Computing the State Class Timed Automaton of a Time Petri Net

IRCCyN internal report - RI2003_5

Didier Lime and Olivier (H.) Roux*

IRCCyN (Institut de Recherche en Communication et Cybernétique de Nantes)
1, rue de la Noë B.P. 92101
44321 NANTES cedex 3 (France)
{Didier.Lime | Olivier-h.Roux}@irccyn.ec-nantes.fr

Abstract. This report describes a method for building the state class graph of a bounded time Petri net (TPN) as a timed automaton (TA). We consider bounded TPN, whose underlying net is not necessarily bounded. We prove that the obtained TA is timed-bisimilar to the initial TPN. The number of clocks of the automaton is much lower than with other methods available. We have implemented the method, and give some experimental results in this report, illustrating the efficiency of the translation algorithm in terms of clock number.

Key words: Time Petri nets, timed automata, model-checking, dense-time systems.

1 Introduction

Currently, the use of real-time systems is quickly increasing and, at the same time, correctness proofs on these systems must be provided. In the class of timed discrete event systems, time extensions of Petri nets (see (Bowden 1996) for a survey) and timed automata (TA) (Henzinger *et al.* 1994) are widely used to model and analyze such concurrent systems. In this report we consider transition-time Petri nets (TPN) (Merlin 1974), in which time constraints are expressed as time intervals on the transitions of the net, for they have the advantage of being able to express both concurrency and dense-time constraints in a natural way.

The act of verification consists of proving that a formal system description satisfies certain desirable properties formalized as a logical formula. This generally implies the investigation of all or a part of the state-space. However, for dense-time models such as time Petri nets or timed automata, the state-space is infinite because of the real-valued clocks; but it is possible to represent the infinite state-space by a finite partitioning in the state class graph or the region graph.

In a region graph, each region essentially consists of the set of concrete states that are equivalent, in the sense that they will evolve to the same regions in the future. Subsequently, techniques have been sought to develop algorithms that use compact abstract representations of the state-space. Moreover, it has been shown that model-checking for Timed Computation Tree Logic (TCTL) properties is decidable for TA ((Alur *et al.* 1993)). Consequently, there exists several efficient tools like Uppaal (Larsen *et al.* 1997) and Kronos (Yovine 1997) for model-checking TA.

Contrary to TA, the number of discrete states of the TPN (markings) is not necessarily bounded. Recent work (Abdulla and Nylén 2001, de Frutos Escrig *et al.* 2000) consider timed arc Petri nets where each token has a clock representing his "age". They prove that coverability and boundedness are decidable for this class of Petri nets by applying a backward exploration technique (Abdulla and Jonsson 2001, Finkel and Schnoebelen 1998). However, they assume a lazy (non-urgent) behaviour of the net. This means that the firing of transitions may be delayed, even if that implies that some transitions are disabled because their input tokens become too old.

For the model we consider, classical transition-time Petri nets, boundedness is undecidable, and works on this model report undecidability results, or decidability under the assumption that the TPN is bounded (as for reachability decidability (Popova 1991)). Boundedness and other results are obtained by computing the state-space.

State class graph

The mainstream approach to compute the space-state of TPN is the state class graph (Menasche 1982, Berthomieu and Diaz 1991). The nodes of the state class graph are sets of states (a state is a pair consisting of a marking and a firing constraint) of the TPN and the edges are labeled with transitions names. If L is the language accepted by the state class graph and L' is the untimed language accepted by the TPN, then $L = L'$. Some algorithms have been developed in order to explore the state class graph for the verification of specific temporal properties (Toussaint *et al.* 1997) (Delfieu *et al.* 2000). But mainly, the state class graph (without the use of observers (Toussaint *et al.* 1997) that specify a property as a TPN) is used to check untimed reachability properties. An alternative approach has been proposed by Yoneda (Yoneda and Ryuba 1998) in the form of an extension of equivalence classes which allows Computation Tree Logic (CTL) model-checking. Lilius (Lilius 1999) refined this approach so that it becomes possible to apply partial order reduction techniques that have been developed for untimed systems.

Our approach for the verification of TPN consists of translating a TPN into a TA or a composition of TA, in order to use the efficient algorithms and tools available for that model.

Related work

The relationship between TPN and TA is investigated in (Bornot *et al.* 1998) (Sifakis and Yovine 1996) (Sava 2001). In (Sifakis and Yovine 1996), J. Sifakis and S. Yovine are interested in a subclass of 1-safe time stream Petri nets (STPN). For a STPN, given a transition T, an interval $[l, u]$ is associated with each input arc (P, T) of T. A token entering the input place P must wait for a time t ($t \in [l, u]$) before becoming available for the transition T. They propose a translation of STPN into timed automata : a clock is associated with each place ; each time the place receives a token, the clock is reseted. Time intervals on arcs are represented by timing constraints on these clocks. Then, they show that the usual notion of composition used for TA is not suitable to describe this type of Petri nets as the composition of TA. Consequently, they introduce timed automata with deadlines and a flexible notion of composition. While it might permit a lower number of clocks, it is not obvious that the composition result on the considered subclass of 1-safe time stream Petri nets is applicable to TPN. Moreover, translating a given TPN into a TA using these results (which is not the main concern of that paper) would require a *decomposition* of the TPN, which seems quite difficult except for simple structures.

In (Bornot *et al.* 1998) authors consider Petri nets with deadlines (PND) that are 1-safe Petri nets extended with clocks. A PND is a timed automaton with deadlines (TAD) where the discrete transition structure is the corresponding marking graph. The transitions of the marking graph are subject to the same timing constraints as the transitions of the PND. The PND and the TAD have the same number of clocks. They propose a translation of safe TPN into PND with a clock for each input arc of the initial TPN. It defines (by transitivity) a translation of TPN into TAD (that can be considered as standard timed automata). The number of clocks of the TA is greater than (or equal to) the number of transitions of the initial TPN.

Sava (Sava 2001) considers bounded TPN where the associated underlying Petri net is not necessarily safe and proposes an algorithm to compute the region graph of a TPN. The result is a timed automaton with a clock for each transition of the original TPN. This automaton is then restricted for the command by supervision. However, they do not give any result to stop the automaton computation when the TPN is not bounded (which one does not know *a priori*).

The first two approaches are limited to Petri nets whose underlying net is 1-safe. Moreover, in those three approaches, the high number of clock makes it difficult to efficiently analyze the obtained TA.

Our contribution

Our approach consists of building the state class graph as a timed automaton thus keeping the temporal information of the TPN in additional clocks. That makes it possible to preserve all the properties of the state class graph construction : off-line sufficient condition of boundedness and on the fly necessary conditions of unboundedness. The initial TPN and the obtained TA are timed-bisimilar. Obtaining a TA instead of a graph is a great improvement (TCTL-model checking...) for a low additional cost. Moreover, we take great care of keeping the number of clocks of the TA low (in practice, often much lower than the number of transitions of the initial TPN). We have developed a tool for building this automaton and transcribe it in Uppaal and Kronos input formats.

Outline of the report

In this article, we first give a formal semantics for time Petri nets in terms of timed transition systems and we present the state class graph and its construction in section 2. Then, in section 3, we propose an extension of this construction that allows to build the state class graph as a timed automaton. We prove that this timed automaton and the TPN are timed-bisimilar and we also prove a relative minimality of the number of clocks of the obtained automaton. Finally, in section 4, we give a brief description of the tool implementing the algorithm and some experimental results.

