Patricia Bouyer, Serge Haddad and
Pierre-Alain Reynier

Extended Timed Automata and Time Petri Nets

Research Report LSV-06-01

January 2006

Laboratoire
Spécification
et
Vérification

CENTRE NATIONAL
DE LA RECHERCHE
SCIENTIFIQUE

Ecole Normale Supérieure de Cachan
61, avenue du Président Wilson
94235 Cachan Cedex France

Extended Timed Automata and Time Petri Nets

Patricia Bouyer
LSV, CNRS & ENS de Cachan, France
e-mail: `bouyer@lsv.ens-cachan.fr`

Serge Haddad
LAMSADE, CNRS & Université Paris-Dauphine, France
e-mail: `haddad@lamsade.dauphine.fr`

Pierre-Alain Reynier
LSV, CNRS & ENS de Cachan, France
e-mail: `reynier@lsv.ens-cachan.fr`

Abstract

Timed Automata (TA) and Time Petri Nets (TPN) are two well-established formal models for real-time systems. Recently, a linear transformation of TA to TPNs preserving reachability properties and timed languages has been proposed, which does however not extend to larger classes of TA which would allow diagonal constraints or more general resets of clocks. Though these features do not add expressiveness, they yield exponentially more concise models.

In this work, we propose two translations: one from extended TA to TPNs whose size is either linear or quadratic in the size of the original TA, depending on the features which are allowed; another one from a parallel composition of TA to TPNs, which is also linear. As a consequence, we get that TPNs are exponentially more concise than TA.

Keywords*: Time Petri Nets, Timed Automata, Conciseness, Reachability Analysis.*

1 Introduction

Extended timed automata. Timed automata have been defined in the nineties as a powerful model for representing real-time systems [1, 2]. One of the most important properties of this model is that checking reachability properties (or equivalently language emptiness) is decidable. In the original model, a transition is guarded by a clock constraint $x \bowtie h$ (where x is a variable called clock, h is an integer and $\bowtie$ is a comparison operator), and resets to 0 a subset of the clocks.

Several extensions of this original model have been since considered: more general constraints like diagonal constraints [2, 7] or additive constraints [8] have been studied, and while additive constraints lead to undecidability, diagonal constraints preserve the decidability of the model. Also, more general operations on clocks (called updates) have been considered, and models using operations like resetting a clock to some integral value have been studied [13]: decidability of these extensions heavily depends on the nature of the updates and of the clock constraints which are used. All mentioned decidable extensions, do not add expressive power to the original model, they can thus be seen as syntactic sugar, but even though no expressiveness is added, these extensions yield exponentially more concise and "easy-to-design" models [12]. For example, scheduling problems are modeled more easily using these both extensions, see [16].

Time Petri nets. Adding explicit time to Petri nets was first done in the seventies [22, 23]. Since then, timed models based on Petri nets have been extensively studied and various tools have been developed for their analysis (like Tina [10] or Romeo [17]). In this paper, we focus on the model of Time Petri Nets (TPNs) from [22] where a time interval associated with every transition restricts the date at which it can be fired. Furthermore, time cannot elapse if it *temporally* disables a transition.

From TA to TPNs. In [18], the authors compare diagonal-free TAs without strict constraints and Safe-TPNs (*i.e.* TPNs where the number of tokens in a place is at most 1), and give a translation from TAs to Safe-TPNs which preserves timed languages. The complexity of the translation is quadratic. In [6], another translation is designed, which transforms diagonal-free TAs to equivalent Safe-TPNs, and whose complexity is linear. However the transformation which is done in [6] does not extend to more general TA (using diagonal constraints and resets to integral values).

Our contribution. In this work, we present a translation from extended TA (which use diagonal constraints and updates to integral values) into Safe-TPNs and prove it preserves timed languages. We study the complexity of this translation showing that the size of the constructed TPN is linear w.r.t. the size of the TA, except when the TA includes both diagonal constraints and arbitrary resets to integral values. In the latter case, the complexity of the translation becomes quadratic. As a side result (applying conciseness results of [12]), we get that Safe-TPNs are exponentially more concise than classical TA. We also provide a direct construction which witnesses this conciseness result. Finally, we present a second translation from a parallel composition of TA to TPNs. As future work, we discuss another consequence of these translations: they might provide a unified method to check for reachability properties in parallel composition of extended TA.

Organisation of the paper. In section 2, we present basic definitions. Our results are developed in section 3. We first present the construction of TPNs equivalent to extended TA, and we then prove the correctness of this translation. Finally, we compute the complexity of our translation, and present our conciseness results. In section 4, we present our construction for parallel compositions of TA. At last, in section 5, we discuss the perspectives of this work.

2 Timed Automata and Time Petri Nets

Let Σ be a finite alphabet, Σ^* is the set of finite words of Σ. We also use $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$ with ε (the empty word) not in Σ. The sets $\mathbb{N}$, $\mathbb{Q}_{\geq 0}$ and $\mathbb{R}_{\geq 0}$ are respectively the sets of natural, non-negative rational and non-negative real numbers.

A *timed word* over Σ is a finite sequence $w = (a_0, t_0)(a_1, t_1)\dots(a_n, t_n)$ s.t. for every $0 \leq i \leq n$, $a_i \in \Sigma$, $t_i \in \mathbb{R}_{\geq 0}$ and $t_{i+1} \geq t_i$. In the following, we will equivalently write $w = (a, t)$ with $a = (a_i)_{0\leq i\leq n}$ and $t = (t_i)_{0\leq i\leq n}$.

An interval I of $\mathbb{R}_{\geq 0}$ is a $\mathbb{Q}_{\geq 0}$-*interval* iff its left endpoint belongs to $\mathbb{Q}_{\geq 0}$ and its right endpoint belongs to $\mathbb{Q}_{\geq 0} \cup \{\infty\}$. We set $I^{\downarrow} = \{x \mid x \leq y \text{ for some } y \in I\}$, the *downward closure* of I. We denote by $\mathcal{I}(\mathbb{Q}_{\geq 0})$ the set of $\mathbb{Q}_{\geq 0}$-intervals of $\mathbb{R}_{\geq 0}$.

A *valuation* v over a finite set X is a mapping in $\mathbb{R}_{\geq 0}^X$. We note $\mathbf{0}$ the valuation which assigns to every clock $x \in X$ the value 0.

2.1 Timed Transition Systems

Timed transition systems describe systems which combine discrete and continuous evolutions. They are used to define and compare the semantics of TPNs and TA.

Definition 1 (Timed Transition System (TTS))**.** *A* timed transition system *is a transition system $S = (Q, q_0, \rightarrow)$, where Q is the set of states, $q_0 \in Q$ is the initial state, and the transition relation $\rightarrow$ consists of delay moves $q \xrightarrow{d} q'$ (with $d \in \mathbb{R}_{\geq 0}$), and discrete moves $q \xrightarrow{a} q'$ (with $a \in \Sigma_\varepsilon$). Moreover, we require standard properties for the transition relation $\rightarrow$:*

Time-determinism: *if $q \xrightarrow{d} q'$ and $q \xrightarrow{d} q''$ with $d \in \mathbb{R}_{\geq 0}$, then $q' = q''$;*

0**-delay:** *$q \xrightarrow{0} q$;*

Additivity: *if $q \xrightarrow{d} q'$ and $q' \xrightarrow{d'} q''$ with $d, d' \in \mathbb{R}_{\geq 0}$, then $q \xrightarrow{d+d'} q''$;*

Continuity: *if $q \xrightarrow{d} q'$, then for every d' and d'' in $\mathbb{R}_{\geq 0}$ such that $d = d' + d''$, there exists q'' such that $q \xrightarrow{d'} q'' \xrightarrow{d''} q'$.*

With these properties, a *run* of S can be defined as a finite sequence of moves $\rho = q_0 \xrightarrow{d_0} q_0' \xrightarrow{a_0} q_1 \xrightarrow{d_1} q_1' \xrightarrow{a_1} q_2 \dots \xrightarrow{a_n} q_{n+1}$ where discrete actions and delays alternate. To such a run corresponds a timed word $w = (a_i, t_i)_{0\leq i\leq n}$ over Σ_ε where $t_i = \sum_{j=0}^{i} d_j$ is the date at which a_i happens. Finally, by projection of w over Σ, we get the timed word *Timed*(ρ) which is the timed word accepted by run ρ.

