two sets, the set H of *high symbols* and the set L of *low symbols*.

2.1 Security alphabet and timed words

A *security alphabet* is a pair consisting of two disjoint sets of *actions* (L, H). The set L contains the *low actions*, which can be performed by the system and can be observed by the external environment. The set H contains the *high actions*, which can be performed by the system and are visible only inside the system. We let l, h and a range over L, H and $L \cup H$, respectively.

Given any *time domain* T (natural numbers or non-negative rational numbers, as examples), we consider (possibly finite) *timed sequences* of the form $(a_1, t_1) \ldots (a_n, t_n) \ldots$, with $a_i \in (L \cup H)$ and $t_i \in T$, $t_i < t_{i+1}$ describing the

temporal behavior of a system that performs action a_i at time t_i.

Given a finite timed sequence $\omega_1 = (a_1, t_1) \dots (a_n, t_n)$ and a infinite timed sequence $\omega_2 = (a'_1, t'_1) \dots (a'_m, t'_m) \dots$, we say that $\omega_1 \in \omega_2$ if and only if there exists i such that for any $1 \leq j \leq n$ it holds that $a'_{i+j} = a_j$ and $t'_{i+j} = t_j$. Moreover with $\omega_1 + t$, where t is a time, we denote the sequence $(a_1, t_1 + t) \dots (a_n, t_n + t)$.

An infinite timed sequence $(a_1, t_1) \dots (a_n, t_n) \dots$ satisfying the *time progress property*, namely that for each time value $t \in T$ there is some index i such that $t_i > t$, is called a *timed word*.

Given a timed word $\omega = (a_1, t_1) \dots (a_n, t_n) \dots$, let us denote with ω_L the *observable part* of ω, i.e. the (possibly finite) timed sequence $(a_{i_1}, t_{i_1}) \dots (a_{i_m}, t_{i_m}) \dots$ such that for each index i_j, $a_{i_j} \in L$ and, for each $i_j < k < i_{j+1}$, $a_k \in H$.

2.2 Clock valuations and clock constraints

We assume a set X of variables measuring time, called *clocks*, ranged over by x. Intuitively, clocks increase uniformly with time when an automaton is in whatsoever state.

A *clock valuation* over a set of clocks X is a mapping $v : X \to T$ assigning time values to clocks. For a clock valuation v and a time value t, let $v + t$ denote the clock valuation such that $(v+t)(x) = v(x)+t$. For a clock valuation v and a subset of clocks $Y \subseteq X$, let $v[Y]$ denote the clock valuation such that $v[Y](x) = 0$, if $x \in Y$, and $v[Y](x) = v(x)$, otherwise.

Given a set of clocks X, we consider the set of *clock constraints* over X, denoted $\Theta(X)$, that is defined by the following grammar, where θ ranges over $\Theta(X)$, $x \in X$, $c \in T$ and $\# \in \{<, \leq, =, \neq, >, \geq\}$:

$$\theta ::= x \# c \mid \theta \wedge \theta \mid \neg\theta \mid \theta \vee \theta \mid true \,.$$

We write $v \models \theta$ when *the clock valuation v satisfies the clock constraint θ*. More precisely, $v \models x \# c$ iff $v(x) \# c$, $v \models \theta_1 \wedge \theta_2$ iff both $v \models \theta_1$ and $v \models \theta_2$, $v \models \theta_1 \vee \theta_2$ iff either $v \models \theta_1$ or $v \models \theta_2$, $v \models \neg\theta$ iff $v \not\models \theta$, and $v \models true$. We will assume, without loss of expressivity, that every constant c is integer (see [1]).

2.3 The formalism

Definition 2.1 A *LH-Timed Automaton* is a tuple

$$\mathcal{A} = ((L, H), A_1, \dots, A_m, F),$$

where:

(i) (L, H) is a security alphabet.

(ii) For each $1 \leq i \leq m$, $A_i = (Q_i, q_i^0, X_i, \delta_i)$ is a *sequential automaton*, with:
- a finite set of *states* Q_i

- an *initial state* $q_i^0 \in Q_i$
- a set of *clocks* X_i
- a set of *transitions* $\delta_i \subseteq Q_i \times \Theta(X_i) \times (L \cup H) \times 2^{X_i} \times Q_i$.

The sets of clocks $X_1, \ldots, X_m$ are pairwise disjoint.

(iii) $F \subseteq 2^{\Pi_{1 \leq i \leq m} Q_i}$ is a finite set of sets of *final* states.

Intuitively, a transition (q, θ, a, Y, q') of an automaton A_i fires in correspondence with the performance of action a when state q is active and the clock valuation of A_i satisfies the clock constraint θ. In such a case, state q' is entered and the clocks in Y are reset.

Let us describe now the behavior of $\mathcal{A} = ((L, H), A_1, \ldots, A_m, F)$.

A *configuration* of $\mathcal{A}$ is a tuple $s = ((q_1, v_1), \ldots, (q_m, v_m))$ such that, for each $1 \leq i \leq m$, q_i is a state in Q_i and v_i is a clock valuation over clocks X_i. The *initial configuration* s_0 is the tuple $((q_1^0, v_1^0), \ldots, (q_m^0, v_m^0))$, with q_i^0 the initial state of A_i and with v_i^0 the valuation such that $v_i^0(x) = 0$ for each clock $x \in X_i$.

There is a *step* from configuration $s = ((q_1, v_1), \ldots, (q_m, v_m))$ to configuration $s' = ((q_1', v_1'), \ldots, (q_m', v_m'))$ at time t with action a, written $s \rightarrow_t^a s'$, if and only if, for each $1 \leq i \leq m$, either there is a transition $(q_i, \theta_i, a, Y_i, q_i') \in \delta_i$ such that $v_i + t \models \theta_i$ and $v_i' = (v_i + t)[Y_i]$, or $q_i' = q_i$, $v_i' = v_i + t$ and no transition $\langle q, \vartheta, a', Y, q' \rangle$ in A_i is such that $a' = a$.

A timed word $\omega = (a_1, t_1) \ldots (a_n, t_n) \ldots$ is *accepted* by $\mathcal{A}$ if there exists an infinite sequence of steps $r = s_1 \rightarrow_{t_1}^{a_1} s_2 \rightarrow_{t_2 - t_1}^{a_2} \ldots s_n \rightarrow_{t_n - t_{n-1}}^{a_n} s_{n+1} \ldots$ such that s_1 is the initial configuration and the states crossed infinitely many times are a set in F. The *language accepted by* A is the set of timed words accepted by $\mathcal{A}$ and is denoted by $\mathcal{L}(\mathcal{A})$ and called a timed regular language.

In the next section we will use the following properties of Timed Automata. Their proofs can be found in [1].

Theorem 2.2 *The class of timed regular languages is closed under (finite) union and intersection.*

Theorem 2.3 *The emptiness problem of timed regular languages is decidable.*

By application of a cartesian product construction, any LH-Timed Automaton can be transformed into an equivalent one (namely, a LH-Timed Automaton accepting the same language) consisting of only one sequential component.

2.4 An Example

Let us assume a user's browser that interacts with its cache, with a given site w, and with other sites which may be treated as one only site e.

In Figure 1 we model this system by a LH-Timed Automaton. Automaton A_u represents the behavior of the user. This can perform a request r_e to the site e and then receive the answer a_e. Moreover, it can perform a request

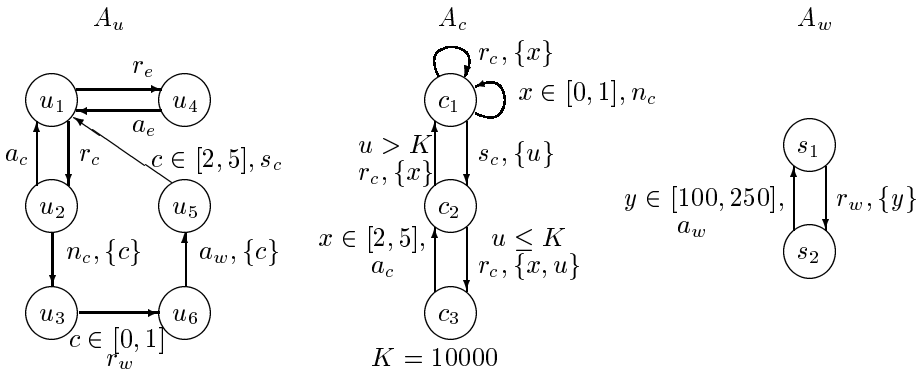

Fig. 1. The web system

r_c to the cache to obtain a web page in w. If the requested page is in the cache, then the cache gives a positive answer a_c. Otherwise, the cache gives a negative answer n_c, the user downloads the page from w (actions r_w and a_w) and, then, the page is cached (action s_c).

Automaton A_w represents the site w. The time elapsed between a request r_w (which resets clock y) and an answer a_w is in the interval $T_2 = [100, 250]$.

Automaton A_c represents the cache. When a page in w is requested by the user (action r_c) and the page is not yet in the cache (state c_1), the cache gives a negative answer (action n_c). In this case the user downloads the page, which is cached (action s_c, which resets clock u). Now, if the page is not requested for a time greater than 10000 the page is removed from the cache (such a deadline is checked by clock u) and the next request r_c causes A_c to reach state c_1. When the page is in the cache (state c_2 is active and $u \leq K$ holds), the time elapsed between a request r_c and an answer a_c is in the interval $T_1 = [2, 5]$.

Now, assume that the only observable actions for the site e are r_e and a_e, since interactions between the browser and the cache and between the browser and w cannot be seen. In [2] it is shown that, when the user visits the site e, this can infer whether the user has recently visited some web page in w or not and thus violate the privacy of the user. In fact, assume that e contains an applet that, when executed, causes a request of the page in w and, then, a request to e itself. The automaton A_a in Figure 2 represents the applet. When the user's browser downloads the page of e, it performs the applet. Therefore, if e receives the original request and the request caused by the applet within 100 units of time, it infers that no communication between the user and w has happened in the meantime, i.e. that the page was in the cache of the user. In fact, the browser takes at least 100 units of time to download a page from w.

The overall system is described by the LH-Timed-Automaton

$$\mathcal{A} = ((\{r_e, a_e\}, \{a_c, r_c, s_c, r_w, a_w\}), A_u, A_c, A_w, A_a, F),$$

where F is the set of all states of the composed automata.

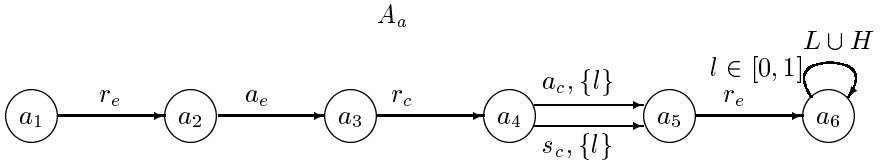

Fig. 2. The applet

2.5 Region graph

Let us recall now the notion of *region graph* of a Timed Automaton, as given in [1]. We can consider, without loss of generality, automata with only one sequential component. Moreover, as it was mentioned before, we assume that all constants in clock constraints of automata are integers.

Let us consider the equivalence relation $\sim$ over clock valuations such that:

- for each clock x, either $\lfloor v(x) \rfloor = \lfloor v'(x) \rfloor$, or both $v(x)$ and $v'(x)$ are greater than c_x, with c_x the largest integer appearing in clock constraints over x.
- for each pair of clocks x and y with $v(x) \leq c_x$ and $v(y) \leq c_y$, $fract(v(x)) \leq fract(v(y))$ if and only if $fract(v'(x)) \leq fract(v'(y))$ ($fract(\_)$ indicates the fractional part).
- for each clock x with $v(x) \leq c_x$, $fract(v(x)) = 0$ if and only if $fract(v'(x)) = 0$.

From the definition it follows that for each pair of valuations v and v', and for each clock constraint θ, it holds that:

$$\text{if } v \sim v' \text{ then } v \models \theta \text{ iff } v' \models \theta.$$

A *clock region* is an equivalence class of clock valuations, induced by $\sim$. We denote by $[v]$ the clock region to which the clock valuation v belongs. Note that the set of the clock regions is finite.

A *region* is a pair $(q, [v])$, with q a state and $[v]$ a clock region. The *initial region* is the pair $(q^0, [v^0])$ with q^0 the initial state and v^0 the valuation such that $v^0(x) = 0$, for each clock x.

The *region graph* $R(\mathcal{A})$ is a graph having the regions of $\mathcal{A}$ as set of nodes and having an edge $\langle (q, [v]), a, (q', [v']) \rangle$ if and only if, for some pair of valuations $v \in [v]$ and $v' \in [v']$, $(q, v) \rightarrow_t^a (q', v')$ for some time t.

By introducing "empty" transitions, i.e. transitions which represent only elapsing of time, we can construct a new region automaton in which every transition takes at most one time unit. So a computation is divided into several steps which represent moves from a clock region to the next time successor region, given by the elapsing of time. From now on we assume this kind of region automata.

3 Timing Attacks

A timing attack is an attack in which timing of events is important and not only their "ordering". We assume that an attacker has some knowledge about the internal structure of the system to be attacked.

We consider four questions:

1 Is it decidable whether an attacker with a given attack description can detect a given private (high level) action h?

2 Is there any timing attack such that an attacker can detect a given private (high level) action h? (the action h is or is not secure)

3 Is there any high level action h such that an attacker with a given attack description can detect the action h?

4 Is there any timing attack and a high level action h such that an attacker can detect the action h? (the system is or is not secure)

As anticipated, we study two major kinds of attacks.

3.1 A Passive Stopwatch Attacker

An attacker observes only low level actions and with the help of his stopwatch he can measure time which elapsed between any two of them (it is the same as he would have no information about the time of an attacked system initialization and/or system history till the moment when the attack starts). His input is the sequence of low level actions $\omega = (l_1, 0)(l_2, t_2) \dots (l_k, t_k)$ such that t_2 and $t_i - t_{i-1}, 3 \leq i \leq k$, represent the time which is elapsed between l_1 and l_2 and between l_{i-1} and l_i, respectively. We will call *timing* of the sequence $l_1, \dots, l_k$ the sequence of delays between the low level actions l_i.

Definition 3.1 (stopwatch timing attack) A finite sequence of low level actions $\omega = (l_1, 0)(l_2, t_2) \dots (l_k, t_k)$ is a stopwatch timing attack on privacy of $\mathcal{A}$ to detect h, with notation $\mathcal{A} \stackrel{sw}{\Rightarrow}_{\omega} h$, iff

(i) there exist $\omega' \in \mathcal{L}(\mathcal{A})$ and t such that $\omega + t \in \omega'_L$,

(ii) for every $\omega' \in \mathcal{L}(\mathcal{A})$ if $\omega + t \in \omega'_L$, for some t, then there exists $t' \in [t, t+t_k]$ such that $(h, t') \in \omega'$,

(iii) there exist $\omega'' \in \mathcal{L}(\mathcal{A})$ and $(l_1, t'_1), \dots, (l_k, t'_k) \in \omega''_L$ such that for any $t' \in [t'_1, t'_k]$ it holds that $(h, t') \notin \omega''$. (i.e. it is really a timing attack - with different timing h cannot be detected).

Note that from the second condition it follows that timing of ω'' in the third condition is indeed different from that of ω in the sense that there is no t such that $\omega'' = \omega + t$ i.e. that $t_i - t_{i-1}$ differs from $t'_i - t'_{i-1}$ for at least one i.

An attack $(l_1, 0)(l_2, t_2)$ (only two low level actions and time between them are observed) will be called a *simple stopwatch timing attack* (question/answer

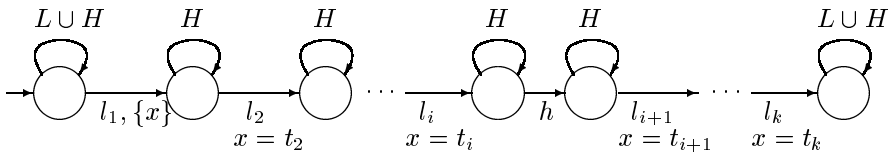

Fig. 3. The automaton accepting $L_{\omega,i,h}$

scenario). The attack to "smart cards" [6] is a simple stopwatch timing attack.

