Model checking of Time Petri Nets using the State Class Timed Automaton*

Didier Lime and Olivier (H.) Roux

Abstract. In this paper, we propose a method for building the state class graph of a bounded time Petri net (TPN) as a timed automaton (TA), which we call the state class timed automaton. We consider bounded TPN, whose underlying net is not necessarily bounded. We prove that our translation preserves the behavioural semantics of the TPN (the initial TPN and the obtained TA are proved timed-bisimilar). It allows us to check real-time properties on TPN by using the state class TA. This can be done efficiently thanks to a reduction of the number of clocks. We have implemented the method, and give some experimental results illustrating the efficiency of the translation algorithm in terms of number of clocks. Using the state class TA, we also give a framework for expressing and efficiently verifying TCTL properties on the initial TPN.

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, timed 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.

The two main timed extensions of Petri Nets are Time Petri Nets (TPN) (Merlin 1974) and Timed Petri Nets (Ramchandani 1974). While a transition can be fired within a given interval for TPN, in Timed Petri Nets, temporisation represents the minimal duration between the firing of transitions (or the exact duration with an "as soon as possible" firing rule). There are also numerous way of representing time. It may be relative to places, transitions, arcs or tokens. The classes of Timed Petri Nets (time relative to place, transition...) are included in the corresponding classes of Time Petri Nets (Pezze and Toung 1999). TPN are mainly divided in P-TPN (Khansa *et al.* 1996), A-TPN (de Frutos Escrig *et al.* 2000, Abdulla and Nylén 2001) and T-TPN (Berthomieu and Diaz 1991) where a time interval is relative respectively to places, arcs and transitions. Finally, Time Stream Petri Nets (Diaz and Senac 1994) were introduced to model multimedia applications and are related to A-TPN.

Recently, some papers consider A-TPN with a "lazy" (non-urgent) semantics (de Frutos Escrig *et al.* 2000, Abdulla and Nylén 2001). 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. The advantage is that boundedness and coverability are decidable. However, urgency, which is required for the modeling of critical real-time systems, is lost.

For the modeling of urgency together with concurrency and dense-time constraints, transition-time Petri nets (TPN) with strong semantics as defined in (Merlin 1974, Berthomieu and Diaz 1991) are more adapted and widely used (Vicario 2001, Berthomieu and Vernadat 2003, Penczek and Polrola 2004). In this model, time constraints are expressed as time intervals on the transitions of the net. This is the model that we consider in this paper.

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.

Unlike TA, the number of discrete states of the TPN (markings) is not necessarily bounded. 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 state-space 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 the names of the transitions. 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. Berthomieu and Vernadat (Berthomieu and Vernadat 2003) propose an alternative construction of the graph of atomic classes of Yoneda applicable to a larger class of nets.

Our approach for the verification of TPN consists in translating a TPN into a 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), Sifakis and 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 reset. 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), the 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 so that forbidden markings become unreachable. 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 clocks makes it difficult to efficiently analyze the obtained TA.

Our contribution

Our approach consists in 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: Sufficient conditions of boundedness that can be tested before the computation and on-the-fly necessary conditions of unboundedness allowing to stop the computation if we assume that the net is unbounded. The initial TPN and the obtained TA are timed-bisimilar. Obtaining a TA instead of a graph is a great improvement (e.g. TCTL model-checking can be applied directly) for a low additional cost. Moreover, the number of clocks of the obtained TA is low (in practice, often much lower than the number of transitions of the initial TPN). We also show how to express and efficiently verify TCTL properties on a TPN by using its associated state class TA. Since the two are bisimilar it also allows us to say that TCTL model-checking is decidable on bounded TPN, which has never been proved to our knowledge. We have developed a tool for building this automaton and transcribe it in Uppaal and Kronos input formats.

Outline of the paper

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. In section 4, we give a brief description of the tool implementing the algorithm and some experimental results. Finally, we provide a framework for checking TCTL properties on the TPN with our automaton, in section 5.

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 $t \in \mathit{enabled}(M)$.

A transition t_k is said to be *newly* enabled by the firing of the firable transition t_i from the marking M, and we denote it by $\uparrow \mathit{enabled}(t_k, M, t_i)$, if the transition t_k 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,

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

By extension, we will denote by $\uparrow \mathit{enabled}(M, t_i)$ the set of the transitions which are newly enabled by the firing of 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, $\forall t \in \mathit{enabled}(M), \overline{0}(t) = 0$.

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:*

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

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

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

Let us consider the TPN in Figure 1. Let us assume that t_1 is firable. In Figure 1a, t_2 is enabled by the marking M and by the marking $M' = M - {}^\bullet t_1 + t_1{}^\bullet$ but not by $M - {}^\bullet t_1$. Transitions t_1 and t_2 are *newly enabled* after the firing of t_1.

Now, in Figure 1b, t_2 is enabled by the marking M, by the marking $M' = M - {}^\bullet t_1 + t_1{}^\bullet$ and by $M - {}^\bullet t_1$. t_1 is *newly enabled* by the firing of t_1 but not t_2: t_2 *remains* enabled.

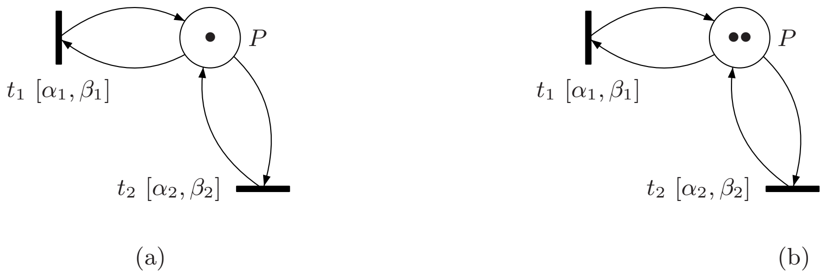

P
$t_1\ [\alpha_1, \beta_1]$
$t_2\ [\alpha_2, \beta_2]$
(a)
P
$t_1\ [\alpha_1, \beta_1]$
$t_2\ [\alpha_2, \beta_2]$
(b)

Fig. 1. Example of newly enabled transitions

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, with some restrictions, (see 3.2) a finite 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 inequations called the* firing domain.

