

From Time Petri Nets to Timed Automata

Franck Cassez and Olivier H. Roux

Institut de Recherche en Communication et Cybernétique de Nantes (IRCCyN)

France

1. Introduction

In this chapter we introduce a formalism, *Time Petri Nets (TPNs)*, to model real-time systems. We compare it with another well-known formalism, *Timed Automata (TA)*, used for specifying timed systems. We precisely define the semantics of TPNs and TA and compare them according to two criteria: the *languages* (or set of behaviours) they can generate, and the *trees* (or branching behaviours) they can generate. We show that every TPN can be translated into an equivalent¹ TA.

Then, we introduce a real-time logic to specify properties of real-time systems. We show how to check that a given TPN satisfies a property written in this logic. For this, we use our translation² from TPNs to TA and check the property on the equivalent TA. Finally we briefly report on experiments for checking real-time properties of TPNs using this framework.

1.1. Petri Nets with Time

The two main extensions of Petri Nets with time are Time Petri Nets (TPNs) (Merlin, 1974) and Timed Petri Nets (Ramchandani, 1974). In a TPN a transition can fire within a time interval whereas for Timed Petri Nets it fires as soon as possible. For Timed Petri Nets, time can be considered relative to places or transitions (Sifakis, 1980; Pezzè, 1999). It is interesting to formally compare the different classes of Petri Nets with time: this gives a better idea of what one subclass should be used for. The expressive power of (time) Petri Nets can be compared w.r.t. the set of (timed) behaviors they can generate. One class C is a subclass of another C' , if for every net n in C , there is a net n' in C' which can generate the same behaviors as n . In this case, we say that the class C is less expressive than C' . For instance, the two subclasses P-Timed Petri Nets and T-Timed Petri Nets are expressively equivalent (Sifakis, 1980; Pezzè, 1999) (i.e., P-Timed Petri Nets are less expressive than T-Timed Petri Nets and vice-versa). The same subclasses are defined for TPNs i.e., T-TPNs and P-TPNs. Both classes of Timed Petri Nets are less expressive than both P-TPNs and T-TPNs (Pezzè, 1999). P-TPNs and T-TPNs are incomparable (Khansa et al., 1996). Finally TPNs are less expressive than Time Stream Petri Nets (Diaz and Senac, 1994) which were introduced to model multimedia applications.

Another way of comparing two classes is to determine the status of different decision problems (e.g., *reachability*, *coverability*, *boundedness*) for the two classes. For instance, reachability is undecidable for TPNs, as well as boundedness. Recent work (de Frutos Escrig et al., 2000; Abdulla and Nylén, 2001) considers timed arc Petri nets where each token has a clock representing its "age". The authors prove that coverability and boundedness are decidable for this class of Petri nets by applying a backward exploration technique. They use a lazy (non-urgent) behavior of the net: the firing of transitions may be delayed, even if that implies that

¹This equivalence is formally defined in the chapter.

²This translation preserves the properties of this logic.

some transitions are disabled because their input tokens become too old.

The class T-TPNs is the most commonly-used subclass of TPNs to specify real-time systems. In this chapter, we focus on this subclass that will be henceforth referred to as TPNs. For classical TPNs (with closed intervals), boundedness is undecidable (Berthomieu and Diaz, 1991), and papers on this model report undecidability results, or decidability under the assumption that the TPN is bounded, e.g., reachability in (Popova, 1991).

1.2. Verifying Time Petri Nets

The main objective in specifying real-time systems with formalisms like TPNs is to build a model of a system S , and be able to mathematically reason about it. By reasoning we mean “verifying that some properties are satisfied on the model”. The properties we would like to check range from simple ones like “the system cannot reach a bad state” which are *reachability properties*, to more involved properties like “after each failure, the system will reach a stable state within 10 time units”, which are *quantitative real-time properties*. Given a formal model S and a temporal logic formula φ , verifying that S satisfies φ is usually achieved by a *model-checking* algorithm: such an algorithm checks that S is a model of the formula φ . Hence the process of verifying that a formal model of a system satisfies a property in a temporal logic is often called *model-checking*.