With reference to our example, the low sequence $(a_e, 0)$ $(r_e, 50)$ $(a_e, 250)$ $(r_e, 300)$ permits to infer that the web system has performed the secret symbol a_c (i.e. the page was in the cache).

Answer to Question 1: Given $\omega = (l_1, 0)(l_2, t_2) \dots (l_k, t_k)$ and $h \in H$. Does it hold $\mathcal{A} \stackrel{sw}{\Rightarrow}_\omega h$?

To answer this question we have to show that:

- there is at least one word in $\mathcal{L}(\mathcal{A})$ that contains ω and also the high level action h which occurred between l_1 and l_k,
- there is not a word in $\mathcal{L}(\mathcal{A})$ that contains ω but does not contain the high level action h which occurred between l_1 and l_k (this and the previous requirement together correspond to (i) and (ii) in Definition 3.1),
- there is at least one word in $\mathcal{L}(\mathcal{A})$ that contains low level actions $l_1, \dots, l_k$ but does not contain the high level action h which occurred between l_1 and l_k. Note that if the previous requirement is satisfied then clearly the timing of $l_1, \dots, l_k$ is different from that of ω.

The problem can be reduced to the emptiness problem. First we need some notation. Let $\omega = (l_1, 0)(l_2, t_2) \dots (l_k, t_k)$, $i \in [1, k-1]$ and $h \in H$; with $L_{\omega,i,h}$ we denote the set of words such that h appears between l_i and l_{i+1}. The LH-Timed Automaton in Figure 3 recognizes $L_{\omega,i,h}$. With $L_{\omega,\overline{h}}$ we denote the set of words such that h does not appear between l_1 and l_k. The LH-Timed Automaton in Figure 4 recognizes the language $L_{\omega,\overline{h}}$. Moreover, let $\overline{\omega} = (l_1, 0)(l_2, t'_2) \dots (l_k, t'_k)$ for some t'_i; with $L_{\overline{\omega},\overline{h}}$ we denote the set of words such that h does not appear between l_1 and l_k (there is no need to require that the timing of $\overline{\omega}$ is different from the timing of ω). The LH-Timed Automaton in Figure 5 recognizes $L_{\overline{\omega},\overline{h}}$. Now, for a given $\omega = (l_1, 0)(l_2, t_2) \dots (l_k, t_k)$ and $h \in H$, it holds $\mathcal{A} \stackrel{sw}{\Rightarrow}_\omega h$ if and only if

- $\mathcal{L}(\mathcal{A}) \cap \bigcup_{i=1}^{k-1} L_{\omega,i,h} \neq \emptyset$,
- $\mathcal{L}(\mathcal{A}) \cap L_{\omega,\bar{h}} = \emptyset$,
- $\mathcal{L}(\mathcal{A}) \cap L_{\overline{\omega},\bar{h}} \neq \emptyset$.

So, the following proposition derives directly (see Theorems 2.2 and 2.3).

Proposition 3.2 *Let $\omega = (l_1, 0)(l_2, t_2) \dots (l_k, t_k)$ and $h \in H$; it is decidable whether $\mathcal{A} \stackrel{sw}{\Rightarrow}_\omega h$.*

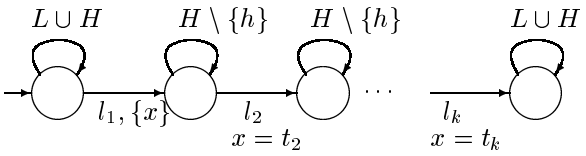

Fig. 4. The automaton accepting $L_{\omega,\overline{h}}$

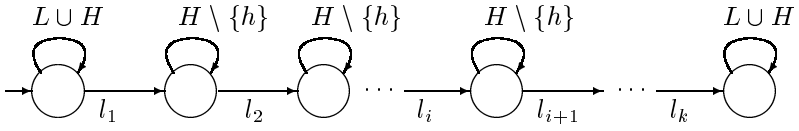

Fig. 5. The automaton accepting $L_{\overline{\omega},\overline{h}}$

Answer to Question 2: Given $\omega = (l_1, 0)(l_2, t_2) \ldots (l_k, t_k)$, is there any $h \in H$ such that $\mathcal{A} \stackrel{sw}{\Rightarrow}_\omega h$? (i.e., can we deduce anything private by the attack ω?)

The problem can be reduced to Question 1. We check, for every action h from H, whether $\mathcal{A} \stackrel{sw}{\Rightarrow}_\omega h$.

Answer to Question 3: Given h, is there any $\omega = (l_1, 0)(l_2, t_2) \ldots (l_k, t_k)$ such that $\mathcal{A} \stackrel{sw}{\Rightarrow}_\omega h$? (i.e., is the high level action h secure?)

First, we will assume only simple stopwatch attacks, i.e. attacks of the form $(l_1, 0)(l_2, t_2)$. There are finitely many pairs of low level actions with infinitely many possible time durations among them, but the next lemma claims that for every automaton there exists a constant C_{max} such that if there is an attack then there is also an attack with delay between l_1 and l_2 shorter than C_{max}.

Lemma 3.3 *For every LH-Timed Automaton $\mathcal{A}$ there exists a constant C_{max} such that for every simple stopwatch timing attack $\omega = (l_1, 0)(l_2, t_2)$ to detect h, i.e. $\mathcal{A} \stackrel{sw}{\Rightarrow}_\omega h$, there exists a simple stopwatch timing attack $(l_1, 0)(l_2, t_2')$ to detect h, such that $t_2' \leq C_{max}$.*

Proof. The idea of the proof is the following. Any computation path between l_1 and l_2 can be shortened in two ways: all cycles which do not contain h can be removed; all delays longer than the biggest constant in clock constraints can be shortened. Hence the high level action can be detected in a shorter time.

Let $R(\mathcal{A})$ be the region automaton corresponding to $\mathcal{A}$. Suppose that it has m transitions and $C_{max} = 2m + 2$, and let $\omega = (l_1, 0)(l_2, t_2)$ such that $\mathcal{A} \stackrel{sw}{\Rightarrow}_\omega h$. The existence of a timing attack means that there is at least one computation path between l_1 and l_2 such that the high level action h is always in it whenever a time delay between these two low level actions is t_2. For every such a computational path, there exists a computational path of a corresponding $R(A)$. Region automaton does not contain direct timing information, but from the choice of the automata we know that each

transition takes at most one time unit. Suppose that $t_2 > C_{max}$. This means that the number of transitions in the subpath of the "computation" of the region automaton starting with transition l_1 and finishing with l_2 exceeds twice the number of all transitions. Moreover, since $\mathcal{A} \stackrel{sw}{\Rightarrow}_\omega h$, somewhere in this path there must be a h transition. It is clear that at least one of the "subcomputations" from l_1 to h and from h to l_2 can be shortened in such a way way that they will contain each transition at most once. And, since every transition represents at most one time unit of elapsed time, it is clear that there exists a simple attack with time between the two observable actions shorter than C_{max}. □

Theorem 3.4 *For every LH-Timed Automaton $\mathcal{A}$ and every action h it is decidable whether there exists a simple stopwatch timing attack $\omega = (l_1, 0)(l_2, t_2)$ such that $\mathcal{A} \stackrel{sw}{\Rightarrow}_\omega h$.*

Proof. For every pair of visible actions l_1, l_2 we try all durations t_2 in the set $\{\frac{n}{2} | n = 1, \ldots, 2C_{max} + 1\}$. According to Lemma 3.3 it has no sense to check delays between l_1, l_2 longer than $C_{max} + 1$ but we have to check all smaller values. Since our timed automaton has only integer constants in clock constraints it is enough to check all integers smaller than $C_{max} + 1$ as well as some time delay between i and $i + 1$, for every $i, 0 \leq i \leq C_{max}$. □

The results of Lemma 3.3 and Theorem 3.4 can be generalized to stopwatch timing attacks.

Theorem 3.5 *For every LH-Timed Automaton $\mathcal{A}$ and every action h it is decidable whether there exists an attack ω such that $\mathcal{A} \stackrel{sw}{\Rightarrow}_\omega h$.*

Proof. Suppose that the region automaton corresponding to $\mathcal{A}$ has m transitions and suppose that there is an attack $\omega = (l_1, 0)(l_2, t_2) \ldots (l_k, t_k)$ which detects a high level action h. It is clear that the same action can be detected with the attack ω' such that the number of its low level actions before and also after h does not exceed the number of transitions of the corresponding region automaton. This means that any attack can be shortened by removing "cycles" before and after transition labeled by h, i.e. for any attack there exists the attack which contains at most $2m$ transitions. Therefore, in order to decide whether there exists an attack to detect h, it is enough to try all sequences of low level actions shorter than $2m$. As regards the time elapsing between any two low level subsequent actions, one has to try every duration from the set $\{\frac{n}{2} | n = 1, \ldots, 2C_{max} + 1\}$. □

Answer to Question 4: Are there any ω and h such that $\mathcal{A} \stackrel{sw}{\Rightarrow}_\omega h$? (is the automaton $\mathcal{A}$ secure?) The decidability of this problem follows from the following corollary of the previous theorem.

Corollary 3.6 *For every LH-Timed Automaton $\mathcal{A}$ its security is decidable, i.e. it can be decided whether there exists an attack ω and $h \in H$ such that $\mathcal{A} \stackrel{sw}{\Rightarrow}_\omega h$.*

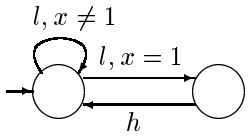

Fig. 6. The LH-Timed Automaton $\mathcal{A}$.

Proof. According to the previous theorem it is enough to check every action from H. □

3.2 A Passive Watch Attacker

An attacker observes only low level actions and, with the help of a watch, he can observe also the (absolute, not relative) time of their occurrence. So, he observes a sequence of actions $\omega = (l_1, t_1) \ldots (l_k, t_k)$.

Definition 3.7 (watch timing attack) A finite sequence of low level actions $\omega = (l_1, t_1) \ldots (l_k, t_k)$ is a watch timing attack on privacy of $\mathcal{A}$ to detect h, with notation $\mathcal{A} \overset{w}{\Rightarrow}_{\omega} h$, iff

(i) there exist $\omega' \in \mathcal{L}(\mathcal{A})$ such that $\omega \in \omega'_L$,

(ii) for every $\omega' \in \mathcal{L}(\mathcal{A})$ if $\omega \in \omega'_L$ then there exists $t \in [t_1, t_k]$ such that $(h, t) \in \omega'$,

(iii) there exist $\omega'' \in \mathcal{L}(\mathcal{A})$ and $(l_1, t'_1), \ldots, (l_k, t'_k) \in \omega''_L$ such that for any $t, t \in [t_1, t_k]$ $(h, t) \notin \omega'$ (i.e. it is really a timing attack, with different timing h cannot be detected). Note again (as with stop watch attacks) that from the second condition it follows that timing of ω'' in the third condition is indeed different from that of ω, in the sense that that t_i differs from t'_i for at least one i.

We shall call a *simple watch timing attack*, a watch attack consisting of the observation of just one low level action and time of its occurrence, i.e. $\omega = (l_1, t_1)$. In this special case we will require that h occurred before time t_1.

The four questions considered for the stopwatch attack have the same answer for the watch attack. It is sufficient to delete the reset of clock x in the automaton of Figures 3, 4 and 5. So, the following theorem derives directly from the proofs of the results of the previous section.

Theorem 3.8 *Properties of stopwatch attacks as stated in the previous section hold also for watch attacks.*

Consider a LH-Timed Automaton $\mathcal{A}$ in Figure 6 with $L = \{l\}, H = \{h\}$. It is easy to see that under stopwatch attack the above automaton is secure, i.e. there is no attack of any length to detect h. On the other side it is enough to observe $(l, 1)$ for a watch attacker to detect that the high level action h was performed. This example justifies the following theorem.

Theorem 3.9 *A watch attacker is strictly more powerful than a stopwatch attacker.*

4 Conclusions and further work

Timing attacks can "break" systems which use "unbreakable" algorithms. Hence the importance of their study for privacy.

In this paper we have considered two types of timing attack and proved decidability of four questions about them. The obtained results can be easily extended to timing attacks which detect not just a single high level action but a sequence of high level actions or a set of them. We see our work as a very preliminary step towards an analysis and a classification of timing attacks on privacy. Further study will concern more efficient decision algorithms than the ones based on region automata, and possibly the individuation of other types of attack.

References

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

[2] Felten, E.W., and M.A. Schneider: Timing attacks on Web privacy. Proc. 7^{th} ACM Conference on Computer and Communications Security, 25–32, 2000.

[3] Focardi, R., and R. Gorrieri: Automatic compositional verification of some security properties. Proc. Second International Workshop on Tools and Algorithms for the Construction and Analysis of Systems, Lecture Notes in Computer Science 1055, Springer, Berlin, 1996, 167-186.

[4] Focardi, R., and R. Gorrieri: *A classification of security properties for process algebras.* Journal of Computer Security **3** (1995), 5–33.

[5] Focardi, R., R. Gorrieri, and F. Martinelli: Information flow analysis in a discrete-time process algebra. Proc. 13^{th} Computer Security Foundation Workshop, IEEE Computer Society Press, 2000.

[6] Kocher P.C.: Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS and Other Systems, Proc. Advances in Cryptology - CRYPTO'96, Lecture Notes in Computer Science 1109, Springer, Berlin, 1996, 104-113.

[7] Lanotte, R., A. Maggiolo-Schettini, and S. Tini: *Privacy in real-time systems.* Proc. MTCS'01. Electronic Notes in Theoretical Computer Science **52(3)** (2001).

[8] Smith, G., and D. Volpano: Secure information flow in a multi-threaded imperative language. Proc. ACM Symposium on Principles of Programming Languages, 1998, 355–364.

[9] Volpano, D., and G. Smith: *Confinement properties for programming languages.* SIGACT News **29** (1998), 33–42.

Process Algebras as Specification Language (work in progress)

Friedger Müffke [1,2]

Computer Science Department
University of Bristol
Bristol, England

Abstract

Recent approaches to the combination of process algebras and temporal logic have shown that it is possible to specify logical formulae as processes. The presented work exploits the applicability of these approaches, in the context of globally clocked process algebras, and shows how logical expressions describing the history of a system can be used to simplify system specifications with process algebras. In order to allow modelling of composed systems at a high level, one-way communication is used to build the model of the system. Finally, an outline of how these models can be refined is given.

1 Motivation and Related Work

Process algebras have been successfully applied to various systems as a formal modelling language. Their strength lies in their ability to easily abstract from details of a system model by means of the hiding operator.

The analysis of these models usually involves either another formalism to specify properties, like temporal logical expressions that can be verified using a model checker, or refinement relations that compare two models of a system at different levels of abstraction.

More recently, logical formulae have been translated to processes such that refinement relations can be used to verify properties of a system model. The first paper explaining this relationship was published recently in [3]. It presents two attempts to use refinement-based approaches for the verification of LTL formulae.

In the first attempt, a property was represented by the most non-deterministic process such that all processes satisfying the property are trace refine-

[1] Supported by EPSRC grant: GR/M 93758.
[2] Email: muffke@cs.bris.ac.uk

This is a preliminary version and considered for the final proceedings, to be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

ments of the property process. As refinement checkers like FDR [2] use finite traces, only pure safety properties can be verified like this. These are exactly the properties for which it is decidable after a finite number of transitions of the system whether the property is not satisfied.

In the second attempt, a constraining process was introduced. This process was put in parallel composition with the system. Using failure refinement it was possible to verify the represented property for system models that have a finite state space and are deadlock-free.