Given a set $F \subseteq Q$ of final states, we say that a run ρ of S is *accepting* if it ends up in a state of F. The timed word *Timed*(ρ) is then said accepted by S.

2.2 Timed Automata

Syntax. First defined in [2], the model of timed automata associates a set of non-negative real-valued variables called *clocks* with a finite automaton. Let X be a finite set of clocks. We write $\mathcal{C}(X)$ for the set of *constraints* over X, which consist of conjunctions of atomic formulae of the form $x \bowtie c$ and $x - y \bowtie c$ for $x, y \in X$, $c \in \mathbb{Z}$ and $\bowtie \in \{<, \leq, \geq, >\}$. We also define the proper subset $\mathcal{C}_{df}(X)$ of *diagonal-free* constraints over X where the constraints of the form $x - y \bowtie h$ (called *diagonal constraints*) are not allowed. Finally, the set $\mathcal{R}(X)$ of arbitrary resets to integral values over the clocks X is defined as the set $(\mathbb{N} \cup \{\perp\})^X$ of mappings from X to $\mathbb{N} \cup \{\perp\}$. The framework of classical resets to zero is obtained by considering the proper subset $\mathcal{R}_0(X) = \{0, \perp\}^X$. For example, the reset $x := 2$ is encoded as a function mapping clock x to the value 2 and other clocks to $\perp$. In the following, we write a general reset as a conjunction $(x_1 := c_1 \wedge \dots \wedge x_k := c_k)$.

Definition 2 (Timed Automaton (TA))**.** *A* timed automaton *$\mathcal{A}$ over Σ_ε is a tuple $(L, \ell_0, X, \Sigma_\varepsilon, E)$ where L is a finite set of* locations*, $\ell_0 \in L$ is the* initial location*, X is a finite set of* clocks *and $E \subseteq L \times \mathcal{C}(X) \times \Sigma_\varepsilon \times \mathcal{R}(X) \times L$ is a finite set of* edges*. An edge $e = (\ell, \gamma, a, R, \ell') \in E$ represents* *a transition from location ℓ to location ℓ' labeled by a with constraint γ and reset $R \in (\mathbb{N} \cup \{\bot\})^X$. We say that the timed automaton $\mathcal{A}$ is* diagonal-free *(resp.* 0-reset*) if the set $\mathcal{C}(X)$ (resp. $\mathcal{R}(X)$) is replaced by its subset $\mathcal{C}_{df}(X)$ (resp. $\mathcal{R}_0(X)$).*

Semantics. For $R \in (\mathbb{N} \cup \{\bot\})^X$, the valuation $R(v)$ is the valuation v' such that $v'(x) = v(x)$ when $R(x) = \bot$ and $v'(x) = R(x)$ otherwise. For any value $d \in \mathbb{R}_{\geq 0}$, the valuation $v+d$ is defined by $(v+d)(x) = v(x)+d$, $\forall x \in X$. Finally, constraints of $\mathcal{C}(X)$ are interpreted over valuations: we write $v \models \gamma$ when the constraint γ is satisfied by v.

Definition 3 (Semantics of TA)**.** *The semantics of a TA $\mathcal{A} = (L, \ell_0, X, \Sigma_\varepsilon, E)$ is the TTS $S_{\mathcal{A}} = (Q, q_0, \rightarrow)$ where $Q = L \times (\mathbb{R}_{\geq 0})^X$, $q_0 = (\ell_0, \mathbf{0})$ and $\rightarrow$ is defined by:*

- ***delay moves:*** *$(\ell, v) \xrightarrow{d} (\ell, v + d)$ if $d \in \mathbb{R}_{\geq 0}$;*
- ***discrete moves:*** *$(\ell, v) \xrightarrow{a} (\ell', v')$ if there exists some $e = (\ell, \gamma, a, R, \ell') \in E$ s.t. $v \models \gamma$ and $v' = R(v)$.*

If F is a set of final locations for $\mathcal{A}$, the timed language accepted by $\mathcal{A}$, denoted $\mathcal{L}(\mathcal{A})$ is the set of timed words accepted by $S_{\mathcal{A}}$ for the final set of states $F \times \mathbb{R}_{\geq 0}$.

Parallel composition of TA. Let $(\mathcal{A}_i)_{1 \leq i \leq n}$ be a family of n TA. We assume that $\mathcal{A}_i = (L_i, \ell_{i,0}, X_i, \Sigma_\varepsilon, E_i)$ for every $1 \leq i \leq n$ and that X_i's are disjoint sets of clocks. Let $f : (\Sigma \cup \{\bot\})^n \rightarrow \Sigma_\varepsilon$ be an n-ary partial synchronization function. The parallel composition of $(\mathcal{A}_i)_{1 \leq i \leq n}$ w.r.t. f is the TA $\mathcal{A} = (L, \ell_0, X, \Sigma_\varepsilon, E)$ such that $L = L_1 \times \ldots \times L_n$, $\ell_0 = (\ell_{1,0}, \ldots, \ell_{n,0})$, $X = X_1 \cup \ldots \cup X_n$, $(\ell_1, \ldots, \ell_n) \xrightarrow{g,a,R} (\ell'_1, \ldots, \ell'_n)$ whenever (i) either $a = \varepsilon$, there exists $1 \leq i \leq n$ such that $\ell_i \xrightarrow{g,\varepsilon,R} \ell'_i$ in E_i, and $\ell_j = \ell'_j$ if $i \neq j$; (ii) or there exists $I \subseteq \{1, \ldots, n\}$, there exist $\ell_i \xrightarrow{g_i,a_i,R_i} \ell'_i$ in E_i (for $i \in I$) such that:

- $g = \bigwedge_{i \in I} g_i$,
- $R(x) = \begin{cases} R_i(x) \text{ if } x \in X_i \text{ and } i \in I \\ \bot \text{ otherwise} \end{cases}$
- $\ell'_i = \ell_i$ if $i \notin I$.
- either $f(a_1, \ldots, a_n) = a$ where $a_i = \bot$ if $i \notin I$.

2.3 Time Petri Nets

Syntax. Introduced in [22], Time Petri nets (TPNs) associate a time interval to each transition of a Petri net.