2 Time Petri nets and state class graph

2.1 Time Petri nets

Definition 1 (Time Petri net). *A time Petri net is a 7-tuple*
$\mathcal{T} = (P, T, {}^\bullet(.), (.)^\bullet, \alpha, \beta, M_0)$*, where*

- $P = \{p_1, p_2, \dots, p_m\}$ *is a non-empty finite set of* places*,*
- $T = \{t_1, t_2, \dots, t_n\}$ *is a non-empty finite set of* transitions *($T \cap P = \emptyset$),*
- ${}^\bullet(.) \in (\mathbb{N}^P)^T$ *is the* backward incidence function*,*
- $(.)^\bullet \in (\mathbb{N}^P)^T$ *is the* forward incidence function*,*
- $M_0 \in \mathbb{N}^P$ *is the* initial marking *of the net,*
- $\alpha \in (\mathbb{Q}^+)^T$ *and* $\beta \in (\mathbb{Q}^+ \cup \{\infty\})^T$ *are functions giving for each transition respectively its* earliest *and* latest *firing times ($\alpha \leq \beta$).*

We define the semantics of time Petri nets as a *Timed Transition System* (TTS) (Larsen *et al.* 1995). In this model, two kinds of transitions may occur : *continuous* transitions when time passes and *discrete* transitions when a transition of the net fires.

A *marking* M of the net is an element of $\mathbb{N}^P$ such that $\forall p \in P, M(p)$ is the number of tokens in the place p.

A transition t is said to be *enabled* by the marking M if $M \geq {}^\bullet t$, (*i.e.* if the number of tokens in M in each input place of t is greater or equal to the valuation on the arc between this place and the transition). We denote it by $t \in enabled(M)$.

A transition t_k is said to be *newly* enabled by the firing of the transition t_i from the marking M, and we denote it by $\uparrow enabled(t_k, M, t_i)$, if the transition is enabled by the new marking $M - {}^\bullet t_i + t_i{}^\bullet$ but was not by $M - {}^\bullet t_i$, where M is the marking of the net before the firing of t_i. Formally,

$$\uparrow enabled(t_k, M, t_i) = ({}^\bullet t_k \leq M - {}^\bullet t_i + t_i{}^\bullet) \wedge ((t_k = t_i) \vee ({}^\bullet t_k > M - {}^\bullet t_i))$$

By extension, we will denote by $\uparrow enabled(M, t_i)$ the set of transitions newly enabled by firing the transition t_i from the marking M.

A *valuation* is a mapping $\nu \in (\mathbb{R}^+)^T$ such that $\forall t \in T, \nu(t)$ is the time elapsed since t was last enabled. Notice that $\nu(t)$ is meaningful only if t is an enabled transition. $\overline{0}$ is the *null valuation* such that for all marking M, $\forall t \in enabled(M), \overline{0}(t) = 0$.

Definition 2 (Semantics of a TPN). *The semantics of a time Petri net $\mathcal{T}$ is defined as a TTS $\mathcal{S}_\mathcal{T} = (Q, q_0, \rightarrow)$ such that*

- $Q = \mathbb{N}^P \times (\mathbb{R}^+)^T$
- $q_0 = (M_0, \overline{0})$
- $\rightarrow \in Q \times (T \cup \mathbb{R}) \times Q$ *is the transition relation including a continuous transition relation and a discrete transition relation.*
  - *The continuous transition relation is defined* $\forall d \in \mathbb{R}^+$ *by :*

$$(M, \nu) \xrightarrow{d} (M, \nu') \ iff \ \begin{cases} \nu' = \nu + d, \\ \forall t_k \in T, M \geq {}^\bullet t_k \Rightarrow \nu'(t_k) \leq \beta(t_k) \end{cases}$$

  - *The discrete transition relation is defined* $\forall t_i \in T$ *by :*

$$(M, \nu) \xrightarrow{t_i} (M', \nu') \ iff \ \begin{cases} M \geq {}^\bullet t_i, \\ M' = M - {}^\bullet t_i + t_i{}^\bullet, \\ \alpha(t_i) \leq \nu(t_i) \leq \beta(t_i), \\ \forall t_k, \nu'(t_k) = \begin{cases} 0 \ if \ \uparrow enabled(t_k, M, t_i), \\ \nu(t_k) \ otherwise \end{cases} \end{cases}$$

2.2 State Class Graph

Starting from the initial state of the TPN, the state-space of the net can be computed by applying the firing rules. However, as the model considered is a dense-time one, the state-space is infinite.

Consequently one needs to group states together in *state classes*. Thus, a finite

with some restrictions, see 3.2 state class graph can be generated, in order to apply formal verification techniques.

The method used to generate the state class graph has been introduced in (Menasche 1982) and (Berthomieu and Diaz 1991).

Definition 3 (State class). *A* state class C*, of a time Petri net, is a pair* (M, D) *where* M *is a marking of the net and* D *a set of inequalities called the* firing domain*.*

The inequalities in D *are of two types (Berthomieu and Diaz 1991)*

$$\begin{cases} \alpha_i \leq \theta_i \leq \beta_i \ (\forall i \textit{ such that } t_i \textit{ is enabled}), \\ -\gamma_{kj} \leq \theta_j - \theta_k \leq \gamma_{jk}, \forall j, k \textit{ such that } j \neq k \textit{ and } (t_j, t_k) \in \textit{enabled}(M) \times \textit{enabled}(M) \end{cases}$$

θ_i *is the firing time of the enabled transition* t_i *relatively to the time when the class was entered in.*

Informally speaking, the class $C = (M, D)$ *contains* the set of reachable states between the firing of two transitions. All the states of a class have the same marking and the inequalities define constraints on the firing times θ of the transitions enabled by this marking.

In order to compute a state class graph, one must be able to decide whether or not two classes are equal.

Given a set of inequalities D, we denote by $[\![D]\!]$ the set of the solutions of D.

Definition 4 (Equality of two classes). *Two classes* $C_1 = (M_1, D_1)$ *and* $C_2 = (M_2, D_2)$ *are* equal *if* $M_1 = M_2$ *and* $[\![D_1]\!] = [\![D_2]\!]$*.*

To check domains for equality, the inequalities sets will be put in some canonical form, and the canonical forms checked for equality. This method is much more efficient than solving the sets of inequalities and comparing the solutions.

Let D be the firing domain of a class C, the canonical form D^* is (Menasche 1982)

$$\begin{cases} \alpha_i^* \leq \theta_i \leq \beta_i^*, \\ \theta_j - \theta_k \leq \gamma_{jk}^* \end{cases}$$

where $\begin{cases} \alpha_i^* = Inf\{\theta_i\} \\ \beta_i^* = Sup\{\theta_i\} \\ \gamma_{jk}^* = Sup\{\theta_j - \theta_k\} \end{cases}$

The algorithm used determines the shortest paths in a constraint graph and its complexity is $O(n^3)$ in time and $O(n^2)$ in space, n being the number of variables in the set of inequalities (Berthomieu 2001)

Computing the graph.