In practice, it appeared to be difficult to get convincing results from this property checking method because a negative answer from it can result either from the fact that the property does not hold for the system or from a modelling error in the system itself. The way FDR handles system and property processes makes it very laboriously to distinguish between the two cases.

The second approach about specifications as refinement is presented in [6]. The authors deal with safety properties in timed CSP and define a new semantics for dense-time. They suggest to discretise the problem and reduce it to untimed CSP processes such that the refinement can be checked using FDR, provided that priority is given to clock signals.

Both approaches use processes to describe properties that are equivalent to logical formulae. This is advantageous because the properties are expressed in the same language as the system model. Moreover, they can be analysed with well-established refinement methods for processes.

In the work presented here, this approach is used to extend the modelling capabilities of process algebras. The system and its properties are modelled as different processes. When they are placed in parallel the latter can reflect whether the system satisfies the property or not. The property process can be seen as an online-diagnostic process; it observes the system and represents its state.

The observation of the system by the diagnostic process is realized using one-way communication. One-way communication and broadcasting in a framework of programming languages was studied in [7]. The Calculus of Broadcasting Systems (CBS) that is closely related to CCS [4] is presented in that publication. A process in CBS describes its ability to speak to its environment. All processes are input-enabled, i.e. they always listen to everything that other processes output. This implies that all processes progress synchronously and only one process can speak at a time. The one-to-one communication realized in CCS is converted to a one-to-all communication.

Furthermore, there is no general way in CBS to distinguish between an action that is heard by a process and one that is ignored by all parallel processes because it depends on the implementation of the communication in the programming language underneath whether an action is heard or not. In the process algebra this is hidden by means of an abstract, so-called translator function. This function translates all actions into either heard (audible) or ignored (non-audible) actions.

The presented process algebra can be seen as an implementation of a slightly amended version of CBS on top of a classical process algebra. The restriction in CBS that only one process can speak at a time is not practical for modelling diagnostic processes. Therefore, a new process algebra is developed that allows both broadcasting communication and multicast communication in CSP-style. Processes can listen to synchronised and unsynchronised actions. Furthermore, processes can progress asynchronously. In contrast to CBS, they are not necessarily input-deterministic.

This process algebra can model systems and properties as well as a combination of both. A property processes placed in parallel composition with a system records the current state of the system without changing or constraining its behaviour. To access this state from the system process an if-then-else construct is added to the process algebra.

The first process algebra augmented with conditional choice was ACP described in [1]. The conditional choice construct is extended here to propositional logic with past operators.

The next section introduces the process algebra that is used as the modelling language. A global clock signal and a second type of action that is used for one-way communication is included in its alphabet. In Section 3 it is explained how properties can be expressed in the process algebra. Finally, the use of the if-then-else construct is presented. Its applicability is shown by means of an example in Section 5.

2 Process Algebra with a Global Clock and One-way Communication

2.1 Informal Description

The process algebra used as a basis for the extension is similar to CSP [9]. It consists of a set of actions $\mathcal{A}$, the prefix operator $\rightarrow$, the external choice operator $+$ and the parallel composition operator $\|$. The parallel operator is augmented with a synchronisation set S that allows to specify actions the two processes in parallel composition have to synchronise on, denoted by $\|_S$. Furthermore, the result of communication is again a visible action. Therefore, multicast communication is possible.

Like in ATP [5] a clock action *clk* is introduced that allows for synchronisation of all components of a system. The components that are ready to perform a *clk* action are blocked until all their parallel components are ready to perform it. Within two clock actions the components can progress independently and asynchronously as long as they do not have to synchronise with each other on some other actions. The interval between two clock actions is often referred to as a clock cycle.

The beginning, resp. the end, of a clock cycle form particular states of the system because all components are in synchronisation and have not yet

performed any action, resp. have performed all actions possible within the clock cycle. These states will be used later to evaluate whether the model satisfies certain safety properties.

As the process algebra was chosen for modelling hardware protocols actions are not delayable. This decision does not influence the further development of the process algebra, but it means that timelock can occur.

By introducing a new type of action the process algebra can express one-way communication actions. Normal actions can be seen as sending actions. The new type acts as receiving actions – listening to the sender. Sending and receiving are complementary to each other but not symmetrical. The new action can only be performed in synchronisation with the sending action, the process performing the sending action, however, can progress independently if its parallel components do not listen to the action (and, of course, do not have to synchronise on it). In other words, on that instant the sending process is not influenced by its parallel processes and it will not be held up from progressing. Therefore, the action that the receiver performs is called "passive synchronisation". This distinguishes one-way communication from classical communication where both the sender and receiver processes can be delayed. In contrast to asynchronous communication, the sender and receiver in a one-way communication system communicate without delay.

The new action type can be integrated into a process algebra with synchronised communication without the need for the clock action. The clock action is only used when properties are modelled. Note also that normal synchronisation is not affected by passive synchronisation actions. The blocking due to non-readiness of a parallel component can still occur. Passive synchronisation action are added orthogonally and have no influence on the existing semantics.

For convenience, the new listening actions are denoted by a tilde on top of an action. Hence, the process $Q = \tilde{a} \rightarrow P$ is ready to receive an action a. This can be read as "Q is listening to a".

2.2 Remarks on One-way Communication

The implication of the passive synchronisation actions is that the behaviour of synchronisation changes temporarily. In classical process algebras the synchronisation set is fixed, in the presented extension the synchronisation set depends on the state of the system. The two *active* semantic rules in Table 1 specify how to adapt the synchronisation set. The rules state that passive synchronisation actions always have to be performed in synchronisation with all other parallel processes. That means, in the following example, if the passive synchronisation action $\tilde{a}$ is enabled the synchronisation set must contain a, otherwise it should be kept empty. Note, that the processes are equal in the sense that normal actions are defined to be equal whether they are received

by another component or not. [3]

$$(b \to STOP + \tilde{a} \to STOP) \|_{\{\}} a \to STOP = \\ b \to (STOP \|_{\{\}} a \to STOP) + (a \to STOP \|_{\{a\}} a \to STOP) = \\ b \to a \to STOP + a \to STOP$$

The extension of the process algebra is a conservative extension as the properties of classical process algebra are retained if the new actions are not applied. It is truly an extension as it adds expressiveness with respect to the synchronisation behaviour. To be more precise, in classical process algebra it is not possible to model actions that are performed in synchronisation only if another process is ready to do so.

The following examples show the two possibilities of classical communication in process algebra. In the first example both processes have to synchronise on a. Therefore, only one a can be performed. In the second one, the synchronisation set is empty and actions can be performed in an interleaving manner preserving the defined order of each process.

(1) $$a \to a \to STOP \|_{\{a\}} a \to b \to STOP = \\ a \to b \to STOP$$

(2) $$a \to a \to STOP \|_{\{\}} a \to b \to STOP = \\ a \to a \to a \to b \to STOP + \\ a \to a \to b \to a \to STOP + \\ a \to b \to a \to a \to STOP$$

In contrast, passive synchronisation actions allow the modelling of behaviour that is more restrictive than interleaving actions, but less constraining than synchronised actions. Therefore, passive synchronisation adds further expressiveness to the process algebra. In the third example, the first a has to be performed in synchronisation. Thereafter, a and b can occur in any order.

(3) $$a \to a \to STOP \|_{\{\}} \tilde{a} \to b \to STOP = \\ a \to a \to b \to STOP + \\ a \to b \to a \to STOP$$

This behaviour cannot be modelled in classical process algebra. The only way to approximate the behaviour is using an auxiliary communication medium, e.g. modelled by a channel such as in [9]. The main deficiency is that the medium delays the sending and receiving action. It is an important property of one-way communication that sent messages are received immediately. This plays a major role when modelling logical formulae as processes.

[3] Later, action transitions are augmented by a boolean variable to distinguish between received and ignored actions.

passive prefix

$$\frac{}{\tilde{a} \to p \xrightarrow{\tilde{a}} p}$$

passive parallel sync

$$\frac{p \xrightarrow{\tilde{a}} p' \quad q \xrightarrow{\tilde{a}} q'}{p \|_S q \xrightarrow{\tilde{a}} p' \|_S q'}$$

passive parallel nosync

$$\frac{p \xrightarrow{\tilde{a}} p' \quad q \xrightarrow{\tilde{a}} \not\to}{p \|_S q \xrightarrow{\tilde{a}} p' \|_S q}$$

passive external choice

$$\frac{p \xrightarrow{\tilde{a}} p'}{p + q \xrightarrow{\tilde{a}} p'}$$

Rules for passive synchronisation actions

active parallel sync

$$\frac{p \overset{a,M}{\Longrightarrow} p' \quad q \xrightarrow{\tilde{a}} q'}{p \|_S q \overset{a,T}{\Longrightarrow} p' \|_S q'} \quad a \notin S$$

active parallel nosync

$$\frac{p \overset{a,M}{\Longrightarrow} p' \quad q \xrightarrow{\tilde{a}} \not\to}{p \|_S q \overset{a,M}{\Longrightarrow} p' \|_S q} \quad a \notin S$$

active hiding

$$\frac{p \overset{a,M}{\Longrightarrow} p'}{p \backslash S \overset{\tau,F}{\Longrightarrow} p' \backslash S} \quad a \in S$$

Additional rules for progress transitions

Table 1
Semantics rules

2.3 Formal Definition

The new one-way communication is defined by a second set of actions. For processes that use actions of this set new deduction rules are added to the operational semantics of classical process algebras

Definition 2.1 (Passive Synchronisation) Let $\mathcal{A}$ be the set of (normal) actions, then $\tilde{\mathcal{A}} = \{\tilde{a} | a \in \mathcal{A}\}$ is called the set of passive synchronisation actions.

The semantics for passive synchronisation actions is defined by means of a labelled transition system $LTS_p = (\mathcal{P}, \tilde{\mathcal{A}}, \longrightarrow, P_{init})$ where $\mathcal{P}$ is the set of processes, $\longrightarrow \subset (\mathcal{P} \times \tilde{\mathcal{A}} \times \mathcal{P})$ defines the labelled transition relation, and $P_{init} \in \mathcal{P}$ is the initial process.

The transition rules are presented in the left column of Table 1. The intuition behind these transitions is that they represent the ability to synchronise on the corresponding normal action. The transitions are used in the operational semantics of classical process algebra to support passive synchronisation actions.

The operational semantics of the new process algebra is defined by means of a second labelled transition system $LTS_n = (\mathcal{P}, \bar{\mathcal{A}}, \Longrightarrow, P_{init})$ that uses the same set of processes $\mathcal{P}$. The alphabet is extended by clk and τ, $\bar{\mathcal{A}} = \mathcal{A} \cup \{clk, \tau\}$. $\Longrightarrow \subset (\mathcal{P} \times (\bar{\mathcal{A}} \times \{T, F\}) \times \mathcal{P})$ defines the transition relation.

time determinism

$$\frac{p \overset{clk,T}{\Longrightarrow} p' \quad q \overset{clk,T}{\Longrightarrow} q'}{p + q \overset{clk,T}{\Longrightarrow} p' + q'}$$

time progress

$$\frac{p \overset{clk,T}{\Longrightarrow} p' \quad q \overset{clk,T}{\Longrightarrow} q'}{p \| q \overset{clk,T}{\Longrightarrow} p' \| q'}$$

Table 2
Operational semantics for clock actions

The label of a transition describes the actual progress of the process. It consists of an action and a boolean value T or F. The boolean value indicates whether the action is heard by a listening process. It adds information to the transition relation to simplify verification. The value is contained in the syntactical description of the process and can be deduced from it: only processes that are built from a composition of a sending process and a process ready to synchronise passively can progress with a received action. In this case the boolean value is T.

The deduction rules for operators of classical process algebras can be easily augmented with such a boolean value. In the right column of Table 1 only the additional rules for the parallel composition operator that involve passive synchronisation actions and the rule for the hiding operator are listed, described in the next two paragraphs.

Only composed systems are influenced by passive actions. If one component listens to a normal action both components progress in synchronisation. This is described by rule *active parallel sync*. Otherwise, only the component performing the normal action can progress, described in rule *active parallel nonsync*. It is an important requirement that the normal action is not subject to normal synchronisation requirements; it must not be contained in the synchronisation set. This requirement ensures that the extension is a conservative one.

The hiding rule is mentioned to clarify that τ actions do not reflect whether passive synchronisation has taken place. A corresponding passive rule does not exist. If a passive synchronisation action is hidden the process cannot listen to its environment. Therefore it deadlocks. The τ action is the only invisible action. It is not possible to synchronise on it neither in the classical sense nor with passive synchronisation.

There are two deduction rules for the clock action shown in Table 2. In addition, the *clk* action has to be excluded from those deduction rules of the classical process algebras that would lead to contradictions with these two rules. Furthermore, the *clk* action has no impact on the rules with passive

synchronisation actions because there is no passive action $\widetilde{clk}$.

The transition relation $\Longrightarrow$ is used to define traces of a process in the usual way. It corresponds to the definition in classical process algebras [9].

To simplify the definition of traces a sequence of invisible actions followed by a normal action a, followed by another sequence of invisible actions is combined into one new transition. It is denoted by $\stackrel{\tau^*}{\Longrightarrow}\stackrel{a,M}{\Longrightarrow}\stackrel{\tau^*}{\Longrightarrow}$.

Definition 2.2 (Traces) The set of traces of a process P is defined as the set of all finite sequences consisting of normal actions and clock actions that a process can perform.

$$traces\ (P) = \{tr \in (\mathcal{A} \cup \{clk\})^*_{fin} | n = length(tr) \wedge$$
$$\exists P_1, \ldots, P_n \exists M \in \{T, F\}^n\ \forall i \in \{1, \ldots, n\}\ .\ P_{i-1} \stackrel{\tau^*}{\Longrightarrow}\stackrel{tr_i, M_i}{\Longrightarrow}\stackrel{\tau^*}{\Longrightarrow} P_i\}$$

where $P_0 = P$.

Passive actions have a strong influence on traces of composed processes. It is not true anymore that a trace of a composed process can be separated into the subtraces produced by each component. As an example consider the traces of process $P = a \rightarrow STOP$ and process $Q = b \rightarrow \tilde{a} \rightarrow b \rightarrow STOP$. They are defined as $traces(P) = \{<>, < a >\}$ and $traces(Q) = \{<>, < b >\}$. The second b action in process Q cannot occur because there is no a action the process $\tilde{a} \rightarrow b \rightarrow STOP$ can listen to. However, if P is placed in parallel composition with Q, the composed system can perform b twice if the first b occurs before a. The traces of $P \|_{\{\}} Q$ are $\{<>, < a >, < b >, < a, b >, < b, a >, < b, a, b >\}$.

In the usual way traces define a refinement relation between process by means of subset relation. For example, the process $P \|_{\{\}} Q$ would be an equivalent process in the sense of trace refinement to the process $a \rightarrow b \rightarrow STOP + b \rightarrow a \rightarrow b \rightarrow STOP$ because the processes refine mutually.

3 Property Processes

3.1 Introduction

In [3] the semantics of a property was defined by the set of all infinite traces that satisfy the property. Two classes are distinguishable: Safety properties where it is decidable on the basis of a finite prefix of a trace if the property does not hold, and liveness properties where any finite prefix can be extended to a satisfying trace.

The same semantics is used here. However, the purpose of the property processes is slightly different. Instead of verifying that a system satisfies the properties represented by the processes, here the processes are used to add a new modelling feature: a property process can be seen as an online-diagnostic process. It indicates whether the system in the actual state satisfies a certain property or not.