The inequations in D are of two types (Berthomieu and Diaz 1991):

$$\begin{cases} a_i \leq \theta_i \leq b_i, \textit{ with } a_i, b_i \in \mathbb{Q}^+ \textit{ } (\forall i \textit{ s.t. } t_i \textit{ is enabled}), \\ \theta_j - \theta_k \leq c_{jk}, \textit{ with } c_{jk} \in \mathbb{Q}^+ \textit{ } (\forall j, k \textit{ s.t. } j \neq k \textit{ and } t_j, t_k \in \textit{enabled}(M)) \end{cases}$$

θ_i is the firing time of the enabled transition t_i relatively to the time when the marking of the class occured.

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 inequations define constraints on the firing times θ of the transitions enabled by this marking.

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

Given a set of inequations 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 inequations 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 inequations and comparing the solutions.

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

$$\begin{cases} a_i^* \leq \theta_i \leq b_i^*, \\ \theta_j - \theta_k \leq c_{jk}^* \end{cases}$$

where $\begin{cases} a_i^* = Inf\{\theta_i\} \\ b_i^* = Sup\{\theta_i\} \\ c_{jk}^* = Sup\{\theta_j - \theta_k\} \end{cases}$

The canonical form may be computed by the Floyd-Warshall algorithm, which determines the shortest paths in a constraint graph. Its complexity is $O(n^3)$ in time and $O(n^2)$ in space, n being the number of variables in the set of inequations (Berthomieu 2001)

Computing the graph.

Definition 5 (firability). *Let $C = (M, D)$ be a state class. A transition $t_i \in enabled(M)$ is said to be* firable *from C iff in D, $\forall t_j \in enabled(M) - \{t_i\}, c^*_{ji} \geq 0$.*

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 $\theta_f \leq \theta_j$,
  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. Indeed, the state class graph does not accept the timed language of the TPN. In Figure 2, we can see that if t_1 fires at time 0 then we have the forced sequence t_2 then t_3 because $\alpha(t_3) > \beta(t_2)$, 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.

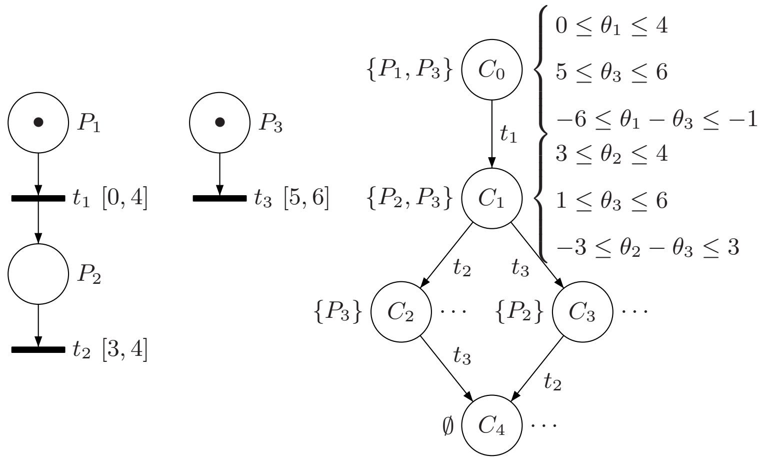


Fig. 2. A TPN and its state class graph

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 $v, v' \in V$, $\sim \in \{<, \le, =, \ge, >\}$ 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 reset R and the renaming function 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 function $\rho : X \to X$, 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 have mentioned before, timing information is lost when computing the state class graph. Consequently, in order to verify complex timed 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 since the complexity of verification algorithms is exponential in this number.

In order to achieve the computation of the state-space of the net as a timed automaton, 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 valuations are 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 then be 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. The 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 all 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 all 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:

$$(M, D, \chi, trans) \xrightarrow{t}^{ext} (M', D', \chi', trans') \text{ iff}$$

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

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 $\Delta(\mathcal{T}) = (L, l_0, X, A, E, Inv)$ is 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,*

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

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

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

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 from the parent location of C' to the location C. In the latter case, we also need to check if C' is included in C, according to definition 11:

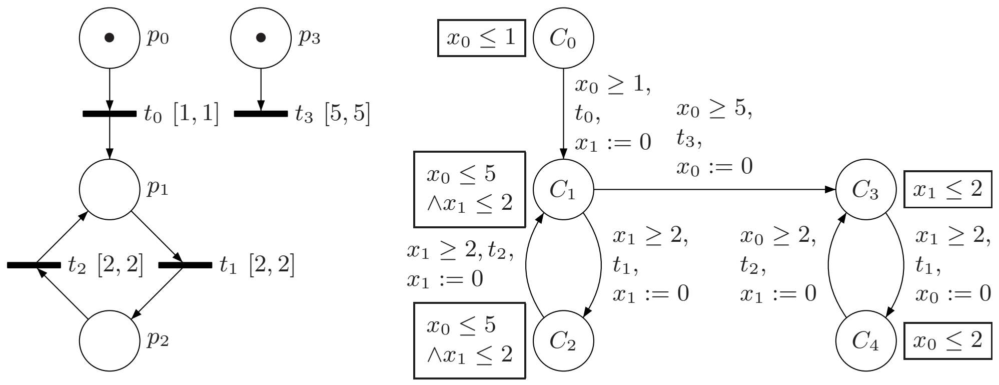
Definition 11 (Inclusion between two extended state classes). *An extended state class* $C' = (M', D', \chi', trans')$ *is* included *in an extended state class* $C = (M, D, \chi, trans)$ *iff* C *and* C' *are clock-similar and* $[\![D']\!] \subset [\![D]\!]$. *This is denoted by* $C' \subset C$.

If $C' \subset C$, then we do not need to compute further on that branch. If $C' \not\subset C$, the domain D of C becomes $D \cup D'$, D' being the firing domain of C', and we compute the successors 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.

When the union $D \cup D'$ is made, this does not introduce any unwanted behaviors since invariants and guards are given syntactically from the net itself. Clock-similarity ensures that merged locations are coherent with respect to this syntactical definition.