Algorithms for verifying properties on TPNs have been designed for more than a decade. Formally, the *behavior* of a TPN can be defined by timed firing sequences which are sequences of pairs (t, d) where t is a transition of the TPN and $d \in \mathbb{R}_{\geq 0}$. A sequence of transitions like $\omega = (t_1, d_1) (t_2, d_2) \dots (t_n, d_n) \dots$ indicates that t_1 is fired after d_1 time units, then t_2 is fired after d_2 time units have elapsed since t_1 was fired, and so on, so that transition t_i is fired at absolute time $\sum_{k=1}^i d_k$. A *marking* M is *reachable* in a TPN if there is a timed firing sequence ω from the initial marking M_0 to M . Reachability analysis of TPNs relies on the construction of the so-called State-Class Graph (SCG) that was introduced in (Berthomieu and Menasche, 1983) and later refined in (Berthomieu and Diaz, 1991). It has been recently improved in (Lilius, 1998) by using partial-order reduction methods.

For bounded TPNs, the SCG construction obviously solves the *marking reachability* problem: “Given a marking M , is it possible to reach M from M_0 ?”. If one wants to solve the *state reachability* problem: “Given M and $v \in \mathbb{R}_{\geq 0}$ and a transition t , can we reach a marking M such that transition t has been enabled for v time units?”, the SCG is not precise enough and an alternative graph, the *Strong State Class Graph* has to be built for this purpose (Berthomieu and Vernadat, 2003). The previous two graphs allow for checking *qualitative* properties written in a real-time logic called LTL (Emerson, 1990). A more powerful real-time logic, CTL* (Emerson, 1990), can be checked on TPNs using yet another more precise graph.

Anyway, none of the previous graphs is a good³ abstraction (accurate enough) for checking *quantitative* real-time properties e.g., “it is not possible to stay in marking M more than n time units” or “from marking M , marking M' is always reached within n time units”. In this chapter we introduce a logic to specify such quantitative real-time properties and present an algorithm to check for such properties on TPNs.

1.3. Timed Automata

Timed Automata (TA) were introduced by Alur & Dill (Alur and Dill, 1994) and have since been extensively studied. They are now widely used to model real-time systems. TA forms an extension of finite automata with dense time *clocks* and enables one to specify real-time

³The use of *observers* is of little help as it requires to specify a property as a TPN; thus it is hard to specify properties on markings.

systems. It has been shown that model-checking a quantitative real-time logic, called TCTL, is decidable (Alur and Dill, 1994; Henzinger et al., 1994) for TA and some of their extensions (Bouyer et al., 2000). There also exist several efficient tools like UPPAAL (Larsen et al., 1997), KRONOS (Yovine, 1997) and CMC (Laroussinie and Larsen, 1998) for model-checking TA and many real-time industrial applications have been specified and successfully verified with them. In this chapter, we show how to translate TPNs into equivalent TA. This enables us to use the technology and tools developed for TA to verify TPNs.

1.4. Outline of the Chapter

In Section 2 we fix notations and provide basic notions for defining the formal semantics of Time Petri Nets and Timed Automata. In Section 3, we introduce Time Petri Nets and define their semantics. We also give the main properties of this model together with an algorithm to compute a finite representation of the state space of a Time Petri Net; this algorithm can be used to check marking reachability. In Section 4, we compare the expressiveness of Timed Automata and Time Petri Nets and show that they are equivalent w.r.t. language equivalence. We give the translation from TPNs to TA and show in Section 5 how to check quantitative properties on TPNs using a timed temporal logic. In Section 6, we apply the framework defined in Section 4 on some examples. Finally, we conclude with recent or ongoing work on this subject in Section 7.