Definition 5 (firability). *Let $C = (M, D)$ be a state class. A transition t_i is said to be* firable *from C iff in D, $\alpha_i \leq \min_{t_j \in enabled(M)} \beta_j$.*

Given a class $C = (M, D)$ and a firable transition t_f, the class $C' = (M', D')$ obtained from C by the firing of t_f is given by

- $M' = M - {}^\bullet t_f + t_f{}^\bullet$
- D' is computed along the following steps, and denoted by $next(D, t_f)$
  1. $\forall j \neq t_f$ addition of constraints $\theta_j \geq 0$,
  2. variable substitutions $\forall j, \theta_j = \theta_f + \theta'_j$,
  3. elimination (using for instance the Fourier-Motzkin method (Dantzig 1963)) of all variables relative to transitions disabled by the firing of t_f,
  4. addition of inequations relative to newly enabled transitions

$$\forall t_k \in \uparrow enabled(M, t_f), \alpha(t_k) \leq \theta'_k \leq \beta(t_k).$$

  5. determination of the canonical form D'^*

We propose to write the state class graph as a transition system $(C, C_0, \rightarrow)$ where

- $C = \mathbb{N}^P \times \mathbb{R}^T$,
- $C_0 = (M_0, D_0)$, where M_0 is the initial marking and $D_0 = \{\alpha(t_i) \leq \theta_i \leq \beta(t_i) | \, t_i \in enabled(M_0)\}$,
- $\rightarrow \in C \times T \times C$ is the transition relation defined by :

$$(M, D) \xrightarrow{t} (M', D') \text{ iff } \begin{cases} M' = M - {}^\bullet t + t^\bullet, \\ t \text{ is firable from } (M, D), \\ D' = next(D, t), \end{cases}$$

Limitations of the state class graph method. As we have mentioned in the introduction, the most widely used method for checking TPN is the construction of the state class graph, which gives the untimed language accepted by the TPN. However, as it is, the state class graph can only be used conveniently for checking untimed reachability properties. More precisely, shifting the origin of time is done by a projection and is an irreversible process. As a consequence, the state class graph does not accept the timed language of the TPN. In Figure 1, we can see that if t_1 fires at time 0 then we have the forced sequence t_2 then t_3, while if t_1 fires at time 4 then the forced sequence is t_3 then t_2. This information does not appear in class C_1, even if we add the firing domains.

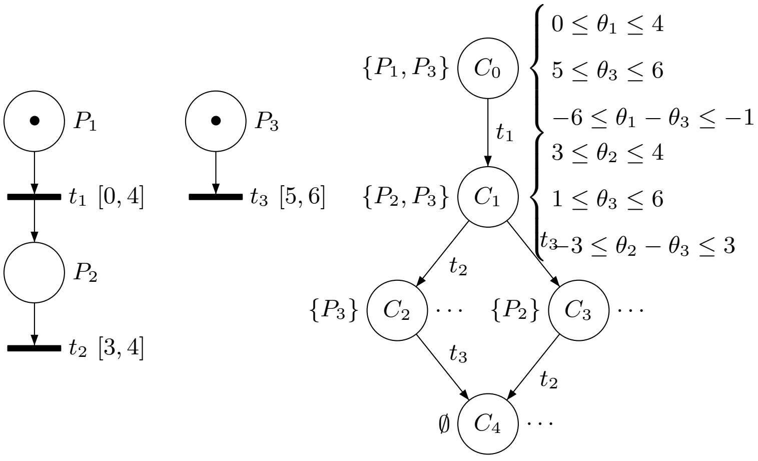

Fig. 1. A TPN and its state class graph

In order to check real-time properties the current method adds observers to the TPN and computes the state class graph of the system "TPN + observer" (Toussaint *et al.* 1997).

However the observer approach suffers from other problems. The observer may be as big as the net if the property to be verified involves markings, and so the number of classes may be squared. Furthermore, each property requires a specific observer, and thus a new computation of the state class graph.

Our method, described in the next section, overcomes all these problems : the state class timed automaton accepts the same timed language as the TPN, the verification is non-intrusive, and the state class TA needs to be computed only once even if several properties are to be checked.

3 State class timed automaton

3.1 Timed Automata

We denote by $\mathcal{C}(V)$ the set of simple constraints on the set of variables V, *i.e.* $\mathcal{C}(V)$ is the set of boolean combinations (with the operators $\vee$, $\wedge$ and $\neg$) of terms of type $v - v' \sim c$ or $v \sim c$, with $v, v' \in V$, $\sim \in \{<, \leq, =, \geq, >\}$ and $c \in \mathbb{N}$.

Definition 6 (Timed Automaton). *(Henzinger* et al. *1994) A* Timed Automaton *is a 6-tuple* (L, l_0, X, A, E, Inv) *where*

- L *is a finite set of* locations*,*
- l_0 *is the* initial location*,*
- X *is a finite set of positive real-valued* clocks*,*
- A *is a finite set of* actions*,*
- $E \subset L \times \mathcal{C}(X) \times A \times 2^X \times 2^{X^2} \times L$ *is a finite set of edges. let us consider* $e = (l, \delta, \alpha, R, \rho, l') \in E$*. e is the edge linking the location l to the location* l'*, with the* guard δ*, the action* α*, the set of clocks to be reseted R and the renaming fonction* ρ*.*
- $Inv \in \mathcal{C}(X)^L$ *maps an* invariant *to each location.*

Let ν be a valuation. For a set of clocks R, we denote by $\nu[R \leftarrow 0]$ the valuation such that $\nu[R \leftarrow 0](x) = 0$ if $x \in R$ and $\nu[R \leftarrow 0](x) = \nu(x)$, otherwise. For a renaming fonction $\rho : X \to X$, we denote by $\nu' = \nu[\rho]$ the valuation such that $\forall x \in X, \nu'(\rho(x)) = \nu(x)$. For a constraint $\delta \in \mathcal{C}(X)$, we say that the evaluation of δ by ν, $\delta(\nu)$ is true iff $\forall x \in X, \delta(\nu(x))$ is true.

Definition 7 (Semantics of a Timed Automaton). *The semantics of a Timed Automaton H is defined as a TTS* $\mathcal{S}_H = (Q, Q_0, \to)$ *where* $Q = L \times (\mathbb{R}^+)^X$*,* $Q_0 = (l_0, \overline{0})$ *is the initial state and* $\to$ *defined, for* $a \in A$ *and* $\tau \in \mathbb{R}+$*, by*

- *discrete transitions :* $(l, \nu) \xrightarrow{a} (l', \nu')$ *iff* $\exists (l, \delta, a, R, \rho, l') \in E$ *such that*
$$\begin{cases} \delta(\nu) = \texttt{true}, \\ \nu' = \nu[R \leftarrow 0][\rho], \\ Inv(l')(\nu') = \texttt{true} \end{cases}$$
- *continuous transitions :* $(l, \nu) \xrightarrow{\tau} (l, \nu')$ *iff* $\begin{cases} \nu' = \nu + \tau, \\ \forall \tau' \in [0, \tau], Inv(l)(\nu + \tau') = \texttt{true} \end{cases}$

3.2 State class timed automaton

As we mentioned before, temporal information is lost when computing the state class graph. Consequently, in order to verify complex temporal properties, we want to keep track of the time during which each transition has been enabled. In the semantics of TPN that we have shown in section 2, these times correspond to the valuation ν of the net. Moreover, it is very important to have a low number of clocks in the resulting automaton as verification algorithms complexity is exponential in this number. To address the irreversibility problem of the state class graph, we add clocks in the generated state classes which represent the valuations of the transition as given in the semantics of TPN.