This is illustrated by the net in Figure 3a and its state class timed automaton given in Figure 3b. When making the loop $C_2 \overset{t_2}{\to} C_1$ we do add a new potential behavior from location C_1, namely $C_1 \overset{t_3}{\to} C_3$. However, in the final automaton, starting from the initial location, the corresponding guard will not be true until we have fired t_1 and t_2.



p_0
p_3
t_0 $[1,1]$
t_3 $[5,5]$
p_1
t_2 $[2,2]$
t_1 $[2,2]$
p_2
$x_0 \leq 1$
C_0
$x_0 \geq 1,$ $t_0,$ $x_1 := 0$
$x_0 \geq 5,$ $t_3,$ $x_0 := 0$
$x_0 \leq 5$ $\wedge x_1 \leq 2$
C_1
C_3
$x_1 \leq 2$
$x_1 \geq 2, t_2,$ $x_1 := 0$
$x_1 \geq 2,$ $t_1,$ $x_1 := 0$
$x_0 \geq 2,$ $t_2,$ $x_1 := 0$
$x_1 \geq 2,$ $t_1,$ $x_0 := 0$
$x_0 \leq 5$ $\wedge x_1 \leq 2$
C_2
C_4
$x_0 \leq 2$

Fig. 3. Looping with domain union

Example. Another example of a TPN and the corresponding state class TA is shown in Figure 4. 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 5.

Bisimulation. 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 $\mathcal{T}$. First we give the definition of bisimulation relations.

Definition 12 (Bisimulation). *Let $\mathcal{S}_1 = (S_1, s_1^0, \rightarrow_1)$ and $\mathcal{S}_2 = (S_2, s_2^0, \rightarrow_2)$ be two transition systems on the same alphabet A. Let $\mathcal{R} \in S_1 \times S_2$ be a binary relation on the sets of states of the two systems.*

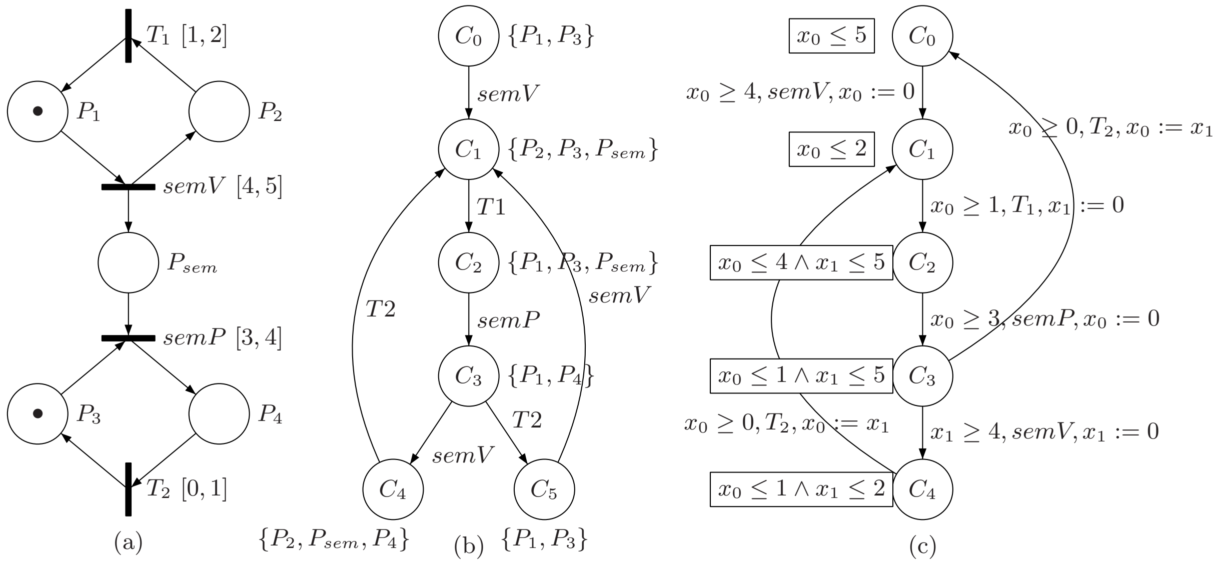
$\mathcal{R}$ is a bisimulation iff $\forall (s_1, s_2) \in S_1 \times S_2$ s.t. $s_1 \mathcal{R} s_2, \forall a \in A$,

$$\begin{cases} \exists s_1' \in S_1 \textit{ s.t. } s_1 \xrightarrow{a}_1 s_1' \Rightarrow \exists s_2' \in S_2 \textit{ s.t. } s_2 \xrightarrow{a}_2 s_2' \textit{ and } s_1' \mathcal{R} s_2', \\ \exists s_2' \in S_2 \textit{ s.t. } s_2 \xrightarrow{a}_2 s_2' \Rightarrow \exists s_1' \in S_1 \textit{ s.t. } s_1 \xrightarrow{a}_1 s_1' \textit{ and } s_1' \mathcal{R} s_2' \end{cases}$$

Two transition systems $\mathcal{S}_1 = (S_1, s_1^0, \rightarrow_1)$ and $\mathcal{S}_2 = (S_2, s_2^0, \rightarrow_2)$ are said *bisimilar* if there exists a bisimulation $\mathcal{R}$ in $S_1 \times S_2$ such that $s_1^0 \mathcal{R} s_2^0$.

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 the 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.



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

Fig. 4. Two cyclic tasks synchronized *via* a semaphore: TPN model (a), its state class graph (b) and its state class timed automaton (c)

The proof for theorem 1 is given in Appendix A.