2. Preliminaries

Let Σ be a finite alphabet. Σ^* (resp. Σ^ω) denotes the set of finite (resp. infinite) sequences over Σ and $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$ (called words in the sequel). By convention if $u \in \Sigma^\omega$, then the *length* of u , denoted $|u|$, is ω ; otherwise if $u = a_1 \cdots a_n$, $|u| = n$. We also use $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$ where $\varepsilon \notin \Sigma$, where ε is the empty word. B^A stands for the set of mappings from A to B . If A is finite and $|A| = n$, an element of B^A is also a vector in B^n . The usual operators $+$, $-$, $<$ and $>$ are used on vectors of A^n with $A = \mathbb{N}, \mathbb{Q}, \mathbb{R}$ and are the point-wise extensions of their counterparts in A . The set $\mathbb{B} = \{\text{tt}, \text{ff}\}$ denotes the boolean truth values *true* and *false*, $\mathbb{R}_{\geq 0}$ denotes the set of non-negative reals and $\mathbb{R}_{> 0} = \mathbb{R}_{\geq 0} \setminus \{0\}$. A *valuation* v for the set of variables X is an element of $\mathbb{R}_{\geq 0}^X$. For $v \in \mathbb{R}_{\geq 0}^X$ and $d \in \mathbb{R}_{\geq 0}$, $v + d$ denotes the valuation defined by $(v + d)(x) = v(x) + d$, and for $X' \subseteq X$, $v|_{X'} \mapsto 0$ denotes the valuation v' with $v'(x) = 0$ for $x \in X'$ and $v'(x) = v(x)$ otherwise. $\mathbf{0}$ denotes the valuation s.t. $\forall x \in X, \mathbf{0}(x) = 0$. An *atomic constraint* is a formula of the form $x \bowtie c$ for $x \in X$, $c \in \mathbb{Q}_{\geq 0}$ and $\bowtie \in \{<, \leq, \geq, >\}$. We denote $\mathcal{C}(X)$ the set of *constraints* over a set of variables X which consists of conjunctions of atomic constraints. Given a constraint $\varphi \in \mathcal{C}(X)$ and a valuation $v \in \mathbb{R}_{\geq 0}^X$, we denote $\varphi(v) \in \mathbb{B}$ the truth value obtained by substituting each occurrence of x in φ by $v(x)$.

2.1. Timed Languages and Timed Transition Systems

Let Σ be a fixed finite alphabet s.t. $\varepsilon \notin \Sigma$ and A be a finite alphabet which can contain ε .

Definition 1 (Timed Word) A timed word w over Σ is a finite or infinite sequence

$$w = (a_0, d_0)(a_1, d_1) \cdots (a_n, d_n) \cdots$$

s.t. for each $i \geq 0$, $a_i \in \Sigma$, $d_i \in \mathbb{R}_{\geq 0}$ and $d_{i+1} \geq d_i$.

A timed word $w = (a_0, d_0)(a_1, d_1) \cdots (a_n, d_n) \cdots$ over Σ can also be viewed as a pair $(v, \tau) \in \Sigma^\infty \times \mathbb{R}_{\geq 0}^\infty$ s.t. $|v| = |\tau|$. The value d_k gives the absolute time (considering the initial instant is 0) of action a_k . We write $\text{Untimed}(w) = a_0 a_1 \cdots a_n \cdots$ for the untimed part of w , and

$\text{Duration}(w) = \sum_{k \geq 0} d_k$ for the duration of the timed word w . We let $\text{TW}^*(\Sigma)$ (resp. $\text{TW}^\omega(\Sigma)$) be the set of finite (resp. infinite) timed words over Σ and $\text{TW}(\Sigma) = \text{TW}^*(\Sigma) \cup \text{TW}^\omega(\Sigma)$. A *timed language* L over Σ is a set of timed words i.e., any set $L \subseteq \text{TW}(\Sigma)$.

Timed Transition Systems (TTS) are usual transition systems with two types of labels: discrete labels for events and positive reals' labels for time elapsing. Consequently, they have two types of transitions: discrete transitions and time transitions:

Definition 2 (Timed Transition System) A timed transition system (TTS) (over a set of actions A) is a tuple $S = (Q, Q_0, A, \rightarrow, F, R)$ where:

- Q is a set of states;
- $Q_0 \subseteq Q$ is the set of initial states;
- A is a finite set of actions disjoint from $\mathbb{R}_{\geq 0}$;
- $\rightarrow \subseteq Q \times (A \cup \mathbb{R}_{\geq 0}) \times Q$ is a set of edges. If $(q, e, q') \in \rightarrow$, we also write $q \xrightarrow{e} q'$;
- $F \subseteq Q$ and $R \subseteq Q$ are respectively the set of final and repeated states.