Definition 8 (Extended state class). *An extended state class is a 4-tuple* $(M, D, \chi, trans)$*, where* M *is a marking,* D *is a firing domain,* χ *is a set of real-valued clocks and* $trans \in (2^T)^\chi$ *maps clocks to sets of transitions.*

For each clock $x \in \chi$, $trans$ gives the set of transitions whose valuation is represented by x. A transition t must be associated with only one clock. Then, $trans^{-1}(t)$ is reduced to a single element.

In the following paragraphs, we will define from a theoretical point of view the state class automaton $\Delta(\mathcal{T})$ of a TPN $\mathcal{T}$. We will show that its computation is based on that of an *extended* state class graph $\Delta'(\mathcal{T})$. Practical details of the algorithm computing the state class TA will be then given.

Extended state class graph We first define an extended state class graph. This is done in very much the same way as for the classical state class graph. Differences lie in the computation of χ and $trans$.

Let $disabled(M, t) = enabled(M) - enabled(M - {}^\bullet t)$ be the set of transitions disabled by the firing of t leading to the marking M. Let $C = (M, D, \chi, trans)$ be an extended state class, t a transition firable from C and $C' = (M', D', \chi', trans')$ the extended state class obtained by firing t from C. M' and D' are computed like for the classical state class graph. Computation of $trans'$ and χ' requires the following steps

1. for each clock x in χ, the disabled transitions are removed from $trans(x)$,
2. the clocks whose image by $trans$ is empty are removed from χ,
3. if there are newly enabled transitions by the firing of t, two cases can occur:
   - there exists a clock x whose value is 0. Then, we simply add the newly enabled transitions to $trans(x)$,

- such a clock does not exist. Then we need to create a new clock x_i associated to the newly enabled transitions. The index, i, is chosen as the smallest available index among the clocks of χ. We add x_i to χ and $trans(x_i)$ is the set of newly enabled transitions

Formally, the extended state class graph can be written as the following transition system : $\Delta'(\mathcal{T}) = (C^{ext}, C_0, \rightarrow^{ext})$ defined by:

- $C^{ext} = \mathbb{N}^P \times \mathbb{R}^T \times 2^X \times (2^T)^X$, X being the set of all clocks,
- $C_0 = (M_0, D_0, \chi_0, trans_0)$, where M_0 is the initial marking, $D_0 = \{\alpha(t_i) \leq \theta_i \leq \beta(t_i) | \, t_i \in enabled(M_0)\}$, $\chi_0 = \{x_0\}$ and $trans_0 = (x_0, enabled(M_0))$
- $\rightarrow^{ext} \in C^{ext} \times T \times C^{ext}$ is the transition relation defined by:
  $(M, D, \chi, trans) \overset{t}{\rightarrow}^{ext} (M', D', \chi', trans')$ iff

$$\begin{cases} t \text{ is firable from } (M, D), \\ M' = M - {}^{\bullet}t + t^{\bullet}, \\ D' = next(D, t), \\ \text{let } DC = \{x \in \chi, trans(x) - disabled(M, t) = \emptyset\}, \\ \text{if } \uparrow enabled(M, t) = \emptyset, \text{ then } \chi' = \chi - DC \text{ and } \forall x \in \chi', trans'(x) = trans(x), \\ \text{else} \begin{cases} \text{if } \exists x_j \in \chi \text{ s.t. } x_j = 0, \text{ then } \chi' = \chi - DC, \\ \begin{cases} \forall x \in \chi' - \{x_j\}, trans'(x) = trans(x), \\ trans'(x_j) = trans(x_j) \cup \uparrow enabled(M, t), \end{cases} \\ \text{else} \begin{cases} i = \min \{k \in \mathbb{N} | x_k \notin \chi - DC\}, \chi' = \chi - DC \cup \{x_i\}, \\ \forall x \in \chi' - \{x_i\}, trans'(x) = trans(x), \\ trans'(x_i) = \uparrow enabled(M, t), \end{cases} \end{cases} \end{cases}$$

In order to compute the extended state class graph we define a convergence criterion as an equivalence relation between extended state classes:

Definition 9 (Clock-similarity). *Two extended state classes $C = (M, D, \chi, trans)$ and $C' = (M', D', \chi', trans')$ are* clock-similar*, and we denote it by $C \approx C'$, iff they have the same markings, the same number of clocks and their clocks are mapped to the same transitions:*

$$C \approx C' \Leftrightarrow \begin{cases} M = M', \\ |\chi| = |\chi|, \\ \forall x \in \chi, \exists x' \in \chi', trans(x) = trans'(x'). \end{cases}$$

We can notice that a straight extension of the equivalence relation between state classes of (Berthomieu and Diaz 1991) would have been clock-similarity *and* equality of the firing domains. However, the additional information in the extended state classes makes it possible to use such a less restrictive relation for the state class timed automaton.

State class timed automaton Given the extended state class graph, we can now define the state class timed automaton $\Delta(\mathcal{T})$:

Definition 10 (State Class Timed Automaton). *The state class timed automaton* $\Delta(\mathcal{T}) = (L, l_0, X, A, E, Inv)$ *defined from the extended state class graph by:*

- L *the set of locations is the set of the extended state classes* C^{ext}*,*
- l_0 *is the initial state class* $(M_0, D_0, \chi_0, trans_0)$*,*
- $X = \bigcup_{(M,D,\chi,trans)\in C^{ext}} \chi$
- $A = T$ *is the set of transitions*
- E *is the set of edges defined as follows,*

$$\forall C_i = (M_i, D_i, \chi_i, trans_i), C_j = (M_j, D_j, \chi_j, trans_j) \in C^{ext},$$

$$\exists C_i \overset{t}{\rightarrow}^{ext} C_j \Leftrightarrow \exists (l_i, \delta, a, R, \rho, l_j) \ s.t. \begin{cases} \delta = (trans_i^{-1}(t) \geq \alpha(t)), \\ a = t, \\ R = trans_j^{-1}(\uparrow enabled(M_i, t)), \\ \forall x \in \chi_i, x' \in \chi_j, \\ \ s.t. \ trans_i(x) = trans_j(x') \\ \ and \ x' \notin R, \rho(x) = x' \end{cases}$$

- $\forall C_i \in C^{ext}, Inv(l_i) = \bigwedge_{x\in\chi_i, t\in trans_i(x)} (x \leq \beta(t))$*.*

Algorithm Computing the transition system $\Delta'(\mathcal{T})$ is done by a classical breadth first graph generation algorithm and its computation as well as that of $\Delta(\mathcal{T})$ are done simultaneously. More precisely, each time a new extended state class C' is computed, we check if it is clock-similar to a previously computed one C. If not, we create a new location associated with C', otherwise we simply create an edge between the location C and the parent location of C'. In the latter case, we also need to check if C' is included in C, according to definition 11:

Definition 11 (Inclusion between two extended state classes). *An extended state class* $C' = (M', D', \chi', trans')$ *is* included *in an extended state class* $C = (M, D, \chi, trans)$ *iff* C *and* C' *are clock-similar and* $[\![D']\!] \subset [\![D]\!]$*. This is denoted by* $C' \subset C$*.*

If $C' \subset C$, then we do not need to compute further on that branch, while if it is not, the domain D of C becomes $D \cup D'$, D' being the firing domain of C', and we compute the sons corresponding to the firable transitions in C'.