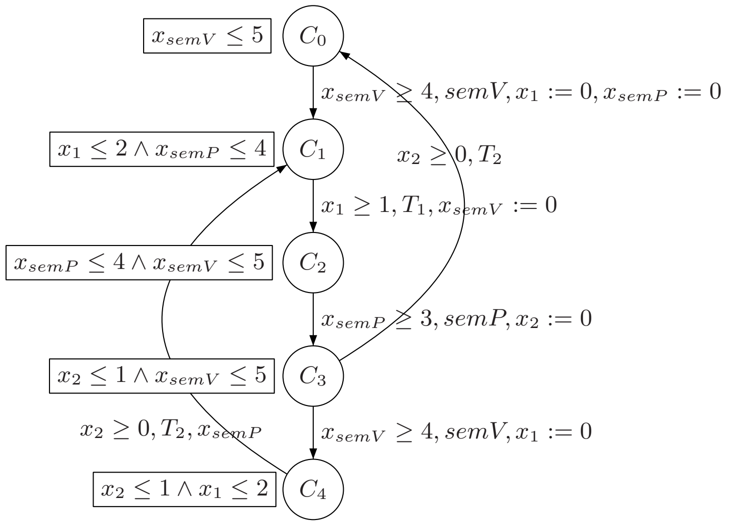
Finiteness of the state class timed automaton. Berthomieu and Diaz have shown that a TPN has a finite number of state classes if and only if it is bounded, provided that the α and β functions are rational (Berthomieu and Diaz 1991). The proof, that relies only on markings and firing domains, is fully 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 and the earliest and latest firing times of transitions are rationals.*

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

Theorem 3. *A TPN is bounded if the underlying Petri net is bounded.*

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 assume that the state class TA will not be finite. Again, theorem 4 is a straight extension of that of (Berthomieu and Diaz 1991).



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

Fig. 5. TA with one clock per transition of the TPN

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 $t_1, \ldots, t_k$ enabled at the same time τ_0. For all $i \leq k$, we have $\nu(t_i)(\tau_0) = 0$. 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 in choosing the first unused clock *i.e.* the one with the smallest index.

We will now give several results concerning the number of clocks of the state class timed automaton. First, the following theorem comes straightforwardly:

Theorem 5 (Upper bound for the number of clocks). *Let $\mathcal{T}$ be a time Petri net and let $\Delta(\mathcal{T})$ be its state class timed automaton. $\Delta(\mathcal{T})$ has a number of clocks lower or equal to the maximum number of transitions enabled by the reachable markings of $\mathcal{T}$.*

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 $Inv(l)$.

Let l be a location of a TA, $clk(l)$ denotes the set of clocks that appear either in the invariant of l or in the guard of one of the outgoing edges of l: $clk(s) = \{x \in X | x \in op(Inv(l)) \vee \exists (l, \delta, \alpha, R, l') \in E, x \in op(\delta)\}$.

Let us now recall the definition of activity of Daws and Yovine:

Definition 13 (Active clocks). *In a location $l \in L$, the set of active clocks $act(l)$ is given by the least fixed point of the equation:*

$$act(l) = clk(l) \bigcup_{(l,\delta,\alpha,R,\rho,l')} \rho^{-1}(act(l'))$$

According to that definition, the automaton has no *inactive* clocks since in each location, all the used clocks appear in the invariant, or in a guard in a "reachable" location. 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 ($x, y \in clk(l)$) are equal*

This is sufficient to say that applying the method of (Daws and Yovine 1996) on the state class timed automaton will not reduce its number of clocks.

Let us now define the notion of orthogonality of two clocks.

Definition 14 (Orthogonality of two clocks). *x and y are two orthogonal clocks, denoted by $x \perp y$ iff $\nexists l \in L$ s.t. $x \in clk(l) \wedge y \in clk(l)$*

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

The proofs for theorems 6 and 7 are given in Appendix A.

4 Experimental results

We have implemented the building of the state class timed automaton in a tool named Romeo (Lime and Roux 2004) that consists of a graphical user 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 number of clocks, of (Bornot *et al.* 1998, Sifakis and Yovine 1996, Sava 2001) and Romeo. Contrary to our method, no tool implementing the first two methods are available to our knowledge. This is not a problem since, for these methods, the number of clocks is given by the theory. For the method described in (Sava 2001), Sava proposes no implementation and a few theoretical points are unclear, so we used the solutions and the implementation of (Gardey *et al.* 2005).

The examples we used for testing include modeling 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.

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 first two colums (NA meaning “not available”); 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).

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

	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

of the algorithm of Sava, as proposed in (Gardey *et al.* 2005). 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.

Figure 6 shows a simple example, which we will use to illustrate the advantages of our method. Note that the underlying Petri net is unbounded so (Sifakis and Yovine 1996) and (Bornot *et al.* 1998) cannot be applied to it. The result of (Sava 2001) is given in Figure 7a. Figure 7b shows the result of (Sava 2001) plus (Daws and Yovine 1996). Finally, we give our corresponding extended state class graph and state class timed automaton in Figure 8a and Figure 8b respectively. We can see that our method yields a TA with the same underlying structure as the with the method of Sava. However, the number of clocks is lower, even more than what is obtained by applying the reduction algorithms of Daws and Yovine.

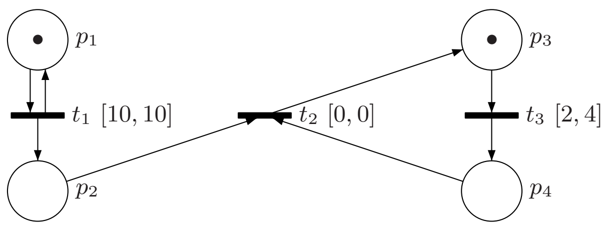


Fig. 6. A periodic producer and a cyclic consumer

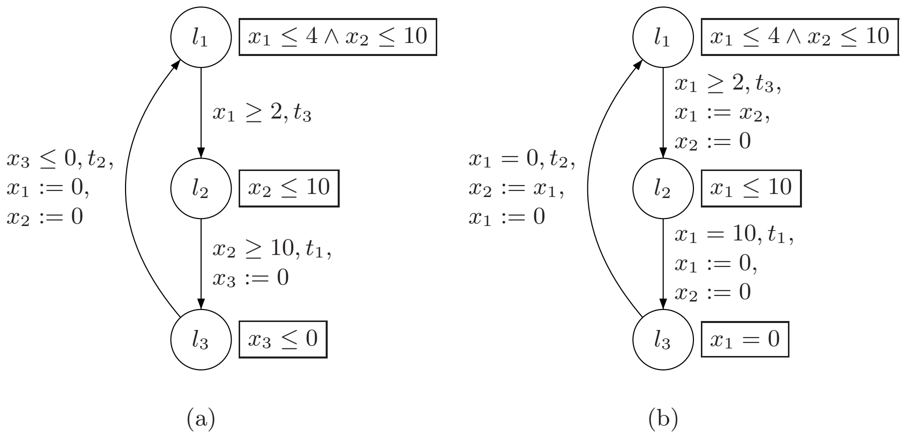


Fig. 7. Automaton produced by [Sav01] (a) plus [DY96] (b) for the net of Figure 6