For a time transition $q \xrightarrow{d} q'$ with $d \in \mathbb{R}_{\geq 0}$, d denotes a delay and not an absolute time. We assume that in any TTS there is a transition $q \xrightarrow{0} q'$ and in this case $q = q'$. A run ρ of length $n \geq 0$ is a finite ($n < \omega$) or infinite ($n = \omega$) sequence of alternating time and discrete transitions of the form:

$$\rho = q_0 \xrightarrow{d_0} q'_0 \xrightarrow{a_0} q_1 \xrightarrow{d_1} q'_1 \xrightarrow{a_1} \dots q_n \xrightarrow{d_n} q'_n \dots$$

with $d_i \in \mathbb{R}_{\geq 0}$ and $a_i \in A$. We write $\text{first}(\rho) = q_0$. We assume that a finite run ends with a time transition d_n . If ρ ends with d_n , we let $\text{last}(\rho) = q'_n$ and write $q_0 \xrightarrow{d_0 a_0 \dots d_n} q'_n$. We write $q \xrightarrow{*} q'$ if there is run ρ s.t. $\text{first}(\rho) = q_0$ and $\text{last}(\rho) = q'$. The set of *reachable states* in S is the set of states q s.t. $q_0 \xrightarrow{*} q$ for some $q_0 \in Q_0$. The *trace* of an infinite run ρ is the timed word $\text{trace}(\rho) = (a_{i_0}, d_0 + \dots + d_{i_0}) \cdot \dots \cdot (a_{i_k}, d_0 + \dots + d_{i_k}) \cdot \dots$ that consists of the sequence of letters $(a_{i_k})_{k \in \mathbb{N}}$ of $A \setminus \{\varepsilon\}$. If ρ is a finite run, we define the trace of ρ by $\text{trace}(\rho) = (a_{i_0}, d_0 + \dots + d_{i_0}) \cdot \dots \cdot (a_{i_k}, d_0 + \dots + d_{i_k})$ where the a_{i_k} are in $A \setminus \{\varepsilon\}$. By definition $\text{Untimed}(\rho) = \text{Untimed}(\text{trace}(\rho))$ and $\text{Duration}(\rho) = \sum_{d_k \in \mathbb{R}_{\geq 0}} d_k$. A run ρ is *zeno* if $|\text{Untimed}(\rho)| = \omega$ and $\text{Duration}(\rho) = \text{Duration}(\text{trace}(\rho)) = r$ with $r \in \mathbb{R}_{\geq 0}$.

A run is *initial* if $\text{first}(\rho) \in Q_0$. A run ρ is *accepting* if *i)* either ρ is a finite initial run and $\text{last}(\rho) \in F$ or *ii)* ρ is an infinite initial run and there is a state $q \in R$ that appears infinitely often on ρ . A timed word $w = (a_i, d_i)_{i \geq 0}$ is *accepted* by S if there is an accepting run ρ s.t. $\text{trace}(\rho) = w$. The *timed language*, $\mathcal{L}(S)$, accepted by S is the set of finite and infinite timed words accepted by S . We let $\mathcal{L}^*(S)$ (resp. $\mathcal{L}^\omega(S)$) be the set of finite (resp. infinite) timed words accepted by S . When we omit the sets F and R , it means that $F = R = Q$ i.e., every state is final and repeated. In this case the language accepted by the TTS is *prefix-closed*, that is, if the TTS accepts a timed word w , then it also accepts every finite prefix of w .

2.2. Equivalences on Timed Transition Systems

We can define two types of equivalences on TTS: roughly speaking, language equivalences are based on the set of timed words two TTS generate. Branching equivalences (like simulation or bisimulation) are finer in the sense that they involve the branching structure of the two TTS.

Definition 3 (Timed Language Equivalence) Let $S_i = (Q_i, Q_0^i, A, \rightarrow_i, F_i, R_i)$ with $i = 1, 2$ be two TTS. S_1 and S_2 are language equivalent, denoted $S_1 =_{\mathcal{L}} S_2$, if $\mathcal{L}(S_1) = \mathcal{L}(S_2)$.

Branching equivalences can be defined on TTS and are more constraining:

Definition 4 (Strong Timed Similarity) Let $S_1 = (Q_1, Q_0^1, A, \longrightarrow_1, F_1, R_1)$ and $S_2 = (Q_2, Q_0^2, A, \longrightarrow_2, F_2, R_2)$ be two TTS and \preceq be a binary relation over $Q_1 \times Q_2$. We write $s \preceq s'$ for $(s, s') \in \preceq$. \preceq is a strong (timed) simulation relation of S_1 by S_2 if:

1. if $s_1 \in F_1$ (resp. $s_1 \in R_1$) and $s_1 \preceq s_2$ then $s_2 \in F_2$ (resp. $s_2 \in R_2$);
2. if $s_1 \in Q_0^1$ there is some $s_2 \in Q_0^2$ s.t. $s_1 \preceq s_2$;
3. if $s_1 \xrightarrow{d}_1 s'_1$ with $d \in \mathbb{R}_{\geq 0}$ and $s_1 \preceq s_2$ then $s_2 \xrightarrow{d}_2 s'_2$ for some s'_2 , and $s'_1 \preceq s'_2$;
4. if $s_1 \xrightarrow{a}_1 s'_1$ with $a \in A$ and $s_1 \preceq s_2$ then $s_2 \xrightarrow{a}_2 s'_2$ and $s'_1 \preceq s'_2$.