One can notice that when making a loop (*i.e.* inclusion or mere clock-similarity) a renaming might be done in the automaton on the created edge

This is not accepted by all model-checkers, but Kronos, for instance, allows it.

An example of a TPN and the corresponding state class graph TA is shown in Figure 2. For a better comprehension of this example, we also provide the TA as it would have been obtained with one clock per transition of the TPN in figure 3.

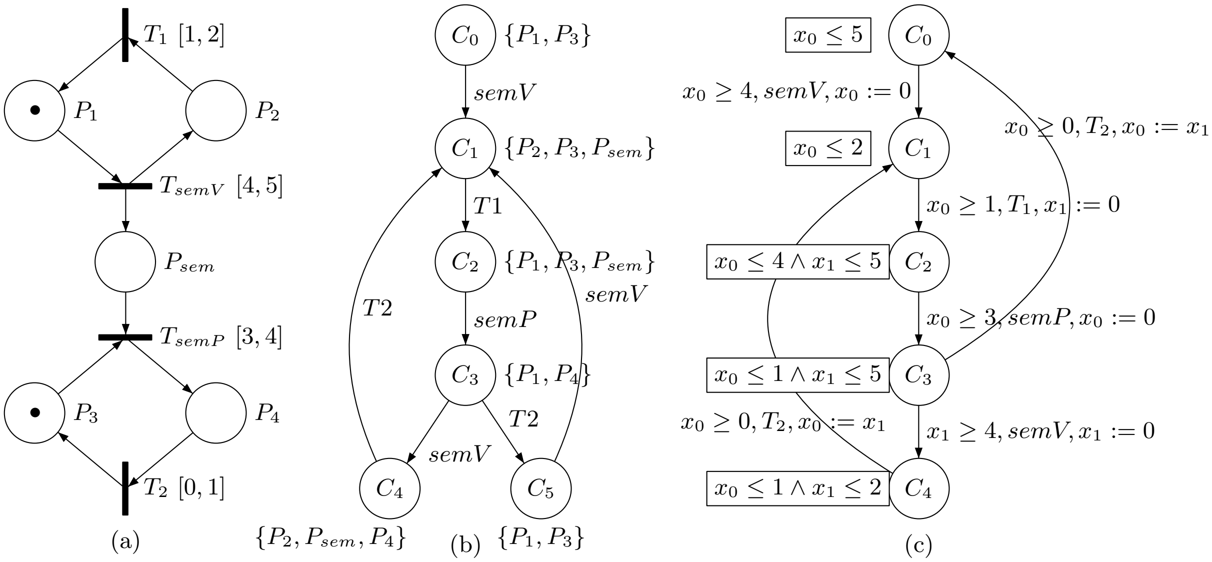
T_1 [1, 2]
P_1
P_2
T_{semV} [4, 5]
P_{sem}
T_{semP} [3, 4]
P_3
P_4
T_2 [0, 1]
(a)
C_0 $\{P_1, P_3\}$
$semV$
C_1 $\{P_2, P_3, P_{sem}\}$
$T1$
C_2 $\{P_1, P_3, P_{sem}\}$
$semV$
$T2$
$semP$
C_3 $\{P_1, P_4\}$
$T2$
$semV$
C_4
C_5
$\{P_2, P_{sem}, P_4\}$
(b)
$\{P_1, P_3\}$
$x_0 \leq 5$
C_0
$x_0 \geq 4, semV, x_0 := 0$
$x_0 \geq 0, T_2, x_0 := x_1$
$x_0 \leq 2$
C_1
$x_0 \geq 1, T_1, x_1 := 0$
$x_0 \leq 4 \wedge x_1 \leq 5$
C_2
$x_0 \geq 3, semP, x_0 := 0$
$x_0 \leq 1 \wedge x_1 \leq 5$
C_3
$x_0 \geq 0, T_2, x_0 := x_1$
$x_1 \geq 4, semV, x_1 := 0$
$x_0 \leq 1 \wedge x_1 \leq 2$
C_4
(c)

Fig. 2. Two cyclic tasks synchronized *via* a semaphore : TPN model (a), its state class graph (b) and its state class timed automaton (c)

We define a bisimulation between the TPN $\mathcal{T}$ and its state class TA $\Delta(\mathcal{T})$. That proves that the timed language accepted by $\mathcal{T}$ is the same as the timed language accepted by $\Delta(\mathcal{T})$, which allows us to check TCTL properties on $\mathcal{T}$.

Theorem 1 (Bisimulation). *Let $Q_\mathcal{T}$ be the set of states of the TPN $\mathcal{T}$ and $Q_\mathcal{A}$ the set of states of the state class timed automaton $\mathcal{A} = (L, l_0, X, A, E, Inv)$. Let $\mathcal{R} \subset Q_\mathcal{T} \times Q_\mathcal{A}$ be a binary relation such that $\forall s = (M_\mathcal{T}, \nu_\mathcal{T}) \in Q_\mathcal{T}, \forall a = (l, \nu_\mathcal{A}) \in Q_\mathcal{A}, s\mathcal{R}a \Leftrightarrow M_\mathcal{T} = M_\mathcal{A}$ if $M_\mathcal{A}$ is the marking associated with l and $\forall t \in enabled(M), \exists x \in X, \nu_\mathcal{T}(t) = \nu_\mathcal{A}(x)$.*

$\mathcal{R}$ is a bisimulation.

The proof for theorem 1 is given in Appendix A.

Finiteness of the state class timed automaton Berthomieu and Diaz have shown that a TPN has a finite number of state classes if and only if it is bounded (Berthomieu and Diaz 1991).

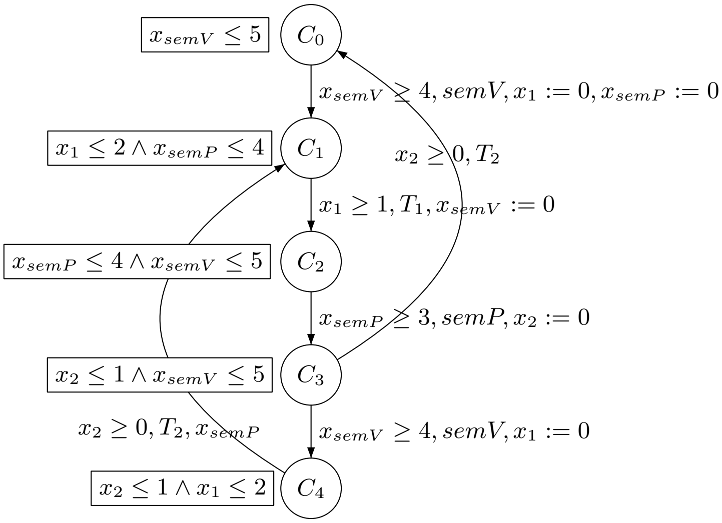
$x_{semV} \leq 5$
C_0
$x_{semV} \geq 4, semV, x_1 := 0, x_{semP} := 0$
$x_1 \leq 2 \land x_{semP} \leq 4$
C_1
$x_2 \geq 0, T_2$
$x_1 \geq 1, T_1, x_{semV} := 0$
$x_{semP} \leq 4 \land x_{semV} \leq 5$
C_2
$x_{semP} \geq 3, semP, x_2 := 0$
$x_2 \leq 1 \land x_{semV} \leq 5$
C_3
$x_2 \geq 0, T_2, x_{semP}$
$x_{semV} \geq 4, semV, x_1 := 0$
$x_2 \leq 1 \land x_1 \leq 2$
C_4