3.2 Modelling Property Processes

The properties that will be modelled as processes are equivalent to logical formulae of a propositional logic with past operators. The set of atomic propositions covers the actions contained in the alphabet $\mathcal{A}$. A formula can consist of first order logic and two timing operators $\mathcal{P}_{clk}$ and $\mathcal{B}_{clk}$ that describe properties in previous clock cycles. $\mathcal{P}_{clk}\phi$ means that ϕ holds in the previous clock cycle and $\phi\mathcal{B}_{clk}\psi$ means that ϕ holds in every clock cycle back to the cycle where ψ holds.

The grammar for a property ψ is as follows where a denotes an atomic proposition indicating that a certain action $a \in \mathcal{A}$ occurred.

$$\begin{aligned} \psi &= \phi \mid \psi \vee \psi \mid \neg\psi \mid a \mid true \\ \phi &= \mathcal{P}_{clk}\psi \mid \psi\mathcal{B}_{clk}\psi \end{aligned}$$

Now we can define when a trace tr satisfies the property ϕ, written $tr \models \phi$. To do this we reverse the trace. This is necessary because the logic deals with the past of the process. Recall that a trace is finite. Its reversal is denoted by $rev(tr)$.

$$tr \models \phi \qquad \textit{iff} \qquad rev(tr) \models_r \phi$$

The definition of $\models_r$ is as follows:

$$\begin{aligned}
&tr \models_r true \\
&tr \models_r \neg\phi && \text{if } tr \not\models_r \phi \\
&tr \models_r a && \text{if } tr_0 \neq clk \wedge tr_{tail} \models_r a \\
& && \vee tr_0 = a \\
&tr \models_r \phi \vee \psi && \text{if } tr \models_r \phi \vee tr \models_r \psi \\
&tr \models_r \phi \wedge \psi && \text{if } tr \models_r \phi \wedge tr \models_r \psi \\
&tr \models_r \mathcal{P}_{clk}\phi && \text{if } tr_0 \neq clk \wedge tr_{tail} \models_r \mathcal{P}_{clk}\phi \\
& && \vee tr_0 = clk \wedge tr_{tail} \models_r \phi \\
&tr \models_r \phi\mathcal{B}_{clk}\psi && \text{if } tr_0 \neq clk \wedge tr_{tail} \models_r \phi\mathcal{B}_{clk}\psi \\
& && \vee tr_0 = clk \wedge (tr_{tail} \models_r \psi \vee tr_{tail} \models_r \phi \wedge \phi\mathcal{B}_{clk}\psi)
\end{aligned}$$

A property process representing a property in this logic consists of two parts running in parallel. One component enables an appropriate action representing the truth value of the property, that the system model might want to use, the other component evaluates the actual truth value of the property.

The enabling process is a simple process modelling a read-write variable. It can be read by all components of the system and is written to by the evaluating component of the property process. The truth value has to be updated

immediately after every *clk* action. The whole process looks like this:

$$E_\phi = clk \to E^0_\phi$$
$$E^0_\phi = set_\phi true \to E^+_\phi + set_\phi false \to E^-_\phi$$
$$E^+_\phi = read_\phi true \to E^+_\phi + clk \to E^0_\phi$$
$$E^-_\phi = read_\phi false \to E^-_\phi + clk \to E^0_\phi$$

To make sure that the value is set exactly at the beginning of each clock cycle, priority has to be given to $set_\phi value$ over normal actions. Alternatively, the clock action can be divided into two clock actions, clk_{start} and clk_{finish} that all components have to synchronise on. The setting of the values is then performed between these two actions.

The evaluating process uses one-way communication and listens to the actions that appear in the formula, call them relevant actions. At the beginning of each clock cycle the process indicates whether the system satisfies the property or not. An action $set_\phi value$ is performed in synchronisation with the enabling process. This action indicates the truth value of the formula at the very beginning of each clock cycle.

The structure of the process is dependent on the formula. We distinguish two types: formulae whose truth value is determined by actions that were performed in the previous clock cycle (past properties) and formulae whose relevant actions are performed within the actual clock cycle (present properties). The formulae useful to express practical properties are typically past properties with present properties as subformulae.

Formulae of the form $\mathcal{P}_{clk}\phi$ or $\phi\mathcal{B}_{clk}\psi$, and the logical combination of both are the only past properties. These properties are easily translated to processes because the *clk* action defines a state where further actions cannot influence their value. At the time when the truth value is updated the relevant actions were performed in the previous clock cycle. Therefore, the property process can revert to another property process for the formula ϕ in the case of $\mathcal{P}_{clk}\phi$ or on processes for ϕ and ψ in the case of $\phi\mathcal{B}_{clk}\psi$. The relevant actions are the $read_\phi value$ actions for the subformulae ϕ and ψ.

The corresponding processes for $\mathcal{P}_{clk}\phi$ and $\phi\mathcal{B}_{clk}\psi$ are presented below.

$$P(\mathcal{P}_{clk}\phi) = \widetilde{read}_\phi true \to clk \to set_{\mathcal{P}_{clk}\phi} true \to P(\mathcal{P}_{clk}\phi)$$
$$+\widetilde{read}_\phi false \to clk \to set_{\mathcal{P}_{clk}\phi} false \to P(\mathcal{P}_{clk}\phi)$$

$$P(\phi\mathcal{B}_{clk}\psi) = \widetilde{read}_\psi true \to clk \to set_{\phi\mathcal{B}_{clk}\psi} true \to P'(\phi\mathcal{B}_{clk}\psi)$$
$$+\widetilde{read}_\psi false \to clk \to set_{\phi\mathcal{B}_{clk}\psi} false \to P(\phi\mathcal{B}_{clk}\psi)$$
$$P'(\phi\mathcal{B}_{clk}\psi) = \widetilde{read}_{\phi\vee\psi} true \to clk \to set_{\phi\mathcal{B}_{clk}\psi} true \to P'(\phi\mathcal{B}_{clk}\psi)$$
$$+\widetilde{read}_{\phi\vee\psi} false \to clk \to set_{\phi\mathcal{B}_{clk}\psi} false \to P(\phi\mathcal{B}_{clk}\psi)$$

The subformula ϕ, say in property $\mathcal{P}_{clk}\phi$, is a present property because the truth value for ϕ in the actual clock cycle is used, represented by the $\widetilde{read}_\phi value$ actions in the process for $\mathcal{P}_{clk}$. These *read* actions have to be re-

placed by a sequence of actions that captures the value of ϕ in the actual clock cycle in order to get the complete property process $P(\mathcal{P}_{clk}\phi)$. For example, to know whether action a is performed in the actual clock cycle, i.e. when $\phi = a$, the process has to listen to a. Thus, $\widetilde{read}_a\, true$ has to be replaced by $\tilde{a}$. However, the absence of a in the clock cycle is more difficult to prove. The absence of a is only assured if the clk action occurs before any a action. Therefore, $\widetilde{read}_a false$ cannot be replaced by any action, but has to be removed. The occurrence of clk will resolve the choice whether $\phi = a$ is true or false in that clock cycle. Table 3 shows the relevant values of the six basic formulae for $\widetilde{read}_\phi true$ and $\widetilde{read}_\phi false$.

ϕ	$\widetilde{read}_\phi true$	$\widetilde{read}_\phi false$
a	$\tilde{a}$	$-$
$\neg a$	$-$	$\tilde{a}$
$\mathcal{P}_{clk}\phi$	$\widetilde{read}_{\mathcal{P}_{clk}\phi} true$	$\widetilde{read}_{\mathcal{P}_{clk}\phi} false$
$\neg\mathcal{P}_{clk}\phi$	$\widetilde{read}_{\mathcal{P}_{clk}\phi} false$	$\widetilde{read}_{\mathcal{P}_{clk}\phi} true$
$\phi\mathcal{B}_{clk}\psi$	$\widetilde{read}_{\phi\mathcal{B}_{clk}\psi} true$	$\widetilde{read}_{\phi\mathcal{B}_{clk}\psi} false$
$\neg\phi\mathcal{B}_{clk}\psi$	$\widetilde{read}_{\phi\mathcal{B}_{clk}\psi} false$	$\widetilde{read}_{\phi\mathcal{B}_{clk}\psi} true$

Table 3
Relevant actions for basic present properties

For a general subformula ϕ in DNF, i.e. $\bigvee_{i=1\ldots n}(\bigwedge_{j=1\ldots m_i}\phi_{ij})$ where each ϕ_{ij} is a basic present formula, the following definition has to be used for $\widetilde{read}_\phi value$. It recursively defines a process that generates the relevant action sequences. Those sequences ending with label $TRUE$ indicate that the property ϕ holds in the actual clock cycle, those ending with label $FALSE$ describe the conditions when the property fails. The definition is as follows: (I_i describes the index set for the ith conjunction. A hat on top of a parameter means that it is left out.)

$$
\begin{aligned}
&P(I_1,\ldots,I_n) = \\
&\quad \sum_{j\in I_i\neq\{j\}} \widetilde{read}_{\phi_{ij}} true \rightarrow P(I_1,\ldots,I_i\backslash\{j\},\ldots,I_n) \\
&+ \sum_{j\in I_i=\{j\}} \widetilde{read}_{\phi_{ij}} true \rightarrow TRUE \\
&+ \sum_{j\in I_i} \widetilde{read}_{\phi_{ij}} false \rightarrow P(I_1,\ldots,\hat{I}_j,\ldots,I_n)
\end{aligned}
$$

$$P() = FALSE$$

To get the final property process for the formula that uses ϕ as a subformula the label *TRUE*, resp. *FALSE*, has to be replaced by what follows $\widetilde{read}_\phi true$, resp. $\widetilde{read}_\phi false$.

The whole generation process of the property process can be automated. The designer only has to think about the property. As an example, the property process for $\Phi = \mathcal{P}_{clk}(\phi \vee \psi)$ with $\phi = a$ and $\psi = b$ is derived here:

$$\begin{aligned}
P(\Phi) &= P(\mathcal{P}_{clk}(\phi \vee \psi)) = \\
&\quad \tilde{a} \rightarrow clk \rightarrow set_\Phi true \rightarrow P(\Phi) \\
&+ \tilde{b} \rightarrow clk \rightarrow set_\Phi true \rightarrow P(\Phi) \\
&+ \widetilde{read}_a false \rightarrow (\tilde{b} \rightarrow clk \rightarrow set_\Phi true \rightarrow P(\Phi) \\
&\qquad + \widetilde{read}_b false \rightarrow clk \rightarrow set_\Phi false \rightarrow P(\Phi)) \\
&+ \widetilde{read}_b false \rightarrow (\tilde{a} \rightarrow clk \rightarrow set_\Phi true \rightarrow P(\Phi) \\
&\qquad + \widetilde{read}_a false \rightarrow clk \rightarrow set_\Phi false \rightarrow P(\Phi))
\end{aligned}$$

As $\widetilde{read}_a false$ and $\widetilde{read}_b false$ have no corresponding actions the process simplifies to:

$$\begin{aligned}
P(\Phi) &= \\
&\quad \tilde{a} \rightarrow clk \rightarrow set_\Phi true \rightarrow P(\Phi) \\
&+ \tilde{b} \rightarrow clk \rightarrow set_\Phi true \rightarrow P(\Phi) \\
&+ clk \rightarrow set_\Phi false \rightarrow P(\Phi)
\end{aligned}$$

4 Conditionals in Process Algebra

Conditional choice was introduced into process algebra in the context of ACP in [1]. Choice was resolved depending on the value of a propositional logic expression. It is represented by an `if then else` construct and is defined as follows:

Definition 4.1 (If-then-else)

The process $P = \texttt{if}\ \ \phi\ \texttt{then}\ P_1\ \texttt{else}\ P_2$ is defined as $\begin{cases} P_1 & \text{if } \phi \text{ holds} \\ P_2 & \text{otherwise.} \end{cases}$

The formula ϕ can be a formula of any propositional logic. Its truth value is usually determined independently of process P.

However, the logic presented earlier takes the state of the process into account. Therefore, the truth value depends on the state of the process, in particular on the trace of the process that led to P. When an `if then else` construct is used a property process for ϕ is created automatically and placed in parallel composition with the system model. Using the property process

$P(\phi)$ the `if then else` construct can be translated into a normal choice expression:

$$P = \widetilde{read}_\phi true \rightarrow P_1 + \widetilde{read}_\phi false \rightarrow P_2$$

The construction of the property process guarantees that the process P continues as $\begin{cases} P_1 & \text{if } tr \models \phi \\ P_2 & \text{otherwise.} \end{cases}$ where tr is the trace that led to P.

The fact that the system indeed satisfies ϕ when P_1 is chosen, and does not satisfy ϕ when P_2 is chosen can be formally established by constructing an appropriate Kripke structure that represents the relevant behaviour and properties of the system.

5 Example

The example presented in this section illustrates how conditionals can be used with property processes. It describes the specification of a clocked sender component from the model of the alternating bit protocol (ABP). For a detailed discussion of the ABP modelled with process algebras see for example [8]. The example is too small for the new modelling feature to be of real benefit. However, it shows the differences to classical process algebras and makes clear what the advantages are.

The sender can be separated into two components with a well-defined interface (ack_data): one component for sending data and one that deals with its acknowledgement. In the ABP, the data sent is only acknowledged as received correctly if the control bit of the acknowledgement has the same value as the control bit of the data message sent. Otherwise, the same message is re-sent. In the example code the value of the control bit is used as a subscript in the process and action names. The sender process S_0 has sent the control bit 0 and is expecting an acknowledgement rcv_0, and so on. The process S_1 has the same structure as S_0 modulo the value of the control bit; it is therefore left out.

$S_0 = clk \rightarrow$ `if` $\mathcal{P}_{clk} ack\_data_0$ `then`
$\quad send_1 \rightarrow S_1$
`else`
$\quad send_0 \rightarrow S_0$

$R = clk \rightarrow (rcv_0 \rightarrow ack\_data_0 \rightarrow R$
$\quad + rcv_1 \rightarrow ack\_data_1 \rightarrow R$
$\quad + R)$

$Sys = (S_0 \|_{\{\}} R \|_{\{\}} P(\mathcal{P}_{clk} ack\_data_0) \|_{\{\}} P(\mathcal{P}_{clk} ack\_data_1)) \backslash H_0 \cup H_1$
where $H_i = \{ack\_data_i, set_{\mathcal{P}_{clk}\, ack\_data_i}\, value, read_{\mathcal{P}_{clk}\, ack\_data_i}\, value\}$

The specification of a sender in classical process algebra would probably consist of one sequential process; it is much harder to model it as a concurrent system. All possibilities have to be taken into account when synchronisation

between a sending and an acknowledging component has to take place. The specification could look like the following:

$$S_0 = clk \rightarrow (send_0 \rightarrow (rcv_0 \rightarrow S_1 + rcv_1 \rightarrow S_0 + S_0) \\ + rcv_0 \rightarrow send_0 \rightarrow S_0 \\ + rcv_1 \rightarrow send_0 \rightarrow S_0)$$

$$Sys = S_0$$

Using the trace semantics defined in Section 3 it can be shown that both models are equivalent. However, the model in classical process algebra is much harder to maintain. It is more difficult to change the protocol because the aspects of sending and receiving are merged into one process. In contrast, the first model clearly separates the components S and R which makes it easy to replace one component by a new one. This becomes important when several designers are developing different parts of a system or when models should be re-used.

6 Conclusion and Future Work

A new modelling approach for globally clocked systems was presented. It integrates temporal logical formulae and behavioural description in process algebras. The logical formulae are translated into processes that are connected with the system model using one-way communication.

The applicability of this approach for high-level specification was shown by the example of a sender component using the alternating bit protocol. A more complex case study analysing the PI-Bus protocol [10] will follow.

Currently, a refinement method is developed that allows to translate the specification into a new model using pure classical process algebra. The main idea is to replace one-way communication by communication of classical process algebras and to split the states with conditional choice into several states such that only normal choice is used in the model.

As an extension to the presented logic, an operator might be added in the future that describes that an action occurs before another one, but within the same clock cycle. It gives additional expressiveness without increasing complexity.