A TTS S_2 strongly simulates S_1 if there is a strong (timed) simulation relation of S_1 by S_2 . We write $S_1 \preceq_S S_2$ in this case.

When there is a strong simulation relation \preceq of S_1 by S_2 and \preceq^{-1} is also a strong simulation relation⁴ of S_2 by S_1 , we say that \preceq is a strong (timed) bisimulation relation between S_1 and S_2 and use \approx instead of \preceq . Two TTS S_1 and S_2 are strongly (timed) bisimilar if there exists a strong (timed) bisimulation relation between S_1 and S_2 . We write $S_1 \approx_S S_2$ in this case.

Let $S = (Q, Q_0, \Sigma_\varepsilon, \longrightarrow, F, R)$ be a TTS. We define the ε -abstract TTS $S^\varepsilon = (Q, Q_0^\varepsilon, \Sigma, \longrightarrow_\varepsilon, F, R)$ (with no ε -transitions) by:

- $q \xrightarrow{d}_\varepsilon q'$ with $d \in \mathbb{R}_{\geq 0}$ iff there is a run $\rho = q \xrightarrow{*} q'$ with $\text{Untimed}(\rho) = \varepsilon$ and $\text{Duration}(\rho) = d$,
- $q \xrightarrow{a}_\varepsilon q'$ with $a \in \Sigma$ iff there is a run $\rho = q \xrightarrow{*} q'$ s.t. $\text{Untimed}(\rho) = a$ and $\text{Duration}(\rho) = 0$,
- $Q_0^\varepsilon = \{q \mid \exists q' \in Q_0 \mid q' \xrightarrow{*} q \text{ and } \text{Duration}(\rho) = 0 \wedge \text{Untimed}(\rho) = \varepsilon\}$.

Definition 5 (Weak Timed Similarity) Let $S_i = (Q_i, Q_0^i, \Sigma_\varepsilon, \longrightarrow_i, F_i, R_i)$ for $i = 1, 2$ be two TTS and \preceq be a binary relation over $Q_1 \times Q_2$. \preceq is a weak (timed) simulation relation of S_1 by S_2 if it is a strong timed simulation relation of S_1^ε by S_2^ε . A TTS S_2 weakly simulates S_1 if there is a weak (timed) simulation relation of S_1 by S_2 . We write $S_1 \preceq_{\mathcal{W}} S_2$ in this case.

When there is a weak simulation relation \preceq of S_1 by S_2 and \preceq^{-1} is also a weak simulation relation of S_2 by S_1 , we say that \preceq is a weak (timed) bisimulation relation between S_1 and S_2 and use \approx instead of \preceq . Two TTS S_1 and S_2 are weakly (timed) bisimilar if there exists a weak (timed) bisimulation relation between S_1 and S_2 . We write $S_1 \approx_{\mathcal{W}} S_2$ in this case. Note that if $S_1 \preceq_S S_2$ then $S_1 \preceq_{\mathcal{W}} S_2$ and if $S_1 \preceq_{\mathcal{W}} S_2$ then $\mathcal{L}(S_1) \subseteq \mathcal{L}(S_2)$.

3. Time Petri Nets

Time Petri Nets were introduced in (Merlin, 1974) and extend Petri Nets with timing constraints on the firings of transitions. In this section, we give the definitions and semantics of an extended class of TPNs using open and/or closed intervals (Bérard et al., 2005a; Cassez and Roux, 2006).

⁴ $S_2 \preceq^{-1} S_1 \iff S_1 \preceq S_2$.