Fig. 3. TA with one clock per transition of the TPN

The proof, that relies only on markings and firing domains, is fully applicable to extended state classes and the following theorem holds:

Theorem 2. *A TPN has a finite number of extended state classes if and only if it is bounded.*

Boundedness of TPN is undecidable, though. Nonetheless we can check off-line the following sufficient condition for boundedness of the TPN. (Berthomieu and Diaz 1991)

Theorem 3. *A TPN is bounded if the underlying Petri net is bounded and the earliest and latest firing times of transitions are rationals.*

However, in most cases, the previous sufficient condition is too restrictive, *i.e.* it is not verified while the TPN is actually bounded. That is why we perform some on-line checking of unboundedness necessary conditions, in order to stop the computation if we know that the state class TA will not be finite. Again, theorem 4 is a straight extension of that of (Berthomieu and Diaz 1991).

Theorem 4. *A TPN is bounded if there does not exist a pair of extended state classes* $C = (M, D, \chi, trans)$ *and* $C' = (M', D', \chi', trans')$*, reachable from the initial extended state class, such that*

1. C' *is reachable from* C
2. $M' > M$
3. $[\![D']\!] = [\![D]\!]$
4. $\forall p \in \{p \in P, M'(p) > M(p)\}, M(p) > \max_{t \in T} {}^{\bullet}(t)(p)$

With only conditions 1 and 2 we already have a necessary condition of unboundedness. With condition 3, that necessary condition becomes stronger and as we add condition 4, the bounded TPN with a pair of classes verifying those four conditions should be very rare. However, the cost of computing conditions 3 and 4 is not to be neglected.

3.3 Number of clocks of the state class timed automaton

The number of clocks of a TA is an important factor for the efficiency of verification algorithms. In the following, we will present and prove several properties on the number of clocks of the state class TA.

In order to generate as few clocks as possible, two things are taken into account in our method. First, when transitions are simultaneously enabled, their valuations are equal on every run through which they remain continuously enabled: let us consider k transitions $t_1, \dots, t_k$ enabled at the same time τ_0. For all k, we have $\nu(t_k)(\tau_0) = 0$. Since time passes at the same rate for all transitions, all valuations remain equal while none of these k transition is fired. When one of them is fired, the common value does not represent its valuation anymore but still does for the remaining $k-1$ transitions, and so on until all the transitions are fired. Therefore we need only one of these valuations for representing those of the k simultaneously enabled transitions. More precisely, the valuation must be the one of the transition which fires the latest. Second, the unused clocks are reused when in need of a new clock for newly enabled transitions. The policy we have chosen consists of choosing the first unused clock *i.e.* the one with the smallest index.

It comes then straightforwardly that the state class TA has a number of clocks lower or equal to the maximum number of transitions enabled by the reachable markings of the TPN. For further properties we need some additional definitions.

If δ is the guard of an edge of the automaton, $op(\delta) \subset X$ is the set of clocks constrained by δ. Similarly, for a location l, $op(Inv(l)) \subset X$ is the set of clocks constrained by $Inv(l)$.

Now we define the notion of usefulness of a clock in a state.

Definition 12 (TPN-useful). $\textit{TPN-useful}(x) = \{l \in L, \exists (l, \delta, \alpha, R, l') \in E, x \in op(\delta)\}$

From that definition we can define the notion of orthogonality of two clocks and, for our particular type of TA, we can give an equivalent definition for the activity of Daws and Yovine (Daws and Yovine 1996).

Definition 13 (Orthogonaliy of two clocks). *x and y are two orthogonal clocks, denoted by $x \perp y$ iff $\textit{TPN-useful}(x) \cap \textit{TPN-useful}(y) = \emptyset$*

Definition 14 (Activity). *Let $Act(TA)$ be the set of active clocks of the timed automaton TA*

$$x \in Act(TA) \Leftrightarrow \mathit{TPN\text{-}useful}(x) \neq \emptyset$$

For our TA, it is clear that orthogonal clocks could be renamed to an unique clock name, thus reducing the total number of clocks by one. However, the following theorem holds.

Theorem 5. *There are no* orthogonal *clocks in the state class timed automaton.*

Furthermore, according to the previous definition, the automaton has no *inactive* clocks since in each state, all the used clocks appear in the invariant. Another point is that we have no *equal* clocks.

Theorem 6. *There is no location l of the state class timed automaton in which two clocks x, y used in l ($l \in \mathit{TPN\text{-}useful}(x) \cap \mathit{TPN\text{-}useful}(y)$) are equal*

The proofs for theorems 5 and 6 are given in Appendix A.

4 Experimental results

We have implemented the building of the state class timed automaton in a tool : Romeo

http://www.irccyn.ec-nantes.fr/irccyn/d/fr/equipes/TempsReel/logs/software-2-romeo that consists of a graphical interface written in TCL/Tk and a computation module, Gpn, written in C++. The input format is a XML TPN description and the computed timed automaton is given in Uppaal or Kronos input format.

The first table (Table 1) compares the efficiency, in terms of clock number, of (Bornot *et al.* 1998), (Sifakis and Yovine 1996) and (Sava 2001) and Romeo. Contrary to our method, no tool implementing the first two methods are available to our knowledge, so the figures given are computed according our understanding of those methods.

The figures given in the first column are obtained by the method of (Bornot *et al.* 1998). They correspond to one clock for each input arc of each transition. The translation of (Sifakis and Yovine 1996) gives one clock per place of the TPN. However, the model they consider is not TPN so we had to extrapolate these figures, which are given in the second column.

These first two methods are limited to 1-safe TPN. That is why some figures are missing in the two first colums ; it corresponds to examples for which the underlying Petri net is unbounded. The algorithm of (Sava 2001) has no such restriction and gives one clock per transition (column 3).

	BST98	SY96	Sav01	Sav01 + DY96	Romeo
Example 1	NA	NA	12	8	6
Example 2	13	12	10	3	3
Example 3	14	14	14	2	2
Example 4	24	24	22	2	2
Example 5	31	29	23	3	2
Example 6	10	5	10	1	1
Example 7	NA	NA	20	11	7
Example 8	NA	NA	21	11	7
Example 9	NA	NA	15	3	3
Example 10	20	31	13	3	3
Example 11	12	20	9	4	4
Example 12	16	12	13	4	4
Example 13	20	16	17	4	4
Example 14	16	20	16	4	4
Example 15	NA	NA	31	4	2
Example 16	NA	NA	17	8	2
Example 17	NA	NA	13	9	4
Example 18	NA	NA	14	10	4
Example 19	NA	NA	20	13	3
Example 20	NA	NA	16	2	2

Table 1. Number of clocks

Those three algorithms are not very efficient with regard to the number of clocks they generate. So, we have applied the off-line clock reduction algorithms of (Daws and Yovine 1996) on the results of the algorithm of Sava. The reduction obtained is very good (column 4). However, the average performance of our method given in the last column, is still quite better than these results on our set of examples. Moreover the number of clocks is always smaller than (or equal) for all the other methods.