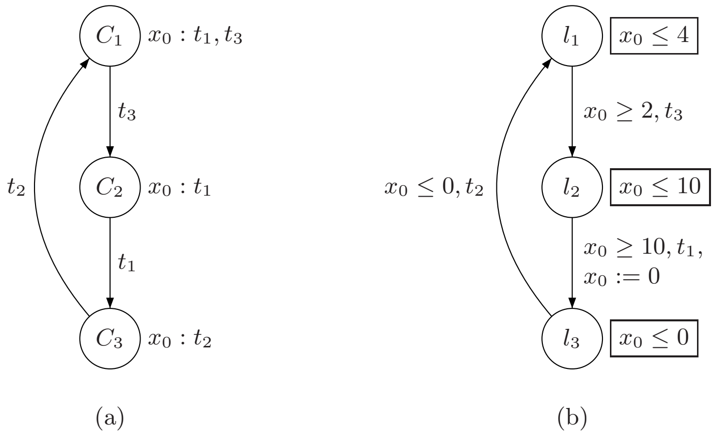


Fig. 8. Extended state class graph (a) and automaton (b) produced by Romeo for the net of Figure 6

	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	1169	4089	14418	46079
Example 8	1294	4358	13557	41249
Example 9	58	135	252	548
Example 10	39	68	126	222
Example 11	50	123	138	330
Example 12	131	351	4256	8977
Example 13	355	878	24401	50876
Example 14	1048	3002	22016	60967
Example 15	1088	5245	1098	5260
Example 16	76	199	200	353
Example 17	735	2263	1403	3508
Example 18	1871	6322	2831	8386
Example 19	11490	50168	14086	56929
Example 20	14	20	16	22

Table 2. Number of locations and transitions

5 TCTL Model-Checking for TPN

Thanks to the state class timed automaton we can do efficient model-checking on a TPN with Kronos or Uppaal, for instance. The properties expressed on the TPN are translated an verified on its associated state class TA. This is possible because of the bisimulation between the TPN and its state class timed automaton and it is efficient thanks to the low number of clocks of the latter.

Since TCTL model-checking is decidable on TA, the bisimulation also allows us to say that TCTL model-checking is decidable on bounded TPN.

We first give a definition of TCTL for TPN. The only difference with the versions of (Alur *et al.* 1993, Henzinger *et al.* 1994) is that the atomic propositions usually associated to states are properties of markings.

5.1 TCTL for TPN

A run in a TPN $\mathcal{T}$ is a path in its state-space along a sequence of firable transitions and starting at any state q. The set of runs of $\mathcal{T}$ is denoted by $[\![\mathcal{T}]\!]$.

Definition 15 (TCTL for TPN). *Let* $\mathcal{T} = (P, T, {}^{\bullet}(.), (.)^{\bullet}, M_0, (\alpha, \beta))$ *be a TPN with* $P = \{p_1, \cdots, p_m\}$ *and* $T = \{t_1, \cdots, t_n\}$*. The temporal logics TPN-TCTL is inductively defined by:*

$$\textit{TPN-TCTL} ::= \mathbf{M}(p_i) \bowtie V \mid t_i - t_j \leq d \mid M_i - M_j \leq d \mid \mathbf{false} \mid \neg\varphi \mid \varphi \rightarrow \psi \mid \varphi\, \exists\mathcal{U}_{\bowtie c}\, \psi \mid \varphi\, \forall\mathcal{U}_{\bowtie c}\, \psi$$

where $\mathbf{M}$ *and* **false** *are keywords,* $t_i, t_j \in T$, $M_i, M_j \in \mathbb{N}^p$, $\varphi, \psi \in$ *TPN-TCTL,* $p_i \in P$, $c, d, V \in \mathbb{N}$ *and* $\bowtie \in \{<, \leq, =, >, \geq\}$. □

Intuitively, $\mathbf{M}(p_i) \bowtie V$ means that the current marking of the place p_i is in relation $\bowtie$ with V. $t_i - t_j \leq d$ means that from the current state, for any firing of t_j there is less than d time units since the last firing of t_i. Similarly $M_i - M_j \leq d$ means that from the current state, when entering marking M_j less than d time units has elapsed since we have last entered in marking M_i. The meaning of the other operators is the usual one. We use the usual shorthands **true** $= \neg$**false**, $\exists\Diamond_{\bowtie c}\phi =$ **true** $\exists\mathcal{U}_{\bowtie c}\,\phi$ and $\forall\Box_{\bowtie c} = \neg\exists\Diamond_{\bowtie c}\neg\phi$.

The semantics of TPN-TCTL is defined on TTS. Let $\mathcal{T} = (P, T, {}^\bullet(.), (.)^\bullet, M_0, \alpha, \beta)$ be a TPN and $S_\mathcal{T} = (Q, q_0, \rightarrow)$ the semantics of $\mathcal{T}$. The truth value of a formula φ of TPN-TCTL for a state (M, ν) is given on Table 3.