7 Acknowledgement

I am grateful to my adviser Kerstin Eder for fruitful discussions and beneficial contributions. Thanks are also due to the anonymous referees for their useful feedback.

References

[1] Baeten, J. and J. Bergstra, *Process algebra with signals and conditions*, in: M. Broy, editor, *Programming and Mathematical Method*, 1990 NATO ASi Series F **Proceedings Summer School Marktoberdorf** (1992), pp. 273–323.

[2] Gardiner, P., M. Goldsmith, J. Hulance, D. Jackson, B. Roscoe and B. Scattergood, "Failures-Divergence Refinement, FDR2 User Manual," Formal Systems (Europe) Ltd., 5 edition (2000).

[3] Leuschel, M., T. Massart and A. Currie, *How to make FDR spin – LTL model checking of CSP by refinement*, Lecture Notes of Computer Science **2021** (2001), pp. 99–118.

[4] Milner, R., "Communication and Concurrency," Prentice Hall, New York, 1989.

[5] Nicollin, X. and J. Sifakis, *The algebra of timed processes, ATP: Theory and application*, Information and Computation **114** (1994), pp. 131–178.

[6] Ouaknine, J. and J. Worrell, *Towards specification as refinement in timed systems*, in: *Proceedings of AVoCS 2002*, Birmingham, UK, 2002, pp. 211–225.

[7] Prasad, K., *A calculus of broadcasting systems*, Science of Computer Programming **25** (1995).

[8] Roscoe, A. W., "The theory and practice of concurrency," Prentice Hall, 1998 pp. 128 – 133.

[9] Schneider, S., "Concurrent and Real-time Systems: The CSP Approach," John Wiley & Sons, Ltd., Chichester, England, 2000.

[10] Siemens, *Omi 324: Pi-bus*, Draft standard, Open Microprocessor systems Initiative, Munich, Germany (1994).

Bounded Model Checking for Timed Automata*

Maria Sorea [a,1]

[a] *SRI International, Computer Science Laboratory,*
333 Ravenswood Avenue, Menlo Park, CA 94025, USA
sorea@csl.sri.com

Abstract

Given a timed automaton M, a linear temporal logic formula φ, and a bound k, bounded model checking for timed automata determines if there is a falsifying path of length k to the hypothesis that M satisfies the specification φ. This problem can be reduced to the satisfiability problem for Boolean constraint formulas over linear arithmetic constraints. We show that bounded model checking for timed automata is complete, and we give lower and upper bounds for the length k of counterexamples. Moreover, we define bounded model checking for networks of timed automata in a compositional way.

1 Introduction

Timed automata [4] are state-transition graphs augmented with a finite set of real-valued clocks. The clocks proceed at a uniform rate and constrain the times at which transitions may occur. Given a timed automaton and a property expressed in a timed logic such as TCTL [3] or T_μ [18], model checking answers the question whether the timed automaton satisfies the given formula. The fundamental graph-theoretic model checking algorithm by Alur, Courcoubetis and Dill [3] constructs a finite quotient, the so-called *region graph*, of the infinite state graph. Algorithms directly based on the explicit construction of such a partition are however unlikely to perform efficiently in practice, since the number of equivalence classes of states of the region graph grows exponentially with the largest time constant and the number of clocks that are used to specify timing constraints. Symbolic model checking algorithms are obtained by characterizing regions as Boolean combinations of linear inequalities over

* This research was supported by the National Science Foundation under grants CCR-00-82560 and CCR-00-86096.

[1] Also affiliated with University of Ulm, Germany.

This is a preliminary version and considered for the final proceedings, to be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

clocks [18]. Based on these algorithms, tools for verifying timed automata, such as, for example Uppaal [22], Kronos [11], HyTech [17], Tempo [30], have been developed.

The technique of bounded model checking has been recently introduced [8], as an alternative to classical model checking. Given a system M modeled as a state machine, a temporal logic specification φ, and a bound k, the bounded model checking (BMC) problem consists in searching for counterexamples of length k to the model checking problem $M \models \varphi$. The BMC problem for finite state models can be reduced to a propositional satisfiability problem, and off-the-shelf propositional satisfiability (SAT) checkers are used to construct counterexamples from satisfying assignments to the propositional variables. It has been demonstrated that BMC is in many cases more effective in falsifying designs than traditional model checking techniques [8,9]. In [13] the BMC paradigm has been extended to programs over infinite state space, and LTL formulas augmented with a decidable set of constraints. For an infinite state system M, a linear temporal logic formula with constraints φ, and a bound k, a Boolean constraint formula, $[\![M, \varphi]\!]_k$, can be constructed that is satisfiable if and only if there is a counterexample of length k for the model checking problem $M \models \varphi$. BMC for infinite state systems is sound, and for invariant properties also complete, but incomplete for the entire LTL logic [13].

The main contribution here is to show that BMC for timed automata is indeed complete for all LTL formulas with clock constraints. We describe how a timed automaton can be directly encoded into a Boolean constraint formula, without constructing the corresponding region graph. Our approach is compositional in that Boolean constraint formulas encoding networks of timed automata can be obtained by Boolean combinations of the encoding of the components. This compositional approach reduces the size of the generated formula considerably. Moreover, we give bounds for the length k of counterexamples for the model checking problem $M \models \varphi$ that depend on the size of the LTL formula φ and the size of the region graph corresponding to the given timed automaton M.

The paper is structured as follows. In Section 2 we review the basic notions of timed automata. Section 3 presents the details of BMC for timed automata together with the completeness results. Lower and upper bounds for the length k of counterexamples are given. Section 4 illustrates BMC for networks, that is, parallel composition of timed automata, and shows how complex systems can be encoded into a Boolean constraint formula in a compositional way, without first computing the product automaton of the components. Finally, in Section 5 we present some experimental results using train gate controller and Fischer's mutual exclusion protocol as benchmarks, and draw conclusions.

2 Timed Automata

We review some basic notions of transition systems and timed automata. Timed automata, as introduced by Alur, Courcoubetis, and Dill [3], are state-transition graphs augmented with a finite set of real-valued clocks. Given a set of clock variables (or simply *clocks*) $Cl = \{x_1, \ldots, x_n\}$, a clock-valuation function $v : Cl \to \mathbb{R}_0^+$ assigns a (positive) real value to each clock. Clock constraints compare clock values with rational constants. Given a set Cl of clock variables, x_1, x_2 arbitrary clocks, $\gamma \in \mathbb{Q}_0^+$, and $\sim \in \{\leq, \geq, <, >, =\}$, the set Φ of *clock (or timing) constraints* over Cl is defined by the grammar

$$g := \mathtt{tt} \mid \mathtt{ff} \mid x_1 \sim \gamma \mid x_1 - x_2 \sim \gamma \mid g_1 \wedge g_2.$$

For a positive integer c, $\Phi(c)$ is the finite subset of all timing constraints $x \sim \gamma$, $x - y \sim \gamma$, where $x, y \in Cl$, $\sim \in \{<, \leq, =, \geq, >\}$ and $\gamma \in \{0, \ldots, c\}$. Clock constraints over Cl are interpreted with respect to clock-valuation functions $v : Cl \to \mathbb{R}_0^+$. For a clock-valuation function v and a clock constraint g over Cl, we write $v \models\!\!\approx g$ (to be read as "*v satisfies g*") to denote that according to the values given by v the constraint g evaluates to true. Formally, $v \models\!\!\approx g$ is defined inductively over the syntactic structure of g, where $x_1, x_2 \in Cl$ are arbitrary clocks, $\gamma \in \mathbb{Q}_0^+$, and $\sim \in \{\leq, \geq, <, >, =\}$:

$$v \not\models\!\!\approx \mathtt{ff} \qquad v \models\!\!\approx \mathtt{tt} \qquad v \models\!\!\approx x_1 - x_2 \sim \gamma \text{ iff } v(x_1) - v(x_2) \sim \gamma$$

$$v \models\!\!\approx x_1 \sim \gamma \text{ iff } v(x_1) \sim \gamma \qquad v \models\!\!\approx g_1 \wedge g_2 \text{ iff } v \models\!\!\approx g_1 \text{ and } v \models\!\!\approx g_2$$

For $\delta \in \mathbb{R}_0^+$, $v + \delta$ denotes the clock valuation that maps each clock $x \in Cl$ to the value $v(x) + \delta$. For a clock $x \in Cl$, $v[r := 0]$ denotes the clock valuation for Cl that maps the clocks in r to the value 0 and leaves all the other clock values unchanged.

A *timed automaton* $\mathcal{S}$ is a tuple $\langle L, l_0, Cl, E, Inv \rangle$, where L is a nonempty finite set of locations, $l_0 \subseteq L$ is the initial location, and Cl is a finite set of clocks. $Inv : L \to \Phi$ assigns a set of downward closed clock constraints to each location L; the elements of $Inv(l)$ are the *invariants* for location l. $E \subseteq L \times \mathcal{P}(\Phi) \times \mathcal{P}(Cl) \times L$ is a finite set of edges. An edge $e = \langle l, g, r, l' \rangle$ represents a transition from location l to location l'. A transition may only be fired if the timing constraint (guard of the transition) g holds with respect to the current value of the clocks, and if the invariant of the target location is satisfied with respect to the modified value of the clocks. Firing a transition does not only change the current location but also resets the clocks in r to 0.

A timed automaton with three locations l_0, l_1, l_2 and two clocks x, y is displayed in Figure 1. The initial location is l_0, transitions are decorated with both timing constraints and clock resets such as $x := 0$. The invariant for location l_0 is $y \leq 1$. Timing constraints that are *true* are omitted.

Alur, Courcoubetis, and Dill [3] introduce the fundamental notion of clock regions, which partition the space of possible clock evaluation for a timed automaton into finitely many regions. For a timed automaton $\mathcal{S}$ with clocks Cl

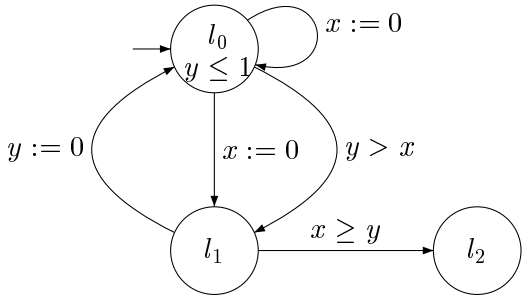

Fig. 1. Example of a timed automaton (the *simple* example).

and largest constant c, occurring in any timing constraint of $\mathcal{S}$, a *clock region* is a set χ of clock valuations, such that for all timing constraints $g \in \Phi(c)$ and for any two $v_1,\ v_2 \in \chi$ it is the case that $v_1 \models g$ if and only if $v_2 \models g$. In this case we write $v_1 \equiv_{\mathcal{S}} v_2$. We will use $[v]$ to denote the clock region to which v belongs.

A *state* of a timed automaton $\mathcal{S}$ is a pair (l, v) where $l \in L$ is a location of $\mathcal{S}$ and v a clock valuation for Cl. An *initial state* is of the form (l_0, v_0) where l_0 denotes the initial state of $\mathcal{S}$ and v_0 maps all clocks in Cl to 0. We extend the satisfiability relation for clock constraints on states, as follows: for a state (l, v) and a timing constraint g, $(l, v) \models g$ iff $v \models g$. A *timed step* is either a *delay step*, where time advances by some positive real-valued δ, or an instantaneous *state transition step*. For a timed automaton $\mathcal{S} = \langle L, l_0, Cl, E, Inv\rangle$, and $\delta \geq 0$, we say that the state $(l, v + \delta)$ is obtained from (l, v) by a *delay step* $(l, v) \xrightarrow{\delta} (l, v + \delta)$, if the invariant constraint $v + \delta \models Inv(l)$ holds. A *state transition step* $(l, v) \xrightarrow{g,r} (l', v')$ occurs if there exists an edge $\langle l, g, r, l'\rangle$, and $v \models g$, $v' = v[r := 0]$, and $v' \models Inv(l')$. The union of delay and state transition steps defines the timed transition relation $\Rightarrow$ of a timed automaton $\mathcal{S}$. Now, a *path* π is an infinite sequence of states $(l_0, v_0), (l_1, v_1), \ldots$ such that $(l_i, v_i) \Rightarrow (l_{i+1}, v_{i+1}), \forall i \geq 0$.

3 System Verification

Given a BMC problem for a timed automaton, an LTL formula with linear arithmetic constraints, and a bound of the length of counterexamples to be searched for, we describe a sound and complete reduction to the satisfiability problem of Boolean constraint formulas. The encoding of the transition relations of the given automaton follows the now-standard approach already taken in [18]. Whereas in [8,5,27] LTL formulas are translated directly into propositional formulas, we use Büchi automata for this encoding. This simplifies substantially the notations and the proofs, but a direct translation can sometimes be more succinct in the number of variables needed. We use the common notions for finite automata over finite and infinite words, and we assume as

given a theory $\mathcal{C}$ of linear arithmetic constraints with a satisfiability solver. This theory includes the clock constraints Φ, and difference constraints of the form $x' - x = y' - y$, or $x' = x + \delta$ where $x, x', y, y' \in Cl$ are clock variables, and δ a positive real valued variable. Pratt observed that most inequalities in program verification are of the form $x - y \leq c$, where c is constant. Given a conjunction C of such constraints, satisfiability of C can be decided using the Bellman-Ford algorithm in time quadratic to the number of variables in C. Shostak's [28] loop residue algorithm generalizes Pratt's results to arbitrary linear inequalities. For the simplicity of the presentation we consider only timed automata that are nonzeno. Nonzenoness can be guaranteed, for example, by restricting the model of timed automata to certain delay steps, as illustrated in [24].

In order to make this paper as self-contained as possible, we recall some notions and definitions from [13]. Consider a set $V := \{x_1, \ldots, x_n\}$ of variables interpreted over nonempty domains $\mathcal{D}_1$ through $\mathcal{D}_n$, together with a type assignment τ such that $\tau(x_i) = \mathcal{D}_i$. For a set of typed variables V, a *variable assignment* is a function ν from variables $x \in V$ to an element of $\tau(x)$. The variables in $V := \{x_1, \ldots, x_n\}$ are also called *state variables*, and a *program state* is a variable assignment over V. A pair $\langle I, T\rangle$ is a *$\mathcal{C}$-program* over V if $I \in \mathsf{Bool}(\mathcal{C}(V))$ and $T \in \mathsf{Bool}(\mathcal{C}(V \cup V'))$, where V' is a primed, disjoint copy of V. V denotes the current state variables, while V' states for the next state variables. I is used to restrict the set of initial program states, and T specifies the transition relation between states and their successor states. The set of $\mathcal{C}$-programs over V is denoted by $\mathsf{Prg}(\mathcal{C}(V))$. The semantics of a program P is given in terms of a *transition system* M in the usual way, and, by a slight abuse of notation, we sometimes write M for both the program and its associated transition system.

A timed automaton $\mathcal{S} = \langle L, l_0, Cl, E, Inv\rangle$ can easily be described in terms of a program with linear arithmetic constraints over state variables $V = \{at, x_1, \ldots, x_n\}$, where at is interpreted over the set L of locations and the clock variables $x_1, \ldots, x_n \in Cl$ are interpreted over $\mathbb{R}_0^+$.

Definition 3.1 Given a timed automaton $\mathcal{S} = \langle L, l_0, Cl, E, Inv\rangle$ with $Cl = \{x_1, \ldots, x_n\}$ the set of clocks. $\mathcal{S}$ can be defined as a $\langle I, T\rangle$ program in $\mathsf{Prg}(\mathcal{C}(V))$ over the set $V = \{at, x_1, \ldots, x_n\}$, and $V' = \{at', x_1', \ldots, x_n'\}$ as follows.