The second table (Table 2) shows the size reduction of the state class timed automaton compared to the state class graph. The average reduction is quite important, while the additional computing cost is fairly low (while not precisely quantified yet, we believe it to be linear in the number of added clocks). Actually, thanks to the size reduction, the computing time of the TA is more often than not smaller than for the graph.

The examples we used for testing include modelization of cyclic and periodic synchronized tasks such as producers-consumers models, or the alternate bit protocol as modeled in (Berthomieu and Diaz 1991). The number of places and transitions are not given since they are not really relevant for what concerns the difficulty of analysis of TPN.

Of course, our set of examples is not exhaustive, however, we believe that these results give a good idea of what the method could give when applied on a real system modelization.

	Locations (TA)	Transitions (TA)	Nodes (Graph)	Transitions (Graph)
Example 1	123	258	355	661
Example 2	33	47	59	79
Example 3	16	16	16	16
Example 4	23	24	23	24
Example 5	48	69	159	206
Example 6	5	10	5	10
Example 7	1140	3990	14418	46079
Example 8	1277	4334	13557	41249
Example 9	58	135	252	548
Example 10	39	68	126	222
Example 11	50	123	138	330
Example 12	123	333	4256	8977
Example 13	407	1076	24401	50876
Example 14	998	3088	22016	60967
Example 15	1088	5245	1098	5260
Example 16	76	207	200	353
Example 17	735	2303	1403	3508
Example 18	1871	6356	2831	8386
Example 19	11490	50268	14086	56929
Example 20	14	20	16	22

Table 2. Number of states and transitions

5 Conclusion

In this report, we have given a method for building the state class graph of a TPN as a timed automaton. We have proved that the initial TPN and the TA obtained are timed-bisimilar Furthermore, the number of clocks of the automaton is lower or equal (in practice much lower) to the number of transitions of the initial TPN and *a fortiori* much lower than other available methods. The computation of the state class graph automaton preserves the properties of the state class graph construction : off-line sufficient condition of boundedness and on the fly necessary conditions of unboundedness. The additional cost of our algorithm compared to the state class graph computation is quite low, and the obtained TA is always smaller (most of the time much smaller) than the corresponding state class graph, so the TA is often faster to compute than the graph. We have implemented the building of the state class timed automaton in a tool: Romeo.

The bisimulation between the TPN and its state class timed automaton allows us to say that TCTL model-checking is decidable for bounded TPN. TCTL properties on the TPN can then be verified using the state class TA, thus properties are verified with very efficient tools like Uppaal or Kronos. The low number of clocks allows to efficiently check complex real-time properties. In addition, verification may still be performed in the same way as for the state class graph, by adding observers to the net in order to monitor a transition firing or the occurrence of a given marking. Since the TA is quite smaller than the corresponding state class graph, this approach becomes more efficient.

Further work includes extension to multi-enabledness of transitions as defined by Berthomieu (Berthomieu 2001), and tuning of the method to an extension of TPN, SE-TPN (Roux and Déplanche 2002), allowing the modeling of preemptive scheduling of real-time processes. We also plan to investigate the possibility of building a coverability graph, when the state class TA is not finite. The method may also allow us to specify a real-time system as a mixed model of TPN and TA, and then obtain a TA modeling the behaviour of the whole system.

References

Abdulla, P. A. and A. Nylén (2001). Timed petri nets and bqos. In: *22nd International Conference on Application and Theory of Petri Nets (ICATPN'01)*. Vol. 2075 of *Lecture Notes in Computer Science*. Springer-Verlag. Newcastle upon Tyne, United Kingdom. pp. 53–72.

Abdulla, P. A. and B. Jonsson (2001). Ensuring completeness of symbolic verification methods for infinite-state systems. *Theoretical Computer Science* **256**, 145–167.

Alur, R., C. Courcoubetis and D. L. Dill (1993). Model-checking in dense real-time. *Information and Computation* **104**(1), 2–34.

Berthomieu, B. (2001). La méthode des classes d'états pour l'analyse des réseaux temporels. In: *3e congrès Modlisation des Systèmes Réactifs (MSR'2001)*. Hermes. Toulouse, France. pp. 275–290.

Berthomieu, B. and M. Diaz (1991). Modeling and verification of time dependent systems using time petri nets. *IEEE transactions on software engineering* **17**(3), 259–273.

Bornot, S., J. Sifakis and S. Tripakis (1998). Modeling urgency in timed systems. *Lecture Notes in Computer Science* **1536**, 103–129.

Bowden, F. D. J. (1996). Modelling time in petri nets. In: *2nd Australia-Japan Workshop on Stochastic Models in Engineering, Technology and Management*. Gold Coast, Australia.

Dantzig, G. B. (1963). Linear programming and extensions. *IEICE Transactions on Information and Systems*.

Daws, C. and S. Yovine (1996). Reducing the number of clock variables of timed automata. In: *1996 IEEE Real-Time Systems Symposium (RTSS'96)*. IEEE Computer Society Press. Washington, DC, USA. pp. 73–81.

de Frutos Escrig, D., V. Valero Ruiz and O. Marroquín Alonso (2000). Decidability of properties of timed-arc petri nets. In: *21st International Conference on Application and Theory of Petri Nets (ICATPN'00)*. Vol. 1825 of *Lecture Notes in Computer Science*. Springer-Verlag. Aarhus, Denmark. pp. 187–206.

Delfieu, D., P. Molinaro and O. H. Roux (2000). Analyzing temporal constraints with binary decision diagrams. In: *25th IFAC Workshop on Real-Time Programming (WRTP'00)*. Palma, Spain. pp. 131–136.

Finkel, A. and P. Schnoebelen (1998). Fundamental structures in well-structured infinite transitions systems. In: *3rd Latin American Theoretical Informatics Symposium (LATIN'98)*. Vol. 1380 of *Lecture Notes in Computer Science*. Springer-Verlag. Campinas, Brazil. pp. 102–118.

Henzinger, T. A., X. Nicollin, J. Sifakis and S. Yovine (1994). Symbolic model checking for real-time systems. *Information and Computation* **111**(2), 193–244.

Larsen, K. G., P. Pettersson and W. Yi (1995). Model-checking for real-time systems. In: *Fundamentals of Computation Theory*. pp. 62–88.

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

Lilius, J. (1999). Efficient state space search for time petri nets. In: *MFCS Workshop on Concurrency '98*. Vol. 18 of *ENTCS*. Elsevier.

Menasche, M. (1982). Analyse des réseaux de Petri temporisés et application aux systèmes distribués. PhD thesis. Université Paul Sabatier. Toulouse, France.

Merlin, P. M. (1974). A study of the recoverability of computing systems. PhD thesis. Department of Information and Computer Science. University of California, Irvine, CA.

Popova, L. (1991). On time petri nets. *Journal Information Processing and Cybernetics, EIK* **27**(4), 227–244.

Roux, O. H. and A.-M. Déplanche (2002). A t-time petri net extension for real time-task scheduling modeling. *European Journal of Automation (JESA)*.