Definition 4 (Labeled TPN)**.** *A labeled time Petri net $\mathcal{N}$ over Σ_ε is a tuple $(P, T, \Sigma_\varepsilon, {}^\bullet(.), (.)^\bullet, M_0, \Lambda, I)$ where:*

- *P is a finite set of* places*,*
- *T is a finite set of* transitions *with $P \cap T = \emptyset$,*
- *${}^\bullet(.) \in (\mathbb{N}^P)^T$ is the* backward *incidence mapping,*
- *$(.)^\bullet \in (\mathbb{N}^P)^T$ is the* forward *incidence mapping,*
- *$M_0 \in \mathbb{N}^P$ is the* initial *marking,*
- *$\Lambda : T \rightarrow \Sigma_\varepsilon$ is the* labeling function
- *$I : T \mapsto \mathcal{I}(\mathbb{N})$ associates with each transition a* firing interval*.*

As commonly in use in the literature, we write ${}^\bullet t$ (resp. $t^\bullet$) to denote the set of places ${}^\bullet t = \{p \in P \mid {}^\bullet t(p) > 0\}$ (resp. $t^\bullet = \{p \in P \mid t^\bullet(p) > 0\}$).

Semantics. A *configuration* of a TPN is a pair (M, ν), where M is a *marking* in the usual sense, *i.e.* a mapping in $\mathbb{N}^P$, with $M(p)$ the number of tokens in place p. A transition t is *enabled* in a marking M if $M \geq {}^\bullet t$. We denote by $\mathit{En}(M)$ the set of enabled transitions in M. The second component of the pair (M, ν) is a valuation over $\mathit{En}(M)$ which associates to each enabled transition its age, *i.e.* the amount of time that has elapsed since this transition is enabled. An enabled transition t can be fired if $\nu(t)$ belongs to the interval $I(t)$. The result of this firing is as usual the new marking $M' = M - {}^\bullet t + t^\bullet$. Moreover, some valuations are reset and we say that the corresponding transitions are newly enabled. Different semantics are possible for this operation. In this paper, we choose the classical semantics [9, 3] (see [5] for alternative semantics). The predicate specifying when t' is newly enabled by the firing of t from marking M is defined by:

$$\uparrow\!\mathit{enabled}(t', M, t) = t' \in \mathit{En}(M - {}^\bullet t + t^\bullet) \\ \wedge ((t' \notin \mathit{En}(M - {}^\bullet t)) \vee t = t')$$

Thus, firing a transition is not considered as an atomic step and the transition currently fired is always reset.

The set $\mathit{ADM}(\mathcal{N})$ of *(admissible) configurations* consists of the pairs (M, ν) such that $\nu(t) \in I(t)^{\downarrow}$ for every transition $t \in \mathit{En}(M)$. Thus time can progress in a marking only when it does not leave the firing interval of any enabled transition.

Definition 5 (Semantics of a TPN)**.** *The semantics of a TPN $\mathcal{N} = (P, T, \Sigma_\varepsilon, {}^\bullet(.), (.)^\bullet, M_0, \Lambda, I)$ is a TTS $S_{\mathcal{N}} = (Q, q_0, \rightarrow)$ where $Q = \mathit{ADM}(\mathcal{N})$, $q_0 = (M_0, \mathbf{0})$ and $\rightarrow$ is defined by:*

- ***delay moves:*** *$(M, \nu) \xrightarrow{\tau} (M, \nu + \tau)$ iff $\forall t \in \mathit{En}(M)$, $\nu(t) + \tau \in I(t)^{\downarrow}$,*
- ***discrete moves:*** *$(M, \nu) \xrightarrow{\Lambda(t)} (M - {}^\bullet t + t^\bullet, \nu')$ iff $t \in \mathit{En}(()M)$ is s.t. $\nu(t) \in I(t)$, and $\forall t' \in \mathit{En}(M - {}^\bullet t + t^\bullet)$,*
  - *$\nu'(t') = 0$ if $\uparrow\!\mathit{enabled}(t', M, t)$*
  - *and $\nu'(t') = \nu(t)$ otherwise.*

If F is a set of final places of $\mathcal{N}$, we note $\mathcal{L}(\mathcal{N})$ the timed language accepted by $\mathcal{N}$, *i.e.* the set of timed words accepted by $S_{\mathcal{N}}$ for the final set of states (M, ν) s.t. $M(f) \neq 0$ for some $f \in F$

A *Safe-TPN* is a TPN $\mathcal{N}$ where all configurations reachable in $S_{\mathcal{N}}$ contain at most one token in every place.

3 From Extended TA to TPNs

In this section, we describe the construction of a TPN "equivalent" to a TA (w.r.t. their timed languages), that is accepting the same timed languages. The correctness is proved in the next section.

We assume we are given a timed automaton $\mathcal{A}$. We will construct an equivalent TPN in a modular way. Note that this TPN will be safe by construction. Places with the same name are shared by several subnets. Omitted labels for transitions stand for ε. A firing interval $[0, 0]$ is depicted by a blackened transition and is called an immediate transition, and intervals $[0, \infty[$ are omitted. A double arrow between a place p and a transition t indicates that p is both an input and an output place for t.

3.1 The construction

The clock evolution subnet. For each clock x of the TA, we construct a subnet which records and tracks the value of x. More precisely, this subnet records both the value of the clock (though in an implicit way) and the truth of all the constraints $x \bowtie c$ appearing in the automaton. The truth value of such a constraint is recorded explicitly, using a place $T_{x \bowtie c}$. For all clock resets $y := h$ and diagonal constraints $x - y \bowtie c$ appearing in the automaton, the subnet has also to take into account the constraint $x \bowtie c + h$ and its negation (except if it is trivially equivalent to *true* or *false*). It must also take into account the constraints $x \leq c$ and $x \geq c$ when $x := c$ is a reset used in the automaton.

The subnet represented in Fig. 1 illustrates our translation in the case x is compared with three constants $\{c_1, c_2, c_3\}$ with $c_1 < c_2 < c_3$. To ease the reading, we assume that 0 does not belong to the set of constants, though this case can be handled similarly.

Let us explain how this subnet simulates time elapsing, how it records the value of the clock, and how it records the truth value of the constraints. First notice that all places along the vertical axis (places $Before^x_{c_1}$, $At^x_{c_1}$, $\ldots$, $After^x_{c_3}$) are mutually exclusive. The unique token labelling one of these places together with the age[1] of the next transition encodes the value of the clock. For instance, if a token is in the place $Before^x_{c_2}$, and if the age of $Reach^x_{c_2}$ is τ then the value of x is $c_1 + \tau$. The value of x will be c_2 in the following cases:

- either the token is in the place $Before^x_{c_2}$, and the age of $Reach^x_{c_2}$ is $c_2 - c_1$,
- either the token is in the place $At^x_{c_2}$,
- or the token is in the place $Before^x_{c_3}$ and the age of $Reach^x_{c_3}$ is 0.

[1]Recall that the age of a transition is the amount of time which has elapsed since the transition has been enabled.