- Definition of the initial state

$$I := (at = l_0 \ \wedge \ x_1 = 0 \ \wedge \ \ldots \ \wedge \ x_n = 0).$$

- Definition of a state transition step corresponding to $e = \langle l, g, r, l'\rangle \in E$

$$T(e) := (at = l \ \wedge \ g \ \wedge \ x_1' = z_1 \ \wedge \ \ldots \ \wedge \ x_n' = z_n \ \wedge \ at' = l' \ \wedge \ Inv(l')(x_1', \ldots, x_n'))$$

where $z_i = 0$ if $x_i \in r$; otherwise $z_i = x_i$. The state formula $Inv(l')(x_1', \ldots, x_n')$ is obtained from the invariant of location l', $Inv(l')$, by replacing the vari-

ables $x_1, \ldots, x_n$ in the constraints of $Inv(l')$ by the primed variables $x'_1, \ldots, x'_n$.

- Definition of delay steps ($Inv(\mathcal{S})$ is the set of all locations that have an invariant different from *true*.)

$$\begin{aligned} D := \exists \delta \geq 0.\ (& \bigwedge_{l \in Inv(\mathcal{S})} (at = l \Rightarrow Inv(l)(x'_1, \ldots, x'_n)) \\ & \wedge\ (at' = at) \\ & \wedge\ (x'_1 = x_1 + \delta)\ \wedge\ \ldots\ \wedge\ (x'_n = x_n + \delta)). \end{aligned}$$

- Definition of the transition relation T

$$T := \bigvee_{e \in E} T(e)\ \bigvee\ D.$$

The timed automaton depicted in Figure 1, for example, is expressed in terms of the program $\langle I, T \rangle$ over state variables $V = \{at, x, y\}$, and $V' = \{at', x', y'\}$, where at and at' are interpreted over the set of locations $\{l_0, l_1, l_2\}$, and the clock variables x, y, x', y' are interpreted over $\mathbb{R}_0^+$. Initially, the program is in location l_0 and the value of the clocks x, y is equal to 0. The transitions are encoded by a conjunction of constraints over the current state variables at, x, y and the next state variables at', x', y'.

$$\begin{aligned} I(at, x, y) := {} & (at = l_0\ \wedge\ x = 0\ \wedge\ y = 0) \\ T(at, x, y, at', x', y') := {} & (at = l_0\ \wedge\ x' = 0\ \wedge\ y' = y\ \wedge\ at' = l_0\ \wedge\ y' \leq 1)\ \vee \\ & (at = l_0\ \wedge\ x' = 0\ \wedge\ y' = y\ \wedge\ at' = l_1)\ \vee \\ & (at = l_0\ \wedge\ y > x\ \wedge\ x' = x\ \wedge\ y' = y\ \wedge\ at' = l_1)\ \vee \\ & (at = l_1\ \wedge\ y' = 0\ \wedge\ x' = x\ \wedge\ at' = l_0)\ \vee \\ & (at = l_1\ \wedge\ x \geq y\ \wedge\ x' = x\ \wedge\ y' = y\ \wedge\ at' = l_2)\ \vee \\ & D(at, x, y, at', x', y') \end{aligned}$$

The delay steps are encoding as

$$\begin{aligned} & D(at, x, y, at', x', y') = \\ & \quad \exists \delta \geq 0.\ ((at = l_0 \Rightarrow y' \leq 1) \wedge (at' = at) \wedge (x' = x + \delta) \wedge (y' = y + \delta)). \end{aligned}$$

The above formula is not contained in $\mathsf{Bool}(\mathcal{C})$, since the definition of D contains an existential quantifier, but the existential quantifier can easily be eliminated.

$$\begin{aligned} & D(at, x, y, at', x', y')\ := \\ & \quad ((at = l_0 \Rightarrow y' \leq 1)\ \wedge\ (x' - x \geq 0)\ \wedge\ (y' - y = x' - x) \wedge\ (at' = at)). \end{aligned}$$

Instead of using 4 clock variables, this formula can be als expressed using 3 variables as follows:

$$\begin{aligned} & D(at, x, y, \delta, at', x', y')\ := \\ & \quad ((at = l_0 \Rightarrow y' \leq 1)\ \wedge\ \delta \geq 0\ \wedge\ x' = x + \delta\ \wedge\ y' = y + \delta\ \wedge\ (at' = at)). \end{aligned}$$

This fact will be used for the compositional encoding of networks of timed automata.

The formulas of the *constraint linear temporal logic* $\mathsf{LTL}(\Phi)$ are linear-time temporal logic formulas with the usual "until" and "release" operators, and constraints $c \in \Phi$ as atoms.

$$\varphi ::= \mathit{true} \mid \mathit{false} \mid c \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \,\mathbf{U}\, \varphi_2 \mid \varphi_1 \,\mathbf{R}\, \varphi_2$$

The formula $\varphi_1 \,\mathbf{U}\, \varphi_2$ holds on a path π if there is a state on the path where φ_2 holds, and at every preceding state on the path φ_1 holds. The release operator $\mathbf{R}$ is the logical dual of $\mathbf{U}$. It requires that φ_2 holds along the path up to and including the first state, where φ_1 holds. However, φ_1 is not required to hold eventually. The derived operators $\mathbf{F}\,\varphi = \mathit{true}\,\mathbf{U}\,\varphi$ and $\mathbf{G}\,\varphi = \mathit{false}\,\mathbf{R}\,\varphi$ denote "eventually φ" and "globally φ". Our logic does not contain a next-step operator. The main interest in removing the next-step operator stems from the fact that we do not want to distinguish between one delay step of duration, say, 1 and two subsequent delay steps of durations 2/5 and 3/5, since these traces are considered to be observationally equivalent. Logics without explicit next-step operator have also been considered, for example, by Alur [1], Henzinger, Nicollin, Sifakis and Yovine [18], and by Dams [10].

Given a program $M \in \mathsf{Prg}(\mathcal{C})$ and a path π in M, the satisfiability relation $M, \pi \models \varphi$ for an $\mathsf{LTL}(\Phi)$ formula φ is given in the usual way with the notable exception of the case of constraint formulas c. In this case, $M, \pi \models c$ if and only if c holds in the start state of π. Assuming the notation above, the *$\mathcal{C}$-model checking problem* $M \models \varphi$ holds iff for all paths $\pi = s_0, s_1, \ldots$ in M with $s_0 \in I$ it is the case that $M, \pi \models \varphi$.

The following lemma states that the logic $\mathsf{LTL}(\Phi)$ preserves bisimulation. The proof is by induction over the syntax of $\mathsf{LTL}(\Phi)$.

Lemma 3.2 Given a program M with a finite bisimulation M' (i.e. $M \approx M'$), and a formula $\varphi \in \mathsf{LTL}(\Phi)$; then $M \models \varphi$ iff $M' \models \varphi$.

Now, given a bound k, a program $M \in \mathsf{Prg}(\mathcal{C}(V))$ and a formula $\varphi \in \mathsf{LTL}(\Phi)$ we consider the problem of constructing a formula $[\![\, M, \varphi \,]\!]_k \in \mathsf{Bool}(\mathcal{C}(V))$, which is satisfiable if and only if there is a counterexample of length k for the $\mathcal{C}$-model checking problem $M \models \varphi$. This construction proceeds as follows.

(i) Definition of $[\![\, M \,]\!]_k$ as the unfolding of the program M up to step k from initial states (this requires k disjoint copies of V).

(ii) Translation of $\neg\varphi$ into a corresponding Büchi automaton $\mathcal{B}_{\neg\varphi}$ whose language of accepting words consists of the satisfying paths of $\neg\varphi$.

(iii) Encoding of the transition system for $\mathcal{B}_{\neg\varphi}$ and the Büchi acceptance condition as a Boolean formula, say $[\![\, \mathcal{B} \,]\!]_k$.

(iv) Forming the conjunction $[\![\, M, \varphi \,]\!]_k := [\![\, \mathcal{B} \,]\!]_k \wedge [\![\, M \,]\!]_k$.

(v) A satisfying assignment for the formula $[\![\, M, \varphi \,]\!]_k$ induces a counterexample of length k for the model checking problem $M \models \varphi$.

Definition 3.3 [Encoding of *$\mathcal{C}$-Programs*] The encoding $[\![\, M \,]\!]_k$ of the kth unfolding of a *$\mathcal{C}$-program* $M = \langle I, T \rangle$ in $\mathsf{Prg}(\mathcal{C}(\{x_1, \ldots, x_n\}))$ is given by the

Boolean constraint formula $[\![M]\!]_k$.

$$
\begin{aligned}
I_0(x[0]) &:= I\langle \{x_i \mapsto x_i[0] \mid x_i \in V\}\rangle \\
T_j(x[j], x[j+1]) &:= T\langle \{x_i \mapsto x_i[j] \mid x_i \in V\} \cup \{x_i' \mapsto x_i[j+1] \mid x_i \in V\}\rangle \\
[\![M]\!]_k &:= I_0(x[0]) \wedge \bigwedge_{j=0}^{k-1} T_j(x[j], x[j+1])
\end{aligned}
$$

where $\{x_i[j] \mid 0 \leq j \leq k\}$ is a family of typed variables for encoding the state of variable x_i in the jth step, $x[j]$ is used as an abbreviation for $x_1[j] \ldots, x_n[j]$, and $T\langle x_i \mapsto x_i[j]\rangle$ denotes simultaneous substitution of the x_i by $x_i[j]$ in formula T.

A two-step unfolding of the *simple* program in Figure 1, for example, is encoded by $[\![simple]\!]_2 := I_0 \wedge T_0 \wedge T_1$ $(*)$.

$$
\begin{aligned}
I_0 := {} & (at[0] = l_0 \wedge x[0] = 0 \wedge y[0] = 0) \\
T_0 := {} & (at[0] = l_0 \wedge x[1] = 0 \wedge y[1] = y[0] \wedge at[1] = l_0 \wedge y[1] \leq 1) \vee \\
& (at[0] = l_0 \wedge x[1] = 0 \wedge y[1] = y[0] \wedge at[1] = l_1) \vee \\
& (at[0] = l_0 \wedge y[0] > x[0] \wedge x[1] = x[0] \wedge y[1] = y[0] \wedge at[1] = l_1) \vee \\
& (at[0] = l_1 \wedge y[1] = 0 \wedge x[1] = x[0] \wedge at[1] = l_0) \vee \\
& (at[0] = l_1 \wedge x[0] \geq y[0] \wedge x[1] = x[0] \wedge y[1] = y[0] \wedge at[1] = l_2) \vee \\
& ((at[0] = l_0 \Rightarrow y[1] \leq 1) \wedge (x[1] - x[0] \geq 0) \wedge \\
& \quad (y[1] - y[0] = x[1] - x[0]) \wedge (at[1] = at[0])) \\
T_1 := {} & (at[1] = l_0 \wedge x[2] = 0 \wedge y[2] = y[1] \wedge at[2] = l_0 \wedge y[2] \leq 1) \vee \\
& (at[1] = l_0 \wedge x[2] = 0 \wedge y[2] = y[1] \wedge at[2] = l_1) \vee \\
& (at[1] = l_0 \wedge y[1] > x[1] \wedge x[2] = x[1] \wedge y[2] = y[1] \wedge at[2] = l_1) \vee \\
& (at[1] = l_1 \wedge y[2] = 0 \wedge x[2] = x[1] \wedge at[2] = l_0) \vee \\
& (at[1] = l_1 \wedge x[1] \geq y[1] \wedge x[2] = x[1] \wedge y[2] = y[1] \wedge at[2] = l_2) \vee \\
& ((at[1] = l_0 \Rightarrow y[2] \leq 1) \wedge (x[2] - x[1] \geq 0) \wedge \\
& \quad (y[2] - y[1] = x[2] - x[1]) \wedge (at[2] = at[1]))
\end{aligned}
$$

The translation of linear temporal logic formulas into a corresponding Büchi automaton is well-studied in the literature (for example, [16]) and does not require additional explanation. Notice however, that, the translation of $\mathsf{LTL}(\Phi)$ formulas yields Büchi automata with $\mathcal{C}$-*constraints* as labels. Both the resulting transition system and the bounded acceptance test based on the detection of reachable cycles with at least one final state can easily be encoded as Boolean constraint formulas [13].

Definition 3.4 [Encoding of Büchi Automata] Let $V = \{x_1, \ldots, x_n\}$ be a set of typed variables, $\mathcal{B} = \langle \Sigma, Q, \Delta, Q^0, F\rangle$ be a Büchi automaton with labels Σ in $\mathsf{Bool}(\mathcal{C})$, and pc be a variable (not in V), which is interpreted over the finite set of locations Q of the Büchi automaton. For a given integer k, we obtain, as in Definition 3.3, families of variables $x_i[j]$, $pc[j]$ $(1 \leq i \leq n,\ 0 \leq j \leq k)$ for representing the jth state of $\mathcal{B}$ in a run of length k. Furthermore, the

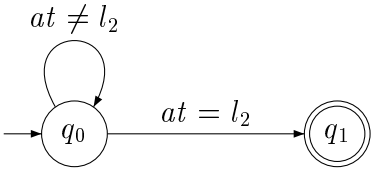

Fig. 2. Automaton for $\mathbf{F}\,(at = l_2)$.

transition relation of $\mathcal{B}$ is encoded in terms of the *C-program* $\mathcal{B}_M$ over the set of variables $\{pc\} \cup V$, and $[\![\,\mathcal{B}_M\,]\!]_k$ denotes the encoding of this program as in Definition 3.3. Now, given an encoding of the acceptance condition

$$acc(\mathcal{B})_k \;:=\; \bigvee_{j=0}^{k-1} \Big(pc[k] = pc[j] \wedge \bigwedge_{v=1}^{n} x_v[k] = x_v[j] \wedge \Big(\bigvee_{l=j+1}^{k} \bigvee_{f \in F} pc[l] = f \Big) \Big)$$

the k-th unfolding of $\mathcal{B}$ is defined by $[\![\,\mathcal{B}\,]\!]_k := [\![\,\mathcal{B}_M\,]\!]_k \wedge acc(\mathcal{B})_k$. The acceptance condition for Büchi automata requires that some final state appears on a run infinitely often. This is encoded by the formula $acc(\mathcal{B})_k$. The first 2 conjuncts $pc[k] = pc[j]$ and $\bigwedge_{v=1}^{n} x_v[k] = x_v[j]$ describe the presence of a cycle in the run between states j and k, while $\bigvee_{l=j+1}^{k} \bigvee_{f \in F} pc[l] = f$ guarantees that inside the cycle, that is, between state $j+1$ and state k, there is at least one final state contained.

An $\mathsf{LTL}(\Phi)$ formula is said to be **R**-free (**U**-free) iff there is an equivalent formula (in negation normal form) not containing the operator **R** (**U**). Note that **U**-free formulas correspond to the notion of *syntactic safety formulas* [19,29]. Now, it can be directly observed from the semantics of $\mathsf{LTL}(\Phi)$ formulas that every **R**-free formula can be translated into an automaton over finite words that accepts a prefix of all infinite paths satisfying the given formula.