Sava, A. T. (2001). Sur la synthèse de la commande des systèmes à évènements discrets temporisés. PhD thesis. Institut National polytechnique de Grenoble. Grenoble, France.

Sifakis, J. and S. Yovine (1996). Compositional specification of timed systems (extended abstract). In: *13th Syposium on Theoretical Aspects of Computer Science*. Springer-Verlag. Grenoble, France. pp. 347–359.

Toussaint, J., F. Simonot-Lion and Jean-Pierre Thomesse (1997). Time constraint verifications methods based time petri nets. In: *6th Workshop on Future Trends in Distributed Computing Systems (FT-DCS'97)*. Tunis, Tunisia. pp. 262–267.

Yoneda, T. and H. Ryuba (1998). CTL model checking of time petri nets using geometric regions. *IEICE Transactions on Information and Systems* **E99-D**(3), 297–396.

Yovine, S. (1997). Kronos: A verification tool for real-time systems. *International Journal of Software Tools for Technology Transfer* **1**(1–2), 123–133.

A Proofs of theorems

Theorem 1 (Bisimulation). *Let $Q_{\mathcal{T}}$ be the set of states of the TPN $\mathcal{T}$ and $Q_{\mathcal{A}}$ the set of states of the state class timed automaton $\mathcal{A} = (L, l_0, X, A, E, Inv)$. Let $\mathcal{R} \subset Q_{\mathcal{T}} \times Q_{\mathcal{A}}$ be a relation such that $\forall s = (M, \nu_{\mathcal{T}}) \in Q_{\mathcal{T}}, \forall a = (l, \nu_{\mathcal{A}}) \in Q_{\mathcal{A}}, s\mathcal{R}a \Leftrightarrow M_{\mathcal{T}} = M_{\mathcal{A}}$ if $M_{\mathcal{A}}$ is the marking associated with l and $\forall t \in enabled(M), \exists x \in X, \nu_{\mathcal{T}}(t) = \nu_{\mathcal{A}}(x)$.*

$\mathcal{R}$ is a bisimulation.

Proof. Let us consider a state $s = (M, \nu_{\mathcal{T}}) \in Q_{\mathcal{T}}$, a state $a = (l, \nu_{\mathcal{A}}) \in Q_{\mathcal{A}}$ such that $s\mathcal{R}a$ and a continuous transition $s \xrightarrow{\delta} s'$.

First, for any δ such that $s \xrightarrow{\delta} s'$ is possible, $a \xrightarrow{\delta} a'$ is also possible. Indeed, $s \xrightarrow{\delta} s'$ is equivalent to $\forall t \in enabled(M), \nu_{\mathcal{T}}(t) + \delta \leq \beta(t)$. Since $s\mathcal{R}a$, this can be written as $\forall t \in enabled(M)$, if x is such that $t \in trans(x)$, $\nu_{\mathcal{A}}(x) + \delta \leq \beta(t)$, which is equivalent to x satisfies $Inv(l)$, since $Inv(l) = (\wedge_{x \in X}(x \leq \min_{t' \in trans(x)} \beta(t')))$. Actually we have an equivalence here : $s \xrightarrow{\delta} s'$ is possible is equivalent to $a \xrightarrow{\delta} a'$ is possible.

Since there is no change of marking, the enabled transitions remain the same in s' as in s and so do the associated clocks in a' : for all $t \in enabled(M')$, if x was the associated clock in a it is still in a'. Moreover, if $a \xrightarrow{\delta} a'$, then $\nu'_{\mathcal{A}} = \nu_{\mathcal{A}} + \delta$ and if $s \xrightarrow{\delta} s'$, then $\nu'_{\mathcal{T}} = \nu_{\mathcal{T}} + \delta$. As a consequence, $\forall t \in enabled(M'), \exists x \in X, \nu'_{\mathcal{A}}(x) = \nu'_{\mathcal{T}}(t)$. Hence, $a'\mathcal{R}s'$. The reasoning is the same when starting from $a \xrightarrow{\delta} a'$, we have then a bisimulation for the continuous transitions.

Now we consider a discrete transition $s \xrightarrow{t} s'$. Since $s\mathcal{R}a$, the marking of s and a is the same and so are the enabled transitions. In addition, if t is firable in a, then a' an s' will obviously have the same marking M'.

In order for t to be firable in a, the guard of the transition of the automaton must be satisfied. t is firable in s, so $\nu_{\mathcal{T}}(t) \geq \alpha(t)$. $s\mathcal{R}a$ implies that $\exists x \in X, \nu_{\mathcal{T}}(t) = \nu_{\mathcal{A}}(x)$. So, $\exists x \in X, \nu_{\mathcal{A}}(x) \geq \alpha(t)$. The guard of the transition being, by construction, $\nu_{\mathcal{A}}(x) \geq \alpha(t)$, it is obviously satisfied. As before, we have actually an equivalence here : t is firable in $s \Leftrightarrow t$ is firable in a.

Now, let us consider an enabled transition t' of s'. Two cases can occur :

- t' is not newly enabled in s', which implies that t' was enabled in s . Then, as $s\mathcal{R}a$, $\exists x \in X, \nu_{\mathcal{A}}(x) = \nu_{\mathcal{T}}(t')$. Since there is still at least an enabled transition in $trans(x)$ in s', the clock still exists in a' and since no time has elapsed, that relation is still true in s' : $\nu'_{\mathcal{A}}(x) = \nu'_{\mathcal{T}}(t')$.
- t' is newly enabled in s'. So, by construction, a new clock x has been created in a' and its valuation is null. We have then $\nu_{\mathcal{A}}(x) = \nu_{\mathcal{T}}(t') = 0$.

We have shown that s' and a' have the same marking and $\forall t' \in enabled(M'), \exists x \in X, \nu'_{\mathcal{T}}(t') = \nu'_{\mathcal{A}}(x)$ *i.e.* $s'\mathcal{R}a'$.

It is straightforward with the same reasoning, to show that if we have $a \xrightarrow{t} a'$, then $s \xrightarrow{t} s'$ leads to a state s' such that $s' \mathcal{R} a'$, hence the bisimulation for discrete transitions. □

Theorem 2. *There are no* orthogonal *clocks in the state class timed automaton.*

Proof. Let $x_0, \ldots, x_n$ be the number of clocks of the state class timed automaton. When x_k ($k \in [0, n]$) was created, all the indexes lower than k where used since the new index is chosen as the smallest available one. As a consequence, x_k is not orthogonal to any of $x_0, \ldots, x_{k-1}$.

So far, we have proved that a clock is not orthogonal to any of the clocks of smaller index. As the orthogonality relation is symmetric, it comes that there is no orthogonal clocks in the automaton. □

Theorem 3. *There are no location l of the state class timed automaton in which two clocks x, y used in l ($l \in$ TPN-useful$(x) \cap$ TPN-useful(y)) are equal*

Proof. This will be shown inductively on the automaton. First, the initial state has only one clock, so there is nothing to prove there.

Then, when we generate a new state. If we have two equal clocks x_i and x_j, then there is some previous state when x_i, for instance, was created. In that state we had $\nu(x_i) = 0$ and the time being the same for all clocks, $\nu(x_j) = 0$. But a new clock will not be created if there already exists a clock equal to 0. As a consequence, this cannot happen. □