3.1. Definition and Semantics

Definition 6 (Time Petri Net) A Time Petri Net (TPN) \mathcal{T} is a tuple $(P, T, \bullet(\cdot), (\cdot)\bullet, M_0, (\alpha, \beta))$ where:

- $P = \{p_1, p_2, \dots, p_m\}$ is a finite set of places;
- $T = \{t_1, t_2, \dots, t_n\}$ is a finite set of transitions;
- $\bullet(\cdot) \in (\mathbb{N}^P)^T$ is the backward incidence mapping; $(\cdot)\bullet \in (\mathbb{N}^P)^T$ is the forward incidence mapping;
- $M_0 \in \mathbb{N}^P$ is the initial marking;
- $\alpha \in (\mathbb{Q}_{\geq 0})^T$ and $\beta \in (\mathbb{Q}_{\geq 0} \cup \{\infty\})^T$ are respectively the earliest and latest firing time mappings.

A labeled TPN is a pair (\mathcal{T}, L) where $L : T \rightarrow \Sigma_\varepsilon$.

The semantics of TPNs can be given by a Timed Transition System. $v \in (\mathbb{R}_{\geq 0})^n$ is a valuation such that each value v_i is the elapsed time since transition t_i was last enabled. $\mathbf{0}$ is the initial valuation with $\forall i \in [1..n], \mathbf{0}_i = 0$. A marking M of a TPN is a mapping in \mathbb{N}^P and if $M \in \mathbb{N}^P$, $M(p_i)$ is the number of tokens in place p_i . A transition t_i is enabled in a marking M iff $M \geq \bullet t_i$ and $\alpha(t_i) \leq v_i \leq \beta(t_i)$. The predicate $\uparrow \text{enabled}(t_k, M, t_i) \in \mathbb{B}$ is true if t_k is enabled by the firing of transition t_i from marking M , and false otherwise. This definition of enabledness is based on (Berthomieu and Diaz, 1991; Aura and Lilius, 2000) which is the most common one. In this framework, a transition t_k is *newly enabled* after firing t_i from marking M if “it is not enabled by $M - \bullet t_i$ and is enabled by $M' = M - \bullet t_i + t_i \bullet$ ” (Berthomieu and Diaz, 1991). Formally this gives:

$$\uparrow \text{Enabled}(t_k, M, t_i) = (M - \bullet t_i + t_i \bullet \geq \bullet t_k) \wedge ((M - \bullet t_i < \bullet t_k) \vee (t_k = t_i)) \quad (1)$$

Definition 7 (Semantics of a TPN) The semantics of a TPN \mathcal{T} is a timed transition system $S_{\mathcal{T}} = (Q, q_0, T, \rightarrow)$ where: $Q = \mathbb{N}^P \times (\mathbb{R}_{\geq 0})^n$, $q_0 = (M_0, \mathbf{0})$, $\rightarrow \in Q \times (T \cup \mathbb{R}_{\geq 0}) \times Q$ consists of:

- the discrete transition relation is defined for all $t_i \in T$ by $(M, v) \xrightarrow{t_i} (M', v')$ iff:

$$\begin{cases} M \geq \bullet t_i \wedge M' = M - \bullet t_i + t_i \bullet \\ \alpha(t_i) \leq v_i \leq \beta(t_i) \\ v'_k = \begin{cases} 0 & \text{if } \uparrow \text{Enabled}(t_k, M, t_i), \\ v_k & \text{otherwise.} \end{cases} \end{cases}$$

- and the continuous transition relation is defined for all $d \in \mathbb{R}_{\geq 0}$ by $(M, v) \xrightarrow{d} (M, v')$ iff:

$$\begin{cases} v' = v + d \\ \forall k \in [1..n], (M \geq \bullet t_k \implies v'_k \leq \beta(t_k)) \end{cases}$$

A run of a time Petri net \mathcal{T} is a (finite or infinite) path in $S_{\mathcal{T}}$ starting in q_0 . The set of runs of \mathcal{T} is denoted by $\text{Runs}(\mathcal{T})$. As a shorthand we write $(M, v) \xrightarrow{d} (M', v')$ for a sequence of time elapsing and discrete steps like $(M, v) \xrightarrow{d} (M'', v'') \xrightarrow{e} (M', v')$. A state (M, v) is reachable in \mathcal{T} if $(M_0, v_0) \xrightarrow{*} (M, v)$. $\text{ReachState}(\mathcal{T})$ is the reachable set of states in \mathcal{T} . A marking M is

reachable in \mathcal{T} if there is a state $(M, \nu) \in \text{ReachState}(\mathcal{T})$. The set of *reachable markings* of \mathcal{T} is denoted $\text{ReachMark}(\mathcal{T})$. If the set $\text{ReachMark}(\mathcal{T})$ is finite we say that \mathcal{T} is *bounded*, otherwise it is *unbounded*.