Definition 3.5 Given an automaton $\mathcal{B}$ over finite words and the notation as in Definition 3.4, the encoding of the k-ary unfolding of $\mathcal{B}$ is given by $[\![\,\mathcal{B}_M\,]\!]_k \wedge acc(\mathcal{B})_k$ with the acceptance condition

$$acc(\mathcal{B})_k \;:=\; \bigvee_{j=0}^{k} \bigvee_{f \in F} pc[j] = f \;.$$

Consider the problem of finding a counterexample of length $k = 2$ to the hypothesis that our running example in Figure 1 satisfies $\mathbf{G}\,\neg(at = l_2)$, that is, the timed automaton never reaches location l_2. The negated property $\mathbf{F}\,(at = l_2)$ is an **R**-free formula, and the corresponding automaton $\mathcal{B}$ over finite words is displayed in Figure 2. This automaton is translated, according to Definition 3.5, into the formula

$$[\![\,\mathcal{B}\,]\!]_2 := I_0(\mathcal{B}) \wedge T_0(\mathcal{B}) \wedge T_1(\mathcal{B}) \wedge acc(\mathcal{B})_2 \;. \qquad (**)$$

The variables $pc[j]$ and $at[j]$ $(j = 0, 1, 2)$ are used to represent the first three states in a run.

$$I_0(\mathcal{B}) := (pc[0] = q_0)$$
$$T_0(\mathcal{B}) := (pc[0] = q_0 \wedge \neg(at[0] = l_2) \wedge pc[1] = q_0) \;\vee$$

$$
\begin{aligned}
& (pc[0] = q_0 \wedge at[0] = l_2 \wedge pc[1] = q_1) \\
T_1(\mathcal{B}) := & (pc[1] = q_0 \wedge \neg(at[1] = l_2) \wedge pc[2] = q_0) \ \vee \\
& (pc[1] = q_0 \wedge at[1] = l_2 \wedge pc[2] = q_1) \\
acc(\mathcal{B})_2 := & (pc[0] = q_1 \vee pc[1] = q_1 \vee pc[2] = q_1)
\end{aligned}
$$

The bounded model checking problem $[\![\, simple \,]\!]_2 \wedge [\![\, \mathcal{B} \,]\!]_2$ for the *simple* program is obtained by conjoining the formulas $(*)$ and $(**)$. Using the BMC procedure over linear arithmetic constraints one finds the counterexample

$$(l_0, x = 0, y = 0) \rightarrow (l_1, x = 0, y = 0) \rightarrow (l_2, x = 0, y = 0)$$

of length 2. Counterexamples for timed property, such as $\mathbf{G}\,(at = l_1 \Rightarrow x > y)$, can also be found by the BMC procedure.

The following two theorems are stated in [13].

Theorem 3.6 (Soundness) Let $M \in \mathsf{Prg}(\mathcal{C})$ and $\varphi \in \mathsf{LTL}(\Phi)$. If there exists a natural number k such that $[\![\, M, \varphi \,]\!]_k$ is satisfiable, then $M \not\models \varphi$.

Theorem 3.7 (Completeness for Finite State Systems) Let M be a *$\mathcal{C}$-program* with a finite set of reachable states, φ be an $\mathsf{LTL}(\Phi)$ formula, and k be a given bound; then: $M \not\models \varphi$ implies $\exists k \in I\!N.\, [\![\, M, \varphi \,]\!]_k$ is satisfiable.

In general, BMC over infinite domains is not complete. Consider, for example, the model checking problem $M \models \varphi$ for the program $M = \langle I, T \rangle$ over the variable $V = \{x\}$ with $I = (x = 0)$ and $T = (x' = x + 1)$ and the formula $\varphi = \mathbf{F}\,(x < 0)$. M can be seen as a one-counter automaton, where initially the value of the counter x is 0, and with every transition the value of x is increased with 1. Obviously, it is the case that $M \not\models \varphi$, but there exists no $k \in I\!N$, such that the formula $[\![\, M, \varphi \,]\!]_k$ is satisfiable. Since $\neg\varphi$ is not an $\mathbf{R}$-free formula, the encoding of the Büchi automaton $\mathcal{B}_k$ must contain, by Definition 3.4 a finite accepting cycle, described by $pc[k] = pc[0] \wedge x[k] = x[0]$ or $pc[k] = pc[1] \wedge x[k] = x[1]$ etc. Such a cycle, however, does not exist, since the program M contains only one noncycling, infinite path, where the value of x increases in every step, that is $x[i+1] = x[i] + 1$, for all $i \geq 0$.

Theorem 3.8 (Completeness for Timed Automata) Let M be a timed automaton defined as a *$\mathcal{C}$-program* over a set of state variables $V = \{x_1, \ldots, x_n\}$, and φ be a formula in $\mathsf{LTL}(\Phi)$; then:

$$M \not\models \varphi \text{ implies } \exists k.\, [\![\, M, \varphi \,]\!]_k \text{ is satisfiable.}$$

Proof. Let M' be the finite region graph corresponding to M, also defined as a $\mathcal{C}$-program over the set of state variables V. From $M \not\models \varphi$, it follows by Lemma 3.2, that $M' \not\models \varphi$. Let

$$[\![\, M', \varphi \,]\!]_k := [\![\, \mathcal{B} \,]\!]_k \wedge [\![\, M' \,]\!]_k$$

be the bounded model checking problem for M' and φ. Since M' is finite, by Theorem 3.7 there exists a k such that $[\![\, M', \varphi \,]\!]_k$ is satisfiable. It remains to show, that if $[\![\, M', \varphi \,]\!]_k$ is satisfiable then also $[\![\, M, \varphi \,]\!]_k$ is satisfiable. From

$[\![M', \varphi]\!]_k$ satisfiable it follows that $[\![M']\!]_k$ and $[\![\mathcal{B}]\!]_k$ are satisfiable. By Definition 3.3

$$[\![M']\!]_k := I'_0(x[0]) \wedge \bigwedge_{j=0}^{k-1} T'_j(x[j], x[j+1])$$

where the state formula $I'_0(x[0])$ encodes the initial state $(l_0, [v_0])$, and the formula $T'_j(x[j], x[j+1])$ defines the transition relation. Obviously, the formula $I'_0(x[0])$ is equivalent to the state formula $I_0(x[0])$, which describes the initial state (l_0, v_0) of the program M. Let $\pi' = s'_0, s'_1, \ldots, s'_{k-1}$, where $s'_i = (l'_i, [v'_i])$ be a k-path in M'. In [31] it has been shown that the region equivalence is a bisimulation relation. Since M and M' are bisimilar, it follows that there exists a k-path $\pi = s_0, s_1, \ldots, s_{k-1}$ in M, where $s_i = (l_i, v_i)$ such that $l_i = l'_i$ and $v_i \in [v'_i]$. Similarly to the unfolding of M', M can be unfold up to step k to make $[\![M]\!]_k$ and $[\![M']\!]_k$ equisatisfiable. □

Lower bounds for the length k of counterexamples can be found by examining the structure of the Büchi automaton for a given $\mathsf{LTL}(\Phi)$ formula. A lower bound is given by the length of the shortest path from the initial state to a final/accepting state of the automaton. For a timed automaton M with c the largest constant appearing in the guards and invariants of M, and t the number of clocks, an upper bound for k is given by $k \leq n \cdot 2^{O(t \log(ct))} \cdot 2^{O(|\varphi|)}$, where n is the number of locations of M and $n \cdot 2^{O(t \log(ct))}$ the number of states in the region graph of M [2].

Corollary 3.9 Let M be a timed automaton with c the largest constant appearing in the guards and invariants of M, and t the number of clocks. Further, let φ be a formula in $\mathsf{LTL}(\Phi)$. If $k = n \cdot 2^{O(t \log(ct))} \cdot 2^{O(|\varphi|)}$ then $M \models \varphi$ iff $[\![M, \varphi]\!]_j$ is unsatisfiable for all $j \leq k$.

4 BMC for Networks of Timed Automata

Complex systems are modeled as *networks of timed automata*, that is, parallel composition of timed automata. Given two timed automata A_1 and A_2. For defining synchronization on same events[2], we assume two finite alphabets Σ_1 and Σ_2, whose elements are used to label the transitions of A_1, respectively A_2. An edge of an automaton over an input alphabet Σ is now a tuple $e = \langle l, a, g, r, l' \rangle$. The product $A_1 \| A_2$ is defined in the obvious way [2]. The locations of the product automaton are pairs of locations of its constituent automata. The invariant of a new location consists of the conjunction of the invariants of the component locations. Symbols that belong to both alphabets are used for synchronization and must be taken simultaneously by both automata. Figure 3 illustrates two timed automata together with the resulting

[2] We present here communication based on synchronized transitions. Communication based on shared variables can be handled similarly.

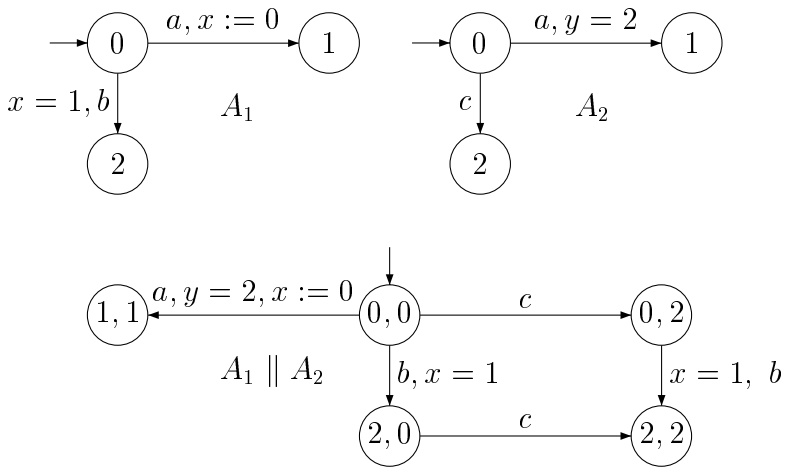

Fig. 3. Product construction for timed automata.

product automaton.

In order to encode the system $A_1 \| A_2$ into a $\mathcal{C}$-program, as described in Section 3 using Definition 3.1, the product automaton has to be constructed first. For networks consisting of a large number of components this leads to an exponential blow up in the number of resulting locations and transitions, and therefore also in the length of the Boolean constraint formulas. Here, we propose a method for encoding a network of timed automata into a $\mathcal{C}$-program in a compositional way, which does not require the construction of the product automaton.

For encoding the actions of a timed automaton we use a variable act that ranges over $\Sigma_1 \cup \ldots \cup \Sigma_n \cup \{delay\}$, where Σ_i $(i = 1, \ldots, n)$ are the alphabets corresponding to the n input automata. The special action $delay$ denotes the fact that a time elapse step is performed.

For a timed automaton A with alphabet σ and set of clocks Cl the formula $fix(A)$ is used to encode "inactivity", that is, the fact that A does not perform any transition.

$$fix(A) := (at' = at \ \wedge \bigwedge_{x \in Cl} x' = x \ \wedge \bigwedge_{\alpha \in \Sigma \cup delay} act \neq \alpha).$$

Every component is encoded in a similar way as illustrated in Definition 3.1, with the additional encoding of transition actions.

Definition 4.1 Consider a network of timed automata $A_1 \| \ldots \| A_n$, where $A_i = \langle L_i, l_i^0, \Sigma_i, Cl_i, E_i, Inv_i \rangle$, over the set of clocks $Cl_i = \{x_{i_1}, \ldots, x_{i_n}\}$, for $i = 1, \ldots, n$. The network is encoded over the set of state variables $V = V_1 \cup \ldots \cup V_n$, as the program

$$\langle I^s, T^s \rangle := \bigwedge_{i=1,\ldots,n} \langle I_i, T_i \rangle \ \wedge \ \delta \geq 0,$$

where δ is a state variable interpreted over $\mathbb{R}_0^+$, and $\langle I_i, T_i \rangle$ encodes the au-

tomaton A_i over the set $V_i = \{at_i, x_{i_1}, \ldots, x_{i_n}, act, \delta\}$ as follows:

- Definition of the initial state (as in Definition 3.1)

$$I_i := (at_i = l_{i_0} \ \wedge \ x_{i_1} = 0 \ \wedge \ \ldots \ \wedge \ x_{i_n} = 0).$$

- Definition of a state transition step corresponding to $e = \langle l, a, g, r, l' \rangle \in E_i$

$$\begin{aligned} T_i(e) := & (at_i = l \ \wedge \ act = a \ \wedge \ g \ \wedge \ x'_{i_1} = z_1 \ \wedge \ \ldots \ \wedge \ x'_{i_n} = z_n \ \wedge \\ & at'_i = l' \ \wedge \ Inv_i(l')(x_{i'_1}, \ldots, x_{i'_n})) \end{aligned}$$

  where $z_{i_j} = 0$ if $x_{i_j} \in r$; otherwise $z = x$. The state formula $Inv_i(l')(x_{i'_1}, \ldots, x_{i'_n})$ is obtained from the invariant of location l', $Inv_i(l')$, by replacing the variables $x_{i_1}, \ldots, x_{i_n}$ in the constraints of $Inv_i(l')$ by the primed variables $x_{i'_1}, \ldots, x_{i'_n}$.

- Definition of delay steps ($Inv(A_i)$ is the set of all locations that have an invariant different from *true*.)

$$\begin{aligned} D_i := & \bigwedge_{l \in Inv(A_i)} (at_i = l \Rightarrow Inv_i(l)(x_{i'_1}, \ldots, x_{i'_n})) \ \wedge \\ & act = delay \ \wedge \\ & x'_{i_1} = x_{i_1} + \delta \ \wedge \ \ldots \ \wedge \ x'_{i_n} = x_{i_n} + \delta \ \wedge \\ & at'_i = at_i. \end{aligned}$$

- Definition of the transition relation T

$$T_i := (\vee_{e \in E_i} T_i(e)) \ \vee \ fix(A_i) \ \vee \ D_i.$$

The network consisting of the timed automata A_1 and A_2 from Figure 3, for example, is defined as a program

$$\langle I^s, T^s \rangle = \langle I_1, T_1 \rangle \ \wedge \ \langle I_2, T_2 \rangle \ \wedge \ \delta \geq 0$$

over the set of variables

$$V = \{at_1, at_2, x, y, act, d\}, \text{ and } V' = \{at'_1, at'_2, x', y'\},$$

where $\langle I_1, T_1 \rangle$ encodes the timed automaton A_1, and $\langle I_2, T_2 \rangle$ encodes A_2.

$$\begin{aligned} I_1 = & (at_1 = 0 \ \wedge \ x = 0) \\ I_2 = & (at_2 = 0 \ \wedge \ y = 0) \\ T_1 = & (at_1 = 0 \ \wedge \ at'_1 = 1 \ \wedge \ x' = 0 \ \wedge \ act = a) \ \vee \\ & (at_1 = 0 \ \wedge \ at'_1 = 2 \ \wedge \ x = 1 \ \wedge \ x' = x \ \wedge \ act = b) \ \vee \\ & (at_1 = at'_1 \ \wedge \ x' = x \ \wedge \ act \neq a \ \wedge \ act \neq b \ \wedge \ act \neq delay) \ \vee \\ & (at_1 = at'_1 \ \wedge \ x' = x + \delta \ \wedge \ act = delay) \\ T_2 = & (at_2 = 0 \ \wedge \ at'_2 = 1 \ \wedge \ y = 2 \ \wedge y' = y \ \wedge \ act = a) \ \vee \\ & (at_2 = 0 \ \wedge \ at'_2 = 2 \ \wedge \ y' = y \ \wedge \ act = c) \ \vee \\ & (at_2 = at'_2 \ \wedge \ y' = y \ \wedge \ act \neq a \ \wedge \ act \neq c \ \wedge \ act \neq delay) \ \vee \\ & (at_2 = at'_2 \ \wedge \ act = delay \ \wedge \ y' = y + \delta) \end{aligned}$$

Theorem 4.2 (BMC for Networks of Timed Automata) Consider two timed automata with disjoint set of clocks, $A_i = \langle L_i, l_i^0, \Sigma_i, Cl_i, E_i, Inv_i \rangle$, for

$i = 1, 2$. Let $M^s = \langle I^s, T^s \rangle$ be the program corresponding to the network $A_1 \| A_2$ as given in Definition 4.1, and $M = \langle I, T \rangle$ be the program encoding the product automaton $A_1 \times A_2$ according to Definition 3.1. Then for a $k \in \mathbb{N}$, the kth unfolding of M^s and M are equisatisfiable, that is $[\![M^s]\!]_k \equiv [\![M]\!]_k$.
Proofsketch. By induction over k we show that $[\![M]\!]_k$ and $[\![M^s]\!]_k$ are equisatisfiable.