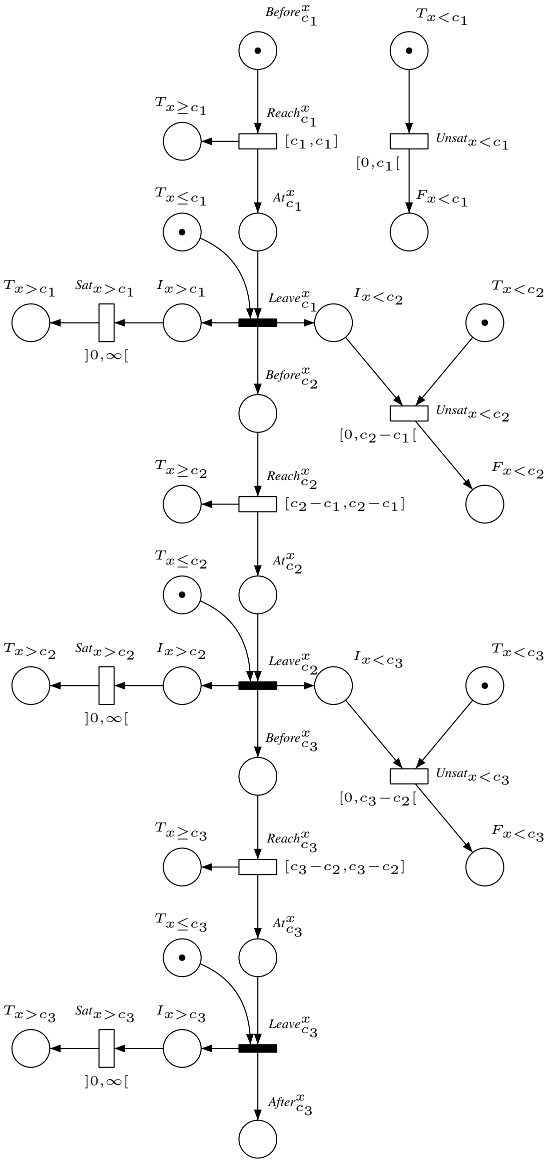

Figure 1. The clock evolution subnet (clock x)

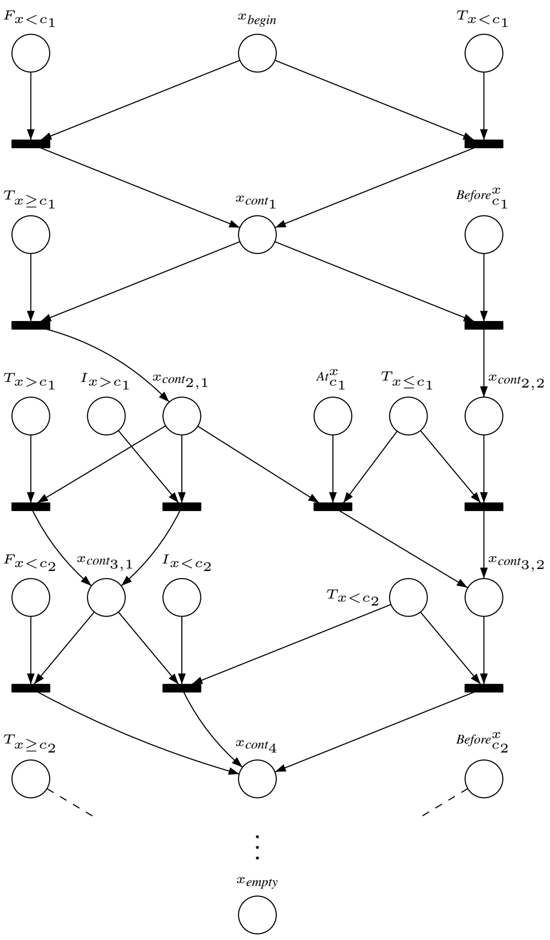

Figure 2. Emptying the clock evol. subnet

Finally, the subnet does not keep track of the exact value of the clock beyond c_3. The truth values of the constraints are updated consistently, while preserving the two following properties, which are fundamental for the correctness of our construction: 1) When the place $T_{x \bowtie c}$ is marked, then the corresponding value of x (say v_x) is such that $v_x \bowtie c$ (but the converse is not necessarily true); 2) For each possible value v_x of x, there is an execution of the subnet of time length v_x such that for every constraint $x \bowtie c$ satisfied by v, the place $T_{x \bowtie c}$ is marked. Finally, note that this subnet does not take care of diagonal constraints because they cannot be handled similarly (their truth values are unchanged when time elapses).

It is worth to notice that the size of this subnet is linear in the number of clock constraints involving x which need to be encoded (see the beginning of this paragraph).

Emptying the clock subnet. Let us assume that a transition of the TA resets the clock x. The marking of the clock evolution subnet must be updated accordingly, whatever its current configuration is.

In order to encode a transition of the TA and to control the global size of the resulting TPN, we proceed in two steps: 1) The first step is depicted in Fig. 2 and consists in consuming all the tokens which are in the clock evolution subnet; 2) The second step is discussed in the next paragraphs, and consists in marking the appropriate places of the clock evolution subnet.

In order to unmark all places of the subnet depicted in Fig 1, we will empty the places in a top-down way. The control places of the subnet of Fig. 2 (namely $\{x_{begin}, x_{cont_1}, x_{cont_{2,1}}, x_{cont_{2,2}}, \ldots, x_{empty}\}$) schedule the unmarking process and memorize some information in order to avoid a quadratic increase of the number of transitions.

Let us partly describe the subnet of Fig. 2. First, it removes the token which is either in place $F_{x<c_1}$ or in place $T_{x<c_1}$ (these two places are mutually exclusive, see transition $Unsat_{x<c_1}$). Then, it removes the token which is either in place $Before^x_{c_1}$, or in place $T_{x\geq c_1}$ (these two places are also mutually exclusive, see transition $Reach^x_{c_1}$). Thanks to the control places of the net in Fig. 2 (place $x_{cont_{2,1}}$ *vs* place $x_{cont_{2,2}}$), we remember whether the token was in place $Before^x_{c_1}$ or in place $T_{x\geq c_1}$. If the token was in place $Before^x_{c_1}$, there will be no token in places $T_{x>c_1}$ and $I_{x>c_1}$. On the contrary, if the token was in place $T_{x\geq c_1}$, then there will be no token in places $At^x_{c_1}$ and $T_{x\leq c_1}$. These remarks allow to bound the width of the "emptying net", and it allows to control the size of the net (on one level, there are at most 4 "concurrent" transitions). Finally note that the subnet is triggered by a token in place x_{begin} and that the clock evolution subnet is empty when a token arrives in place x_{empty}.

It is worth to notice that the size of the TPN emptying the clock evolution subnet is linear in the size of this last subnet.

Updating the marking of the constraints in the clock evolution subnet. We want to update the marking of the places coding the truth values of the constraints in the clock evolution subnet when a clock is reset to some integral value c. However, we want to control the size of the resulting TPN, we thus want to build only one subnet per clock which will update correctly the marking of the evolution subnet, though the new marking will depend on the value of c. The idea of our construction is the following: when the clock x is reset to c, then the constraint $x \leq c$ holds, and consequently, all other larger over-approximations of x ($x \prec c'$, for $\prec \in \{<, \leq\}$ and $c' > c$) also hold. Thus, we will build a propagation chain for the over-approximations which will respect the above implications. Of course, we can reason similarly for the under-approximations.