$$
\begin{array}{lll}
(M,\nu) \models M(p_i) \bowtie V & \text{iff} & M(p_i) \bowtie V \\
(M,\nu) \models t_i - t_j \leq d & \text{iff} & \forall \sigma = (s_0,\nu_0) \xrightarrow{a_1,\,d_1} (s_1,\nu_1) \cdots \xrightarrow{a_n,\,d_n} (s_n,\nu_n) \in [\![\mathcal{T}]\!] \\
& & \quad \text{s.t. } (s_0,\nu_0) = (M,\nu), \\
& & \quad \forall k,l \in [1..n], \text{ s.t. } \begin{cases} a_k = t_i, a_l = t_j, k < l, \\ \forall m, k < m < l, a_m \neq t_i, \end{cases} \\
& & \quad \sum_{n=k+1}^{n=l} d_n \leq d \\
(M,\nu) \models M_i - M_j \leq d & \text{iff} & \forall \sigma = (s_0,\nu_0) \xrightarrow{a_1,\,d_1} (s_1,\nu_1) \cdots \xrightarrow{a_n,\,d_n} (s_n,\nu_n) \in [\![\mathcal{T}]\!] \\
& & \quad \text{s.t. } (s_0,\nu_0) = (M,\nu), \\
& & \quad \forall k,l \in [1..n] \text{ s.t. } \begin{cases} s_k = M_i, s_l = M_j, k < l, \\ \forall m, k < m < l, s_m \neq M_i, \end{cases} \\
& & \quad \sum_{n=k+1}^{n=l} d_n \leq d \\
(M,\nu) \not\models \mathbf{false} & & \\
(M,\nu) \models \neg\varphi & \text{iff} & (M,\nu) \not\models \varphi \\
(M,\nu) \models \varphi \rightarrow \psi & \text{iff} & (M,\nu) \models \varphi \text{ implies } (M,\nu) \models \psi \\
(M,\nu) \models \varphi\,\exists\mathcal{U}_{\bowtie c}\,\psi & \text{iff} & \exists \sigma = (s_0,\nu_0) \xrightarrow{a_1,\,d_1} (s_1,\nu_1) \cdots \xrightarrow{a_n,\,d_n} (s_n,\nu_n) \in [\![\mathcal{T}]\!] \\
& & \quad \text{s.t. } \begin{cases} (s_0,\nu_0) = (M,\nu) \\ \forall i \in [1..n], \forall d \in [0,d_i), (s_i, \nu_i + d) \models \varphi \ \textit{and} \\ \left(\sum_{i=1}^{n} d_i\right) \bowtie c \ \textit{and} \ (s_n, v_n) \models \psi \end{cases} \\
(M,\nu) \models \varphi\,\forall\mathcal{U}_{\bowtie c}\,\psi & \text{iff} & \forall \sigma = (s_0,\nu_0) \xrightarrow{a_1,\,d_1} (s_1,\nu_1) \cdots \xrightarrow{a_n,\,d_n} (s_n,\nu_n) \in [\![\mathcal{T}]\!] \\
& & \quad \text{s.t. } (s_0,\nu_0) = (M,\nu), \\
& & \quad \forall i \in [1..n], \forall d \in [0,d_i), (s_i, \nu_i + d) \models \varphi \ \textit{and} \ \left(\sum_{i=1}^{n} d_i\right) \bowtie c \ \textit{and} \ (s_n, v_n) \models \psi
\end{array}
$$

Table 3. Semantics of TPN-TCTL

The TPN $\mathcal{T}$ satisfies the formula φ of TPN-TCTL, which is denoted, $\mathcal{T} \models \varphi$ iff the initial state of $S_\mathcal{T}$ satisfies φ *i.e.* $(M_0, \nu_0) \models \varphi$.

5.2 Model-Checking for TPN-TCTL

For all properties, except the second, $t_i - t_j \leq d$, and third, $M_i - M_j \leq d$, the formula on the TPN is straightforwardly translated on the state class TA:

Let (M, ν) be a state of $S_{\mathcal{T}}$ and $\Delta((M, \nu))$ the equivalent state of $S_{\Delta(\mathcal{T})}$.

Let φ be a formula to be model-checked on a TPN $\mathcal{T}$. Our method consists of using the state class timed automata $\Delta(\mathcal{T})$ defined in section 3.

Since $\mathcal{T}$ and $\Delta(\mathcal{T})$ are timed bisimilar, $\forall \varphi \in$ TPN-TCTL$-\{t_i - t_j \leq d, M_i - M_j \leq d\}, (M, \nu) \models \varphi \Leftrightarrow \Delta((M, \nu)) \models \varphi$.

Practically speaking, with Uppaal for instance, the marking is an array of integers M. The automaton updates M so that $M[i]$ is the number of tokens in the place p_i of the net. For instance, we might want to check $\mathcal{T} \models \forall \Box_{\leq 3}(\mathbf{M}(p_1) \geq 1 \wedge \mathbf{M}(p_2) \leq 2)$. It means that all the states reached within the next 3 time units will have a marking such that p_1 has more than one token and p_2 less than 2. This is equivalent to checking $\forall \Box_{\leq 3}(M[1] \geq 1 \wedge M[2] \leq 2)$ on the state class timed automaton.

For the $t_i - t_j \leq d$ and $M_i - M_j \leq d$ properties, we cannot use directly the clocks of the state class TA to keep track of the firing times of transitions, since their values may be undefined when transitions are not enabled, and also for the sake of generality. Indeed, while the automaton is quite small, it is possible to understand the meaning of each of its clocks, but as we model real systems with much more clocks it becomes tedious and a potential source of errors.

Instead, we propose a method that allows us to never look at the automaton. The idea is to synchronize the state class automaton with observer automata, on the firing of transitions, or the occurrence of markings, which gives us access to additional clocks specific to the property.

The synchronization between the state class TA and the observers uses the classical composition notion based on a *synchronization function* à la Arnold-Nivat (Arnold 1994). Let $H_1, \ldots, H_n$ be n timed automata with $H_i = (N_i, l_{i,0}, C_i, A, E_i, Inv_i)$. A *synchronization function* f is a partial function from $(A \cup \{\bullet\})^n \hookrightarrow A$ where $\bullet$ is a special symbol used when an automaton is not involved in a step of the global system. Note that f is a synchronization function with renaming. We denote by $(H_1|\ldots|H_n)_f$ the parallel composition of the H_i's w.r.t. f. The configurations of $(H_1|\ldots|H_n)_f$ are pairs $(\overline{l}, v)$ with $\overline{l} = (l_1, \ldots, l_n) \in N_1 \times \ldots \times N_n$ and $v = v_1 \cdots v_n$ with $v_i \in (\mathbb{R}_{\geq 0})^{C_i}$ being the restriction of v to C_i (we assume that all sets C_i of clocks are disjoint). Then the semantics of a synchronized product ot timed automata is also a timed transition system formalized by the following definition:

Definition 16 (Semantics of a Product of Timed Automata). *Let $H_1, \ldots, H_n$ be n timed automata with $H_i = (N_i, l_{i,0}, C_i, A, E_i, Inv_i)$, and f a (partial) synchronization function $(A \cup \{\bullet\})^n \hookrightarrow A$. The semantics of $(H_1|\ldots|H_n)_f$ is a timed transition system $S = (Q, q_0, \rightarrow)$ with $Q = N_1 \times \ldots \times N_n \times (\mathbb{R}_{\geq 0})^C$, q_0 is the initial state $((l_{1,0}, \ldots, l_{n,0}), \overline{0})$ and $\rightarrow$ is defined by:*

– $(\bar{l}, v) \xrightarrow{b} (\bar{l'}, v')$ *iff there exists* $(a_1, \ldots, a_n) \in (A \cup \{\bullet\})^n$ *s.t.* $f(a_1, \ldots, a_n) = b$ *and for any* i *we have:*

. *If* $a_i = \bullet$, *then* $l'_i = l_i$ *and* $v'_i = v_i$,

. *If* $a_i \in A$, *then* $(l_i, v_i) \xrightarrow{a_i} (l'_i, v'_i)$.

– $(\bar{l}, v) \xrightarrow{\epsilon(t)} (\bar{l}, v')$ *iff* $\forall\, i \in [1..n]$, *we have* $(l_i, v_i) \xrightarrow{\epsilon(t)} (l_i, v'_i)$ □

We could equivalently define the product of n timed automata syntactically, building a new timed automaton (Larsen *et al.* 1995) from the n initial ones. In the sequel we consider a product $(H_1|\ldots|H_n)_f$ to be a timed automaton the semantics of which is timed bisimilar to the semantics of the product we have given in definition 16.

Properties on transition firing times. Practically speaking, if we want to refer to transition t in our property we will synchronize our automaton with the one in Figure 9.

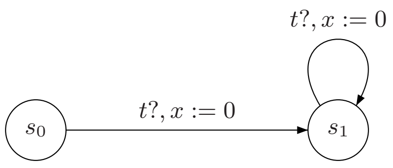


Fig. 9. Observer automaton for transition events

The observer (Figure 9) has two locations, two transitions and one clock. The transitions are synchronized with the firing of transition t and the clock x is reset when it is taken. If the observer is in location s' then the transition t has been fired at least once.

The property $t_i - t_j \leq d$ of the TPN $\mathcal{T}$ is then verified by adding two observer automata O_i and O_j providing respectively clocks x_i and x_j and then verifying the property $s_1^i \wedge s_1^j \Rightarrow x_i - x_j \leq d$ on the product $(\Delta(\mathcal{T})|O_i|O_j)_f$, with the synchronization function f with 3 parameters defined by:

– $f(t_i, t_i?, \bullet) = t_i$

– $f(t_j, \bullet, t_j?) = t_j$

Properties on marking occurrence times. Similarly, if we want to refer to the occurrence of a marking M, we will synchronize with the automaton in Figure 10. Indeed, while firing of transitions and occurrence of markings are strongly linked, it is far more convenient to refer to the marking directly, than to the transitions whose firings have given that marking.

In order to synchronize on changes of markings, we also need a supervisor S (Figure 11). The latter synchronizes with all the transitions of the TPN, thus detecting any potential change of marking. It is then in a *committed* location which must be left before letting the time pass again. As a consequence, the exit transition is taken, with a synchronization vector on all the marking observers (Figure 10), actually forcing them to evaluate whether the current marking is matching the one they look for or not. Those marking observers have the same structure as the transition observers.

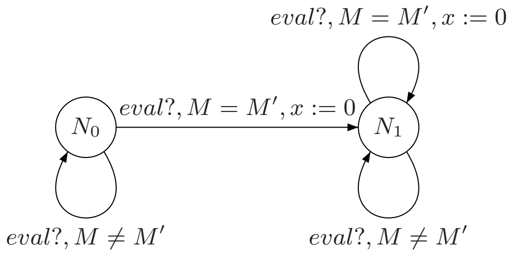


Fig. 10. Observer automaton for marking events

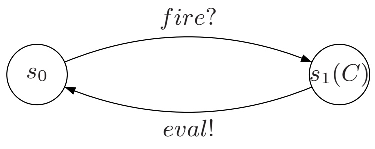


Fig. 11. Supervisor automaton for marking observers

However, synchronization vectors are not allowed in Uppaal, for instance, so we may need to use a counter to synchronize with all the marking observers successively. As a consequence, strictly speaking, synchronization does not occur simultaneously for all the marking observers, but with the elapsing of zero time unit between each of them. That is why TCTL formulas involving markings are required to be evaluated in states when the supervisor is in its initial location, which means that all synchronizations have occurred.

The property $M_i - M_j \leq d$ of the TPN $\mathcal{T}$ is then verified by adding two observer automata O_i and O_j providing respectively the clocks x_i and x_j and then verifying the property $s_0 \wedge N_1^i \wedge N_1^j \Rightarrow$

$x_i - x_j \leq d$ on the product $(\Delta(\mathcal{T})|S|O_i|O_j)_f$, with the synchronization function f with the synchronization function f with 4 parameters defined by:

- $\forall k \in [1..n], f(t_k, fire?, \bullet, \bullet) = t_k$
- $f(\bullet, eval!, eval?, eval?) = eval!$

Property observers. For properties involving up to three clocks it may be interesting to use one observer for each $t_i - t_j \leq d$ element of the property instead of one observer per transition (or marking) involved. This observer has one location l_0, one clock x and two loops on l_0 reseting x. One of the loop synchronizes on t_i and the other on t_j. This has the advantage of using only one clock for this property element instead of two. However, if we want to check a property like $t_i - t_j \leq d_1 \vee t_j - t_k \leq d_2 \vee t_i - t_k \leq d_3 \wedge t_k - t_i \leq d_4$ then this approach would require four clocks and the “one observer per transition” approach only three.