5 Discussion and Conclusion

We presented a bounded model checking procedure (BMC) for timed automata and linear temporal logic with real-valued clock constraints. The main contribution is a complete BMC algorithm for timed automata [3], which is compositional in that Boolean constraint formulas encoding complex systems can be obtained by Boolean combinations of the encoding of the components. A direct encoding of the product automaton would cause an exponential blow up in the length of the resulting Boolean constraint formula. Further, we give lower and upper bounds for the length k of counterexamples, that depend on the structure of the Büchi automaton of the given formula, and the region automaton corresponding to the timed automaton.

Recently and independently, bounded model checking for timed systems has also been studied by other researchers. Niebert, Mahfoudh, Asarin, Bozga, Jain, and Maler [25] give a translation for timed automata into formulas in Pratt's difference logic, and express bounded reachability problems for timed automata as formulas in this logic. Audemard, Cimatti, Kornilowicz, and Sebastiani [5] extend the techniques from [8] to timed systems, and illustrates that the performance time for bounded reachability for timed systems can considerably be improved using symmetry reduction. Penczek, Wozna, and Zbrzezny [27] also extend the techniques from [8] to timed automata and TACTL. The region graph corresponding to the timed automaton, together with a TACTL formula are encoded into a Boolean constraint formula, whose satisfiability is checked using an in-house developed tool. The presented technique is not compositional.

The main problem of the BMC approach is to come up with efficient algorithms for solving the satisfiability problem for Boolean constraint formulas. Specialized data structures for timed automata, such as difference bounded matrices (DBM) [14], clock difference diagrams (CDD) [21], or difference decision diagrams (DDD) [23], can not be applied directly for BMC, since the generated formulas contain clock constraints of the form $x' - x = y' - y$, as needed for encoding the delay steps. However, timing constraints that relate 4 clock variables can be reduced to equivalent timing constraints with 2 variables, expressible in Pratt's difference logic, by introducing a global variable T that measure the time since the system start, without beeing reset, as shown

[3] The completeness proof can be adapted to any systems with a finite bisimulation.

in [25]. For every clock x_i the variable $C_i = T - x_i$ represents the last time when x_i was reset. Now, guards and invariants are evaluated on $T - C_i$ instead of on x_i, time elapse affects only T, and a reset of x_i at time T corresponds to the assignment $C_i := T$.

On the other hand, general-purpose theorem proving, such as PVS [26], which uses a combination of BDDs [7] and linear arithmetic reasoning based on loop residue [28], does not work very efficient. For example, finding a counterexample of length $k = 2$ in the (modified) train gate controller protocol requires around 70 seconds, and for $k = 3$ around 8500 seconds. Recently, new techniques for checking satisfiability of Boolean constraint formulas, have been developed, by combining SAT solvers with domain-specific decision procedures based on lemmas on demand [13,6]. A prototypical satisfiability solver [13,12] has been implemented that combines an own SAT solver with the decision procedures ICS [15]. The core of the satisfiability solver is a refinement algorithm based on lazy theorem proving. In each refinement step, the Boolean satisfiability checker is used to suggest candidate assignments. Then ICS checks whether such a Boolean assignment determines a consistent assignment for the corresponding set of constraints. Whenever such a consistency check fails, the current Boolean formula is refined by adding a Boolean analogue of this inconsistency. The SAT solver is restarted, and a new candidate assignment for the refined formula is suggested.

We have performed some initial experiments, using the train gate controller and Fischer's mutual exclusion protocol [20], with a slight modification of the timing constraints. We encoded the system consisting of train, gate, and controller as a Boolean constraint formula in a compositional way, as described in Section 4, and checked the safety property that whenever the train is in the crossing the gate should be closed. On a Pentium III, 500 MHz, 1GB, we found a counter example of length 4 in 0.01 seconds. Using the correct version of the protocol, that is with timing constraints that guarantee the above safety property, we prove that there is no counterexample of length i, for $i \leq 100$. The timing performance for $k = 10, 20, 30, 40, 50, 60, 70$ is illustrated in Figure 4. For $k = 80$ the obtained time was greater then 4 hours, and for $k = 100$ greater then 5 hours. Note, that in the case of the correct version of the train gate controller we are performing bounded verification, and not only searching for counterexamples.

We also encoded Fischer's mutual exclusion protcol with of $n = 2, \ldots, 10$ processes as a Boolean constraint formula in a compositional way. On a Pentium III, 500 MHz, 1GB, for 5 processes a counterexample of length 9 was found in 25.87 seconds. For a system consisting of 10 processes a counterexample of length 8 was found in 62.42 seconds.

Although in an initial phase, the performed experiments show that BMC is a promising technique for verifying timed systems. Errors in larger systems for which conventional timed model checking tools fail or are inefficient, can be found using BMC.

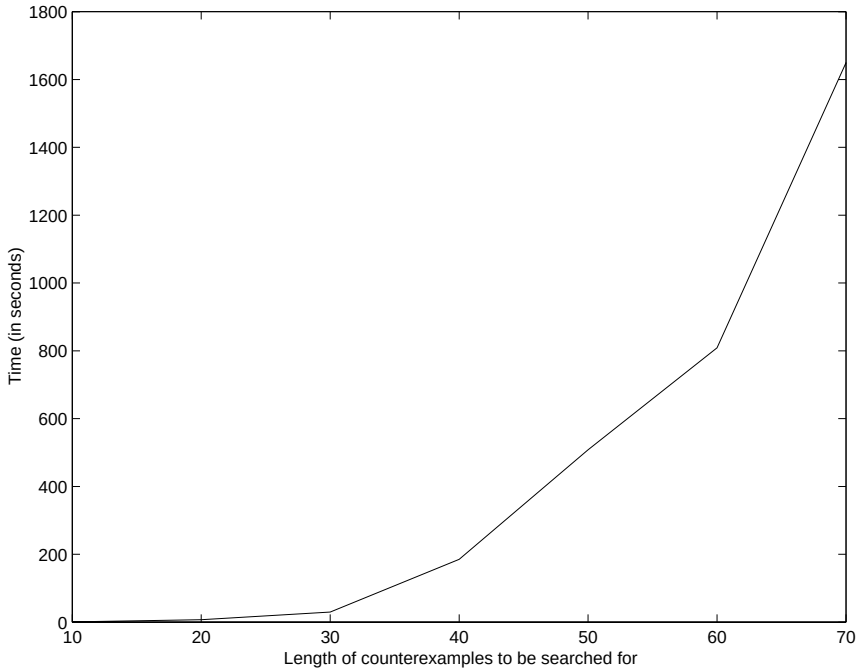

Fig. 4. Train gate controller – time for searching for counterexamples of length 5 to 100. For length 10 we obtain 0.46 seconds, for length 50, 508 seconds, and for length 70, 1655 seconds.

Acknowledgement

We would like to thank the anonymous referees for their helpful comments for improving this paper. Leonardo de Moura helped with the experiments and also provided many useful inputs.

References

[1] Alur, R., "Techniques for Automatic Verification of Real-Time Systems," Ph.D. thesis, Stanford University (1991).

[2] Alur, R., *Timed automata*, Lecture Notes in Computer Science **1633** (1999), pp. 8–22.

[3] Alur, R., C. Courcoubetis and D. Dill, *Model-checking for real-time systems*, 5th Symp. on Logic in Computer Science (LICS 90) (1990), pp. 414–425.

[4] Alur, R. and D. L. Dill, *A theory of timed automata*, Theoretical Computer Science **126** (1994), pp. 183–235.

[5] Audemard, G., A. Cimatti, A. Kornilowicz and R. Sebastiani, *Bounded model checking for timed systems*, Proceedings of the 2nd Workshop on Real-Time Tools (RT-TOOLS'2002) (2002).

[6] Barrett, C. W., D. L. Dill and A. Stump, *Checking satisfiability of first-order formulas by incremental translation to SAT* (2002), to be presented at CAV 2002.

[7] Bryant, R. E., *Graph-based algorithms for boolean function Manipulation*, IEEE Transactions on Computers **C-35** (1986), pp. 677–691.

[8] Clarke, E. M., A. Biere, R. Raimi and Y. Zhu, *Bounded model checking using satisfiability solving*, Formal Methods in System Design **19** (2001), pp. 7–34.

[9] Copty, F., L. Fix, R. Fraer, E. Giunchiglia, G. Kamhi, A. Tacchella and M. Vardi, *Benefits of bounded model checking in an industrial setting*, in: *Computer-Aided Verification, CAV 2001*, Lecture Notes in Computer Science **2101** (2001), pp. 436–453.

[10] Dams, D. R., "Abstract Interpretation and Partition Refinement for Model Checking," Ph.D. thesis, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands (1996).

[11] Daws, C., A. Olivero, S. Tripakis and S. Yovine, *The tool KRONOS*, Lecture Notes in Computer Science **1066** (1996), pp. 208–219.

[12] de Moura, L. and H. Rueß, *Lemmas on demand for satisfiability solvers*, in: *Proceedings of the Fifth International Symposium on the Theory and Applications of Satisfiability Testing (SAT 2002)*, Cincinnati, Ohio, 2002.

[13] de Moura, L., H. Rueß and M. Sorea, *Lazy theorem proving for bounded model checking over infinite domains*, in: *18th Conference on Automated Deduction (CADE)*, Lecture Notes in Computer Science (2002).
URL `http://www.csl.sri.com/users/sorea/papers/CADE02/index.html`

[14] Dill, D., *Timing assumptions and verification of finite-state concurrent systems*, in: *Proceedings of the International Workshop on Automatic Verification Methods for Finite State Systems*, Lecture Notes in Computer Science **407** (1989), pp. 197–212.

[15] Filliâtre, J.-C., S. Owre, H. Rueß and N. Shankar, *ICS: Integrated canonizer and solver*, in: G. Berry, H. Comon and A. Finkel, editors, *Proceedings of CAV'2001*, Lecture Notes in Computer Science **2102** (2001), pp. 246–249.

[16] Gerth, R., D. Peled, M. Vardi and P. Wolper, *Simple on-the-fly automatic verification of linear temporal logic*, in: *Protocol Specification Testing and Verification* (1995), pp. 3–18.

[17] Henzinger, T. A., P.-H. Ho and H. Wong-Toi, *HYTECH: A model checker for hybrid systems*, Lecture Notes in Computer Science **1254** (1997), pp. 460–463.

[18] Henzinger, T. A., X. Nicollin, J. Sifakis and S. Yovine, *Symbolic model checking for real-time systems*, Information and Computation **111** (1994), pp. 193–244.

[19] Kupferman, O. and M. Y. Vardi, *Model checking of safety properties*, Formal Methods in System Design **19** (2001), pp. 291–314.

[20] Lamport, L., *A fast mutual exclusion algorithm*, ACM Transactions on Computer Systems **5** (1987), pp. 1–11.

[21] Larsen, K. G., J. Pearson, C. Weise and W. Yi, *Clock difference diagrams*, Nordic Journal of Computing **6** (1999), pp. 271–298.

[22] Larsen, K. G., P. Pettersson and W. Yi, Uppaal *in a nutshell*, Int. Journal on Software Tools for Technology Transfer **1** (1997), pp. 134–152.

[23] Møller, J., J. Lichtenberg, H. R. Andersen and H. Hulgaard, *Difference decision diagrams*, in: *Computer Science Logic*, The IT University of Copenhagen, Denmark, 1999.

[24] Möller, M. O., H. Rueß and M. Sorea, *Predicate abstraction for dense real-time systems*, in: E. Asarin, O. Maler and S. Yovine, editors, *Theory and Practice of Timed Systems (TPTS'02)*, Electronic Notes in Theoretical Computer Science **65**, 2002.
URL `http://www.elsevier.com/locate/entcs/volume65.html`

[25] Niebert, P., M. Mahfoudh, E. Asarin, M. Bozga, N. Jain and O. Maler, *Verification of timed automata via satisfiability checking*, in: *Proceedings of the 7th International Symposium on Formal Techniques in Real-Time and Fault Tolerant Systems (FTRTFT)*, Lecture Notes in Computer Science (2002).

[26] Owre, S., J. M. Rushby and N. Shankar, *PVS: A prototype verification system*, in: *11th International Conference on Automated Deduction (CADE)*, Lecture Notes in Artificial Intelligence **607** (1992), pp. 748–752.

[27] Penczek, W., B. Wozna and A. Zbrzezny, *Towards bounded model checking for the universal fragment of TCTL*, in: *Proceedings of the 7th International Symposium on Formal Techniques in Real-Time and Fault Tolerant Systems (FTRTFT)*, Lecture Notes in Computer Science (2002).

[28] Shostak, R., *Deciding linear inequalities by computing loop residues*, Journal of the ACM **28** (1981), pp. 769–779.

[29] Sistla, A. P., *Safety, liveness and fairness in temporal logic*, Formal Aspects of Computing **6** (1994), pp. 495–512.

[30] Sorea, M., *Tempo: A model-checker for event-recording automata*, in: *Proceedings of RT-TOOLS'01*, Aalborg, Denmark, 2001, also available as Technical Report SRI-CSL-01-04, Computer Science Laboratory, SRI International, Menlo Park, CA, 2001.
URL `http://www.csl.sri.com/papers/csl-01-04/`

[31] Tripakis, S. and S. Yovine, *Analysis of timed systems using time-abstracting bisimulations*, Formal Methods in System Design **18** (2001), pp. 25–68, Kluwer Academic Publishers.

Recent BRICS Notes Series Publications

NS-02-3 **Walter Vogler and Kim G. Larsen, editors.** ***Preliminary Proceedings of the 3rd International Workshop on Models for Time-Critical Systems, MTCS '02,*** **(Brno, Czech Republic, August 24, 2002), August 2002. vi+141 pp.**

NS-02-2 **Zoltán Ésik and Anna Ingólfsdóttir, editors.** ***Preliminary Proceedings of the Workshop on Fixed Points in Computer Science, FICS '02,*** **(Copenhagen, Denmark, July 20 and 21, 2002), June 2002. iv+81 pp.**

NS-02-1 **Anders Møller and Michael I. Schwartzbach.** ***Interactive Web Services with Java: JSP, Servlets, and JWIG.*** **April 2002. 99 pp.**

NS-01-8 **Anders Møller and Michael I. Schwartzbach.** ***The XML Revolution (Revised).*** **December 2001. 186 pp. This revised and extended report superseeds the earlier BRICS Report NS-00-8.**

NS-01-7 **Patrick Cousot, Lisbeth Fajstrup, Eric Goubault, Jeremy Gunawardena, Maurice Herlihy, Martin Raußen, and Vladimiro Sassone, editors.** ***Preliminary Proceedings of the Workshop on Geometry and Topology in Concurrency Theory, GETCO '01,*** **(Aalborg, Denmark, August 25, 2001), August 2001. vi+97 pp.**

NS-01-6 **Luca Aceto and Prakash Panangaden, editors.** ***Preliminary Proceedings of the 8th International Workshop on Expressiveness in Concurrency, EXPRESS '01,*** **(Aalborg, Denmark, August 20, 2001), August 2001. vi+139 pp.**

NS-01-5 **Flavio Corradini and Walter Vogler, editors.** ***Preliminary Proceedings of the 2nd International Workshop on Models for Time-Critical Systems, MTCS '01,*** **(Aalborg, Denmark, August 25, 2001), August 2001. vi+ 127pp.**

NS-01-4 **Ed Brinksma and Jan Tretmans, editors.** ***Proceedings of the Workshop on Formal Approaches to Testing of Software, FATES '01,*** **(Aalborg, Denmark, August 25, 2001), August 2001. viii+156 pp.**

NS-01-3 **Martin Hofmann, editor.** ***Proceedings of the 3rd International Workshop on Implicit Computational Complexity, ICC '01,*** **(Aarhus, Denmark, May 20–21, 2001), May 2001. vi+144 pp.**