The two propagation subnets are represented in Fig. 3, and take advantage of the above observations. The two causal chains are represented by two different connected

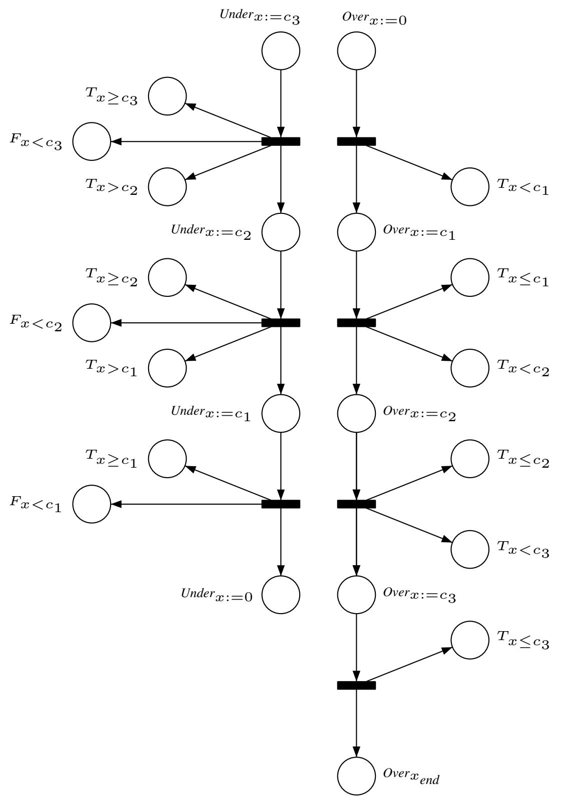

Figure 3. Marking the constraints places in the clock evolution subnet

components. In order to trigger this net when resetting $x := c_i$, one puts a token in place $\mathit{Under}_{x:=c_i}$ and $\mathit{Over}_{x:=c_i}$. For the subnet on the left (resp. on the right), the update of the marking terminates when a token arrives in place $\mathit{Under}_{x:=0}$ (resp. in place $\mathit{Over}_{x_{end}}$).

Note that we have not marked yet the vertical axis of the clock evolution subnet, which implicitly encodes the value of the clock. This will be done by the subnet simulating the transition of the TA (see the last paragraph of this section).

It is worth to notice that the size of the TPN marking the constraints places of the clock evolution subnet is linear in the size of the clock evolution subnet.

Diagonal clock constraints. The truth value of a diagonal constraint $x-y \bowtie h$ (which is invariant by time elapsing) is represented by two mutually exclusive places $T_{x-y\bowtie h}$ and $F_{x-y\bowtie h}$. We build a subnet for every atomic constraint $x - y \bowtie h$ and every reset of the clocks x or y.

Fig. 4 represents the subnet corresponding to the diagonal constraint $x - y \leq h$ and to the reset $y := h'$. When resetting y to h', the truth value of $x - y \leq h$ has to be updated according to the truth value of the (non-diagonal) constraint $x \leq h + h'$. The places $\{\mathit{Diag}_i^{y:=h'}\}_{i=1..d(y)+1}$ schedule the update of the subnets associated with the diagonal constraints involving clock y (and $d(y)$ is the number of such constraints). In Fig. 4, i is the index of the constraint $x - y \leq h$ ($1 \leq i \leq d(y)$).

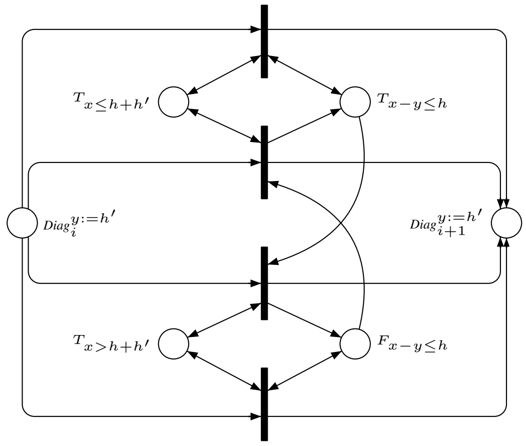

Figure 4. The subnet for $x - y \leq h$ and $y := h'$

By the way, notice that for each diagonal constraint $x - y \leq h$ and for each reset $y := h'$, the size of the corresponding TPN is constant (see Fig. 4). The number of such subnets is proportional to the number of combinations of a diagonal constraint with a reset, that is in the worst case quadratic. If we consider only diagonal constraints and resets to 0, this number will be linear.

Encoding transitions of the TA. With each location ℓ of the automaton, we associate an eponymous place ℓ in the TPN. The place ℓ is initially marked when the location ℓ is the initial one. To simulate an edge $e = (\ell, \gamma, a, R, \ell')$, we must check that the atomic constraints $(\gamma_i)_{1\leq i\leq m(e)}$ are satisfied (if $\gamma = \gamma_1 \wedge \ldots \wedge \gamma_{m(e)}$). To that aim, we use the places T_{γ_i} of the corresponding clock evolution subnets. Then, we successively update the subnets according to the resets R (where $R = (x_1 := c_1 \wedge \ldots \wedge x_{n(e)} := c_{n(e)})$).

This is done by the subnet in Fig. 5 for a transition $e = (\ell, x > c_3 \wedge y \leq c_2, a, x := c_1, \ell')$. Note that we label the transition fire_e by the letter a (notation "fire_e, a"). Note also that the place corresponding to the clock position ($\mathit{At}^x_{c_1}$) is marked at the end of the computation of this subnet.

This subnet has size linear in the size of the original TA.

Our construction is different from the one proposed in [6]. The way time elapsing and clock evolutions are handled is for example different: instead of having one small subnet per clock constraints appearing in the TA, we have only one subnet per clock which encodes its value. This method requires a more involved construction for updating the truth value of the constraints without having a blowup

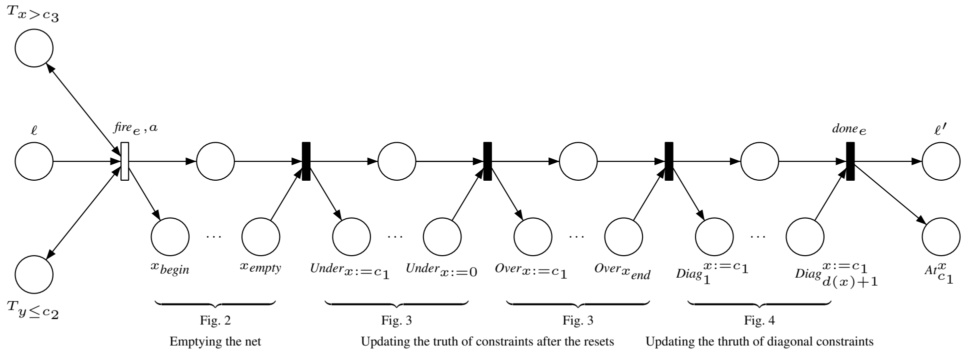

Figure 5. Simulating the transition $e = (\ell, x > c_3 \wedge y \le c_2, a, x := c_1, \ell')$

in the size of the TPN, but allows to deal with diagonal constraints and with more resets of clocks to integral values.

3.2 The correctness proof

The correctness proof relies on the existence of two simulations, one implying the inclusion of the language accepted by the TA into the language accepted by the TPN, and the other one implying the converse inclusion. Let $\mathcal{A}$ be an extended TA and $\mathcal{N}$ be the net obtained applying the construction described in the previous part.

Proof of $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{N})$**.** We define a relation $\mathcal{R}$ between configurations of the TA and the TPN as follows: $(\ell, v)\mathcal{R}(M, \nu)$ iff the following conditions are fulfilled. Let x be a clock and $\mathcal{C}_\mathcal{N}(x) = \{c_1, \ldots, c_n\}$ the set of constants related to x, occurring in the net (these values are supposed to be sorted). Let $c(x) = \inf\{c_j \mid c_j \ge v(x)\}$ with the convention that $c(x) = \infty$ if the set is empty. Then:

- If $c(x) = v(x)$ then $M(\mathit{At}^x_{c(x)}) = 1$;
- Otherwise, if $v(x) < c(x) < \infty$, then $M(\mathit{Before}^x_{c(x)}) = M(I_{x<c(x)}) = 1$, and $\nu(\mathit{Reach}^x_{c(x)}) = \nu(\mathit{Unsat}_{x<c(x)}) = v(x) - c(x)$;
- Otherwise, $M(\mathit{After}^x_{c_n}) = 1$ ($c(x) = \infty$).