5.3 Example

We consider the TPN $\mathcal{T}$ in Figure 4, section 3. Let us suppose that we want to check if the maximum time during which the *consumer* task is blocked on the semaphore is less than 3 time units. That is the maximum time between the firings of T_2 and T_{semP}. The property may be expressed as ”$T_2 - T_{semP} \leq 3$” and thus we need the two observers O_2 and O_{semP} in Figure 12, which give us two new clocks x_2 and x_{semP}. The TCTL property on the product $(\Delta(\mathcal{T})|O_2|O_{semP})_f$ to be verified is then $x_2 - x_{semP} \leq 3$.

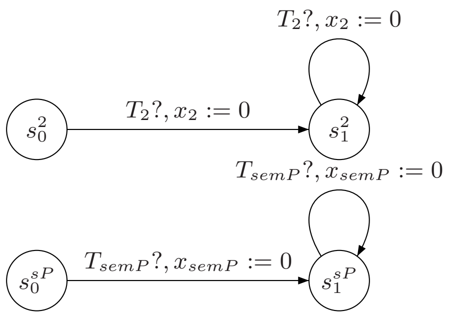


Fig. 12. Observers O_2 and O_{semP} for transitions T_2 and T_{semP}

We now propose some figures concerning the verification of a TCTL property on the state class timed automaton with Kronos using a forward analysis. The property we verified has the form $prop_A \Rightarrow \forall\Diamond_{\leq 30} prop_B$ where $prop_A$ and $prop_B$ are propositions associated to locations and which can be markings. Table 4 gives the results. The first column gives the number of locations of the state class timed TA. Column 2 gives its number of transitions and column 3 its number of clocks. We give the computation time for obtaining the state class timed automaton in the fourth column and the time for verifying the property with KRONOS in the last column. These tests have been performed on a computer with an Intel Pentium II 400MHz processor and 320 Mo of RAM.

	Locations (TA)	Transitions (TA)	Clocks	ROMEO time	KRONOS time
Example 4	23	24	2	0.08	$\leq$0.01
Example 5	48	69	2	0.11	0.04
Example 9	58	135	3	0.17	0.03
Example 12	131	351	4	0.76	0.96
Example 13	355	878	4	2.88	5.79
Example 14	1048	3002	4	6.72	12.05
Example 19	11490	50168	3	86.6	47.7

Table 4. Verifcation of a TCTL property

Of course, little can be generalized from these figures since the results are highly dependent on the automaton and on the property that we check on it. However, they give an idea of the time required for the verification.

6 Conclusion

In this paper, 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 timed 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 generally 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. We have also shown a method for verifying TCTL properties on the TPN, 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, *Scheduling*-TPN (Roux and Déplanche 2002), allowing the modeling of preemptive scheduling of real-time processes. 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 behavior of the whole system.

Acknowledgement. The authors want to thank Raymond Devillers for his useful comments which have led to this updated version.

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.

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

Arnold, A. (1994). *Finite Transition System*. Prentice Hall.

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 F. Vernadat (2003). State class constructions for branching analysis of time Petri nets. In: *9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'2003)*. Springer–Verlag. pp. 442–457.

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.

Diaz, M. and P. Senac (1994). Time stream Petri nets: a model for timed multimedia information. *Lecture Notes in Computer Science* **815**, 219–238.

Gardey, G., O.H. Roux and O.F. Roux (2005). State space computation and analysis of time Petri nets. *Theory and Practice of Logic Programming (TPLP). Special Issue on Specification Analysis and Verification of Reactive Systems.* to appear.

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.

Khansa, W., J.-P. Denat and S. Collart-Dutilleul (1996). P-Time Petri Nets for manufacturing systems. In: *International Workshop on Discrete Event Systems, WODES'96*. Edinburgh (U.K.). pp. 94–102.

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.

Lime, D. and O. H. Roux (2004). http://www.irccyn.ec-nantes.fr/irccyn/d/fr/equipes/TempsReel/logs/software-2-romeo.

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.

Penczek, W. and A. Polrola (2004). Specification and model checking of temporal properties in time Petri nets and timed automata. In: *The 25th International Conference on Application and Theory of Petri Nets, (ICATPN 2004)*. Vol. 3099 of *Lecture Notes in Computer Science*. Springer-Verlag. Bologna, Italy.

Pezze, M. and M. Toung (1999). Time Petri nets: A primer introduction. Tutorial presented at the Multi-Workshop on Formal Methods in Performance Evaluation and Applications, Zaragoza, Spain.

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

Ramchandani, C. (1974). Analysis of asynchronous concurrent systems by timed Petri nets. PhD thesis. Massachusetts Institute of Technology. Cambridge, MA. Project MAC Report MAC-TR-120.

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.

Vicario, E. (2001). Static analysis and dynamic steering of time-dependent systems. *IEEE transactions on software engineering* **27**(8), 728–748.

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 $s \xrightarrow{\delta} s'$.

– *Continous transitions*
  - 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 $Inv(l) = (\wedge_{x \in X}(x \leq \min_{t' \in trans(x)} \beta(t')))$.

    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.

– *Discrete transitions*
  - Let us 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 $\forall t' \in \mathit{enabled}(M')$, $\exists x \in X, \nu'_{\mathcal{T}}(t') = \nu'_{\mathcal{A}}(x)$ *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 6. *There are no location l of the state class timed automaton in which two clocks x, y used in l ($x, y \in clk(l)$) 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. □

Theorem 7. *There are no* orthogonal *clocks in the state class timed automaton.*

Proof. Let $x_0, \dots, x_n$ be the clocks of the state class timed automaton. When x_n was created, all the indexes lower than n where used since the new index is chosen as the smallest available one. As a consequence, all of the clocks appear in the same class and none of their pairs can be orthogonal. □

Notes

* This is an updated version of the paper originally published in Journal of Discrete Event Dynamic Systems (jDEDS), 16(2):179-205, 2006.

Affiliation of authors: IRCCyN (Institut de Recherche en Communication et Cybernétique de Nantes) 1, rue de la Noë B.P. 92101 F-44321 NANTES cedex 3 (France).

`{Didier.Lime | Olivier-h.Roux}@irccyn.ec-nantes.fr`