For every place $T_{x\bowtie c}$ such that $v(x) \bowtie c$, $M(T_{x\bowtie c}) = 1$. For every place $F_{x<c}$ such that $\neg(v(x) < c)$, $M(F_{x<c}) = 1$. For every place $T_{x-y\bowtie c}$ such that $v(x) - v(y) \bowtie c$, $M(T_{x-y\bowtie c}) = 1$. For every place $F_{x-y\bowtie c}$ such that $\neg(v(x) - v(y) \bowtie c)$, $M(F_{x-y\bowtie c}) = 1$. Finally, $M(\ell) = 1$. The marking of remaining places is null, and the age of the remaining enabled transitions may be any admissible value.

We first observe that $(\ell_0, \mathbf{0})\mathcal{R}(M_0, \nu_0)$, and assume that $(\ell, v)\mathcal{R}(M, \nu)$.

First case, simulation of a delay move $(\ell, v) \xrightarrow{d} (\ell, v + d)$. Let X' be the subset of clocks x such that $v(x) \in \mathcal{C}_\mathcal{N}(x)$. Let X'' be the subset of clocks $x \notin X'$ such that $\inf\{c - v(x) \mid c \in \mathcal{C}_\mathcal{N}(x) \wedge c > v(x)\}$ is minimal. We note τ this value (note that $\tau = \infty$ if $X'' = \emptyset$) and $c(x)$ the constant associated with $x \in X''$.

We decompose the delay move such that we only need to successively examine the following cases.

- $X' = \emptyset$ and $d < \tau$. Then $(M, \nu) \xrightarrow{d} (M, \nu + d)$ and $(\ell, v + d)\mathcal{R}(M, \nu + d)$
- $X' = \emptyset$ and $d = \tau$. First, for every $x \in X''$, we fire the transition $\mathit{Unsat}_{x<c(x)}$. Then we let a duration d elapse and finally for every $x \in X''$, we fire $\mathit{Reach}^x_{c(x)}$. The reached configuration (M', ν') fulfills $(\ell, v + d)\mathcal{R}(M', \nu')$.
- $X' \ne \emptyset$ and $d < \tau$. First, for every $x \in X'$, we fire the transition $\mathit{Leave}^x_{v(x)}$. Then we let a duration d elapse and finally for every $x \in X'$, we fire $\mathit{Sat}_{x>v(x)}$. The configuration (M', ν') which is reached is such that $(\ell, v + d)\mathcal{R}(M', \nu')$.

Second case, simulation of a discrete move $(\ell, v) \xrightarrow{a} (\ell', v')$. We "execute" the simulation net associated with the corresponding transition e: we fire transition fire_e and for each reset of clock x (following the order defined by the net), we unmark the clock subnet of x and mark it appropriately. Then we update the places related to the diagonal constraints where x occurs. Finally we mark place ℓ' and, for every reset of clock x, the places $\mathit{Before}^x_{c_1(x)}$ where $c_1(x)$ is the first constant related to x. This configuration (M', ν') is such that $(\ell', v')\mathcal{R}(M', \nu')$.

Proof of $\mathcal{L}(\mathcal{N}) \subseteq \mathcal{L}(\mathcal{A})$**.** Let (M, ν) be a reachable configuration of the net. Note that $\Sigma_{\ell\in L} M(\ell) \le 1$. A configuration with $\Sigma_{\ell\in L} M(\ell) = 1$ will be called *tangible* and otherwise *vanishing*. Given a vanishing configuration (M, ν), (M', ν') is called a tangible successor of (M, ν) iff it is the first tangible configuration encountered in some firing sequence starting from (M, ν). Note that the differences between two tangible successors (M', ν') and (M'', ν'') may only be of the following kinds: a transition Reach_c^x, Leave_c^x, $\mathit{Sat}_{x>c}$ or $\mathit{Unsat}_{x<c}$ is fireable in one marking and just fired in the other.

We define a relation $\mathcal{R}$ between configurations of the automaton and the net as follows. $(\ell, v)\mathcal{R}(M, \nu)$ iff

- either (M, ν) is tangible and the following conditions are fulfilled. First, $M(\ell) = 1$. If $M(\mathit{At}_c^x) = 1$ then $v(x) = c$. If $M(\mathit{Before}_c^x) = 1$ then $v(x) = c' + \nu(\mathit{Reach}_c^x)$ where c' is the constant preceding c or 0 if c is the first one. If $M(\mathit{After}_c^x) = 1 \wedge M(I_{x>c}) = 1$ then $v(x) = c + \nu(\mathit{Sat}_{x>c})$. If $M(\mathit{After}_c^x) = 1 \wedge M(T_{x>c}) = 1$ then $v(x) > c$.
- or (M, ν) is vanishing and $(\ell, v)\mathcal{R}(M', \nu')$ for some (M', ν') tangible successor of (M, ν).

The critical observation (obtained by induction) is that when (M, ν) is tangible, $(\ell, v)\mathcal{R}(M, \nu)$ and $M(T_{cond}) = 1$ then $v \vDash cond$.

First observe that $(\ell_0, \mathbf{0})\mathcal{R}(M_0, \nu_0)$, and assume that $(\ell, v)\mathcal{R}(M, \nu)$.

First case, simulation of a delay move $(M, \nu) \xrightarrow{d} (M, \nu + d)$. Then $(\ell, v) \xrightarrow{d} (\ell, v + d)$ and $(\ell, v + d)\mathcal{R}(M, \nu + d)$ since (M, ν) is necessarily a tangible configuration.

Second case, simulation of a discrete move $(M, \nu) \xrightarrow{t} (M', \nu')$. If t is not a transition fire_e, then $(\ell, v)\mathcal{R}(M', \nu')$. If $t = \mathit{fire}_e$ for some $e = (\ell, \gamma, a, R, \ell')$, then the place ℓ is marked and for every T_{cond}, input place of t, one has $v \vDash cond$. Thus $(\ell, v) \xrightarrow{e} (\ell', v')$ and $(\ell', v')\mathcal{R}(M', \nu')$ since $(\ell', v')\mathcal{R}(M'', \nu'')$ where the latter configuration of the net is obtained by simulating the transition e in the net as already described.

3.3 Complexity results

Proposition 1 (From extended TA to TPN)**.** *Let $\mathcal{A}$ be an extended TA, then there is a Safe-TPN $\mathcal{N}$ equivalent to $\mathcal{A}$ w.r.t. their timed language. The size of this TPN, and the time complexity of this translation, depends on the class to which $\mathcal{A}$ belongs. This complexity is quadratic in general, and linear if $\mathcal{A}$ is either diagonal-free or* 0*-reset.*

Proof. We consider an extended TA $\mathcal{A}$. The size of the TPN built previously is the sum of the sizes of all the subnets we have described. First, we use exactly one place to encode each location of $\mathcal{A}$. Secondly, the subnets encoding the transitions of $\mathcal{A}$ have a size linear in the size of the transition they encode (see Fig. 5). Finally, the sum of the sizes of all the subnets (clock evolution subnet, emptying subnet, marking subnet, diagonal constraint subnets) related to a clock x is linear in the number $N_{\mathit{atomic}}(x)$ of non-diagonal constraints involving x we have to encode to simulate $\mathcal{A}$. Indeed, the clock evolution (Fig. 1), emptying (Fig. 2) and marking subnets (Fig. 3) have all a size linear in $N_{\mathit{atomic}}(x)$, and the diagonal constraint subnet is of constant size, but may appear in number linear in $N_{\mathit{atomic}}(x)$. The total size of our construction is thus linear in the number N_{atomic} of non-diagonal constraints we have to encode. As argued in the presentation of the construction, this number is either linear or quadratic in the size of $\mathcal{A}$, depending on whether $\mathcal{A}$ simultaneously uses both diagonal constraints and arbitrary resets to integral values. This concludes the proof. $\square$

In [12], it is proved that timed automata using diagonal constraints (and also timed automata using resets to integral values) are exponentially more concise than classical TA. Applying this conciseness result and using the linear-time transformation described above, we get the following conciseness result for Safe-TPNs:

Corollary 1 (Conciseness of TPNs: a lower bound)**.** *There is a family of Safe-TPNs $\{\mathcal{N}_k\}_{k\in\mathbb{N}}$ such that the size of $\mathcal{N}_k$ is $\mathcal{O}(k^2 \log(k))$ and such that any diagonal-free and* 0*-reset TA $\mathcal{A}_k$ equivalent to $\mathcal{N}_k$ (w.r.t. their timed languages) has a size at least 2^k.*

Example. The TPN in Fig. 6 recognizes the timed language $\{(a, t_i)_{1\leq i\leq 2^k} \mid t_i < t_{i+1}\}$. Using a slight adaptation of [12], we can prove that this language needs an exponential number of locations in a timed automaton to be accepted. However it is accepted by the TPN $\mathcal{N}_k$ depicted on Fig. 6 whose size is in $\mathcal{O}(k^2 \log(k))$. This TPN implements somehow the increment of a binary counter (each line of the TPN corresponds to one bit: if the token is in the place on the left, the corresponding bit is 0, whereas it is 1 if the token is in the place on the right).

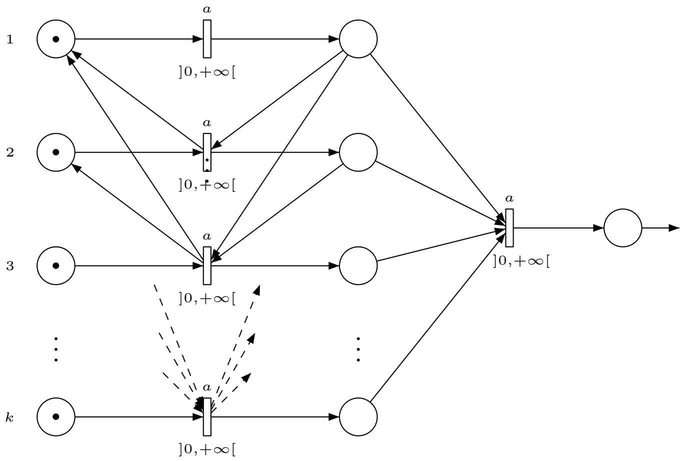

Figure 6. The TPN $\mathcal{N}_k$

Finally, this conciseness result is optimal as there is also an exponential lower bound on the size of TA equivalent to Safe-TPNs, as proved in [21]:

Proposition 2 (Conciseness of TPNs: an upper bound)**.** *Let $\mathcal{N}$ be a Safe-TPN then there is a diagonal-free and* 0*-reset $\mathcal{A}$ equivalent to $\mathcal{N}$ (w.r.t. their timed languages) whose size is exponential in the size of $\mathcal{N}$. Furthermore the time complexity of the translation is exponential in the size of $\mathcal{A}$.*

4 From Parallel Composition of TA to TPNs

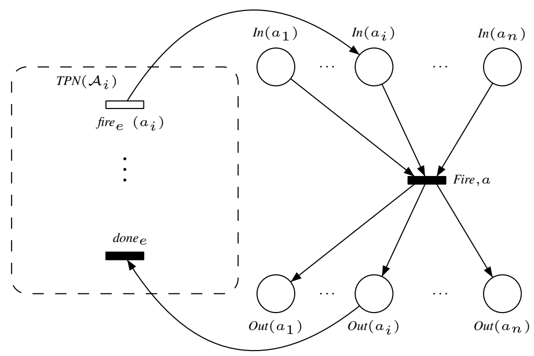

Figure 7. Simulating the parallel composition

In this section, we describe the construction of a Safe-TPN "equivalent" to a parallel composition of TA w.r.t. timed languages. The formal proof of correctness is omitted, due to lack of space.

We assume we are given a family $(\mathcal{A}_i)_{1\leq i\leq n}$ of n TA, and an n-ary synchronization function f. We assume we have built for every TA $\mathcal{A}_i$ a corresponding TPN *TPN*$(\mathcal{A}_i)$, according to the construction presented in the previous section. For each rule of the synchronization function, we add a subnet in order to "synchronize" the corresponding transitions of the TPNs *TPN*$(\mathcal{A}_i)$. We explain here the construction depicted in Fig. 7. First, for every $1 \leq i \leq n$, and for every letter a_i appearing in automaton $\mathcal{A}_i$, we add two places *In*(a_i), and *Out*(a_i), which we connect to the corresponding transitions *fire*$_e$ (originally labeled by a_i in $\mathcal{A}_i$) and *done*$_e$ of *TPN*$(\mathcal{A}_i)$, for every transition e of $\mathcal{A}_i$ labeled by a_i. Then, for every rule $f(a_1, \ldots, a_i, \ldots, a_n) = a$, we build a new immediate transition "*Fire*" labeled by a which checks every input places corresponding to the labels that have to synchronize, and returns the tokens to the corresponding output places.

It is easy to check that this construction is linear in the size of the synchronization function f, and linear in the size of the TPNs *TPN*$(\mathcal{A}_i)$.

Finally note that this construction may induce new deadlocks in the TPN (as a label a_i may be able to synchronize with other labels in different ways), but this has no effect on the timed language which is accepted.

5 Conclusion and Future Work

In this paper, we have studied the relative expressiveness and conciseness of time Petri nets and different extensions of timed automata w.r.t. timed language equivalence. More precisely, we have designed a polynomial translation from a TA with diagonal constraints and resets to integral values to a TPN. This translation becomes linear whenever TA are either diagonal-free or 0-reset. As a consequence of this translation, we get that TPNs are exponentially more concise than classical TA (thus diagonal-free and 0-reset), and we have provided a concrete family of TPNs which witnesses this conciseness property.

We are currently investigating extensions of this work. For example, we believe that an appropriate adaptation of the translation would also handle several other extensions of TA [13].

An obvious perspective for these constructions is to use algorithms developed for analyzing TPNs as an alternative for the analysis of parallel composition of extended TA. We explain below the main differences between two fundamental approaches used to verify TA and TPNs, and we explain what we may expect from the translation we have proposed in this paper.

Analysis in TA: the forward algorithm. In practice, the verification of reachability properties in timed automata is done using symbolic on-the-fly algorithms manipulating *zones*[2] [19, 15, 11]. In particular, the forward analysis computation (which consists in computing iteratively the successors of the initial configurations) is very important and is for example implemented in the much used tool Uppaal [20]. This algorithm may be presented as the construction of the "zone graph". Each vertex of this graph is a pair composed by a location ℓ and a zone $\mathcal{Z}$ in which we store the possible values of the clocks. In order to build the successors of a vertex, we proceed as follows for every transition $t = \ell \xrightarrow{g,a,R} \ell'$.

- We compute the successor of the zone $\mathcal{Z}$, by letting time elapse, taking the intersection with the constraints of the guard, and finally updating the values of clocks that are reset. If the resulting zone is consistent, we canonize it, and update the value of the location by this of the target location ℓ' of the transition.
- Since the values of clocks are in general not bounded, and in order to ensure termination of the algorithm, we have to replace the zone $\mathcal{Z}$ by an abstraction $\mathcal{Z}'$ of it.
- Then the resulting vertex $(\ell', \mathcal{Z}')$ is the successor (in the computation) by the firing of t.

Unfortunately, the abstraction operator used in step 2 must be carefully chosen [11]. For classical timed automata,

[2] A *zone* is a set of valuations defined by a clock constraints.

the choice of a good operator is rather simple, but for *extended timed automata* (*i.e.* timed automata using diagonal constraints and more general resets of clocks), it is quite intricate to find a correct abstraction operator and even to propose a correct forward analysis algorithm [11, 4, 14].

Analysis of TPNs: the class graph. The class graph is an abstraction of the transition system corresponding to another semantics of TPNs. In this semantics, the timed values we store refer to the future of the execution: each vertex of this graph is a pair composed by a marking M and a zone $\mathcal{Z}$ where a variable x_t represents the firing delay associated with an enabled transition t and an extra variable x_0 whose value is always 0. In order to build the successors of a vertex, we proceed as follows for every enabled transition t.

- We add to $\mathcal{Z}$ the constraints $x_t \leq x_{t'}$ for all $t' \neq t \in$ *En*(m), and we check whether the resulting set of constraints is consistent. We canonize the new set of constraints and we compute the new marking M'.
- We remove the variables corresponding to a transition which has been disabled, we modify the constraints corresponding to transitions t' which remain enabled, to express that $x_{t'}$ is now the remaining delay after firing t. At that point, some constraints are not constraints of a zone: we remove variable x_t applyingh a Fourier-Motzkin elimination and obtain a new zone.
- We introduce the variables $x_{t''}$ corresponding to the newly enabled transitions t'' with the constraints expressing that $x_{t''} \in I(t'')$.
- We finally canonize this zone and we get a new zone $\mathcal{Z}'$ s.t. $(M', \mathcal{Z}')$ is the successor zone when firing t.

The key point for termination of this algorithm (when the net is bounded) is that constants appearing in the zones are bounded by the maximal finite bound of the firing intervals. Thus contrary to the zone graph of a TA, no abstraction mechanism is required in order to ensure termination.

Discussion. The constructions we have proposed in this paper suggests a unified simple method for verifying parallel compositions of extended TA: first transform the system into a TPN, and then apply the class graph algorithm (using for example the tool Tina already mentioned). The advantages of this method are the following: 1) it avoids *ad-hoc* techniques for enforcing termination of forward analysis; 2) it may help tackling the state explosion problem due to parallel composition as techniques well-suited for analyzing TPNs might be efficient to handle this parallel composition.

Let us point out a main difference between these two methods: whereas the zone graph somehow computes constraints on the dates of past events, the class graph computes constraints on the dates at which will happen events in the future by storing constraints on the firing dates of transitions. We thus think it is relevant to compare these two points of view, and we currently implementing the constructions we have proposed in this paper to compare the two methods. If the results of our experiments are encouraging, we plan to propose an algorithm which would compute a sort of class graph directly on TA (without first transforming it to an equivalent TPN).

References

[1] R. Alur and D. Dill. Automata for modeling real-time systems. In *Proc. ICALP'90*, vol. 443 of *LNCS*, p. 322–335. Springer, 1990.

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

[3] T. Aura and J. Lilius. A causal semantics for time Petri nets. *Theoretical Computer Science*, 243(1–2):409–447, 2000.

[4] J. Bengtsson and W. Yi. Timed automata: Semantics, algorithms and tools. In *Proc. ACPN'03*, vol. 3098 of *LNCS*, p. 87–124. Springer, 2004.

[5] B. Bérard, F. Cassez, S. Haddad, D. Lime, and O. H. Roux. Comparison of different semantics for time Petri nets. In *Proc. ATVA'05*, vol. 3707 of *LNCS*, p. 293–307. Springer, 2005.

[6] B. Bérard, F. Cassez, S. Haddad, D. Lime, and O. H. Roux. Comparison of the expressiveness of timed automata and time Petri nets. In *Proc. FORMATS'05*, vol. 3829 of *LNCS*, p. 211–225. Springer, 2005.

[7] B. Bérard, V. Diekert, P. Gastin, and A. Petit. Characterization of the expressive power of silent transitions in timed automata. *Fundamenta Informaticae*, 36(2–3):145–182, 1998.

[8] B. Bérard and C. Dufourd. Timed automata and additive clock constraints. *Information Processing Letters*, 75(1–2):1–7, 2000.

[9] B. Berthomieu and M. Diaz. Modeling and verification of time dependent systems using time Petri nets. *IEEE Transactions in Software Engineering*, 17(3):259–273, 1991.

[10] B. Berthomieu, P.-O. Ribet, and F. Vernadat. Construction of abstract state spaces for Petri nets and time Petri nets. *International Journal of Production Research*, 42(14):2741–2756, 2004.

[11] P. Bouyer. Forward analysis of updatable timed automata. *Formal Methods in System Design*, 24(3):281–320, 2004.

[12] P. Bouyer and F. Chevalier. On conciseness of extensions of timed automata. *Journal of Automata, Languages and Combinatorics*, 2005. To appear.

[13] P. Bouyer, C. Dufourd, E. Fleury, and A. Petit. Updatable timed automata. *Theoretical Computer Science*, 321(2–3):291–345, 2004.

[14] P. Bouyer, F. Laroussinie, and P.-A. Reynier. Diagonal constraints in timed automata: Forward analysis of timed systems. In *Proc. FORMATS'05*, vol. 3829 of *LNCS*, p. 112–126. Springer, 2005.

[15] C. Daws and S. Tripakis. Model-checking of real-time reachability properties using abstractions. In *Proc. TACAS'98*, vol. 1384 of *LNCS*, p. 313–329. Springer, 1998.

[16] E. Fersman, P. Petterson, and W. Yi. Timed automata with asynchrounous processes: schedulability and decidability. In *Proc. TACAS'02*, vol. 2280 of *LNCS*, p. 67–82. Springer, 2002.

[17] G. Gardey, D. Lime, M. Magnin, and O. H. Roux. Romeo: A tool for analyzing time Petri nets. In *Proc. CAV'05*, vol. 3576 of *LNCS*, p. 418–423. Springer, 2005.

[18] S. Haar, F. Simonot-Lion, L. Kaiser, and J. Toussaint. Equivalence of timed state machines and safe time Petri nets. In *Proc. WoDES'02*, p. 119–126, 2002.

[19] K. G. Larsen, P. Pettersson, and W. Yi. Model-checking for real-time systems. In *Proc. FCT'95*, vol. 965 of *LNCS*, p. 62–88. Springer, 1995.

[20] K. G. Larsen, P. Pettersson, and W. Yi. Uppaal in a nutshell. *Journal of Software Tools for Technology Transfer*, 1(1–2):134–152, 1997.

[21] D. Lime and O. H. Roux. State class timed automaton of a time Petri net. In *Proc. PNPM'03*, p. 124–133. IEEE Computer Society Press, 2003.

[22] P. M. Merlin. *A Study of the Recoverability of Computing Systems*. PhD thesis, University of California, Irvine, CA, USA, 1974.

[23] C. Ramchandani. *Analysis of Asynchronous Concurrent Systems by Timed Petri Nets*. PhD thesis, MIT, Cambridge, MA, USA, 1974.