If we add two sets of markings, F (final) and R (repeated), we can define the languages accepted by \mathcal{T} . We let S_F (resp. S_R) be the set of states (M, ν) of $S_{\mathcal{T}}$ s.t. $M \in F$ (resp. $M \in R$). The timed language $\mathcal{L}(\mathcal{T})$ accepted by \mathcal{T} is the timed language accepted by $S_{\mathcal{T}}$ with sets S_F and S_R as final and repeated states. Similar definitions hold for $\mathcal{L}^*(\mathcal{T})$ and $\mathcal{L}^\omega(\mathcal{T})$. Moreover, for a labeled TPN (\mathcal{T}, L) , the languages accepted by (\mathcal{T}, L) are the languages accepted by \mathcal{T} where, in each timed word, a transition label t is replaced by $L(t)$.

In Definition 7, we have implicitly assumed that the constraints given on each transition of the TPN by the mapping (α, β) are *closed constraints*. We can also consider that the firing intervals of a transition are left and/or right open. The semantics is defined accordingly substituting $<$ to \leq : assume $\alpha(t_i), \beta(t_i)$ is left-closed and right open; in this case the discrete transition relation for t_i is defined using $\alpha(t_i) \leq \nu_i < \beta(t_i)$ instead of $\alpha(t_i) \leq \nu_i \leq \beta(t_i)$ in Definition 7 and for the continuous transition relation, we should use $\nu'_i < \beta(t_i)$. Most of the results we give in this chapter hold for any type of intervals. In the sequel, we denote \mathcal{TPN} the most general class of TPNs using open or closed intervals and $\mathcal{TPN}(\leq, \geq)$ for the subclass that uses only closed intervals.

Our semantics (Bérard et al., 2005a; Cassez and Roux, 2006) is based on the common definition of (Berthomieu and Diaz, 1991; Aura and Lilius, 2000) for safe TPNs but still there are some advantages using ours. First, previous formal semantics (Berthomieu and Diaz, 1991; Lilius, 1998; Pezzè, 1999; Aura and Lilius, 2000) for TPNs required the TPNs to be *safe*. Our semantics encompasses the whole class of TPNs and is fully consistent with the previous semantics when restricted to safe TPNs⁵. Thus, we have given a semantics to multiple enabledness of transitions which seems the most simple and adequate. Indeed, several interpretations can be given to multiple enabledness (Berthomieu and Diaz, 1991).

Second, some variations can be found in the literature about TPNs concerning the firing of transitions. The paper (Pezzè, 1999) considers two distinct semantics: Weak Time Semantics (WTS) and Strong Time Semantics (STS). According to WTS, a transition *can* be fired only in its time interval whereas in STS, a transition *must* fire within its firing interval unless disabled by the firing of others. The most commonly used semantics is STS as in (Merlin, 1974; Berthomieu and Diaz, 1991; Pezzè, 1999; Aura and Lilius, 2000). A more complete study on the firing policy in TPNs can be found in (Bérard et al., 2005c).

Third, it is possible for the TPN to generate zeno runs or to be unbounded. When it is unbounded, the discrete component (i.e., marking) of the state space of the timed transition system is infinite. If $\forall i, \alpha(t_i) > 0$ then the TPN cannot generate any zeno word and time diverges on each run. Otherwise, if the TPN is bounded and at least one lower bound is 0, whether or not a TPN accepts a zeno run can be decided (Henzinger et al., 1994) (for instance using the equivalent timed automaton we build in section 4.3). In the next subsections we summarize the status of basic decision problems for TPNs.

3.2. Decidable and Undecidable Problems for TPNs

Let $\mathcal{T} = (P, T, \bullet(\cdot), (\cdot)^\bullet, M_0, (\alpha, \beta))$ be a TPN with $|P| = p$, $|T| = n$ and let $S_{\mathcal{T}} = (Q, q_0, T, \rightarrow)$ its semantics. Let us consider the following problems:

- (1) The *marking reachability* problem: Given $m \in \mathbb{N}^p$, is m in $\text{ReachMark}(\mathcal{T})$?
- (2) The *boundedness* problem: Is there a bound $\bar{b} \in \mathbb{N}^p$ s.t. $\forall m \in \text{ReachMark}(\mathcal{T}), m \leq \bar{b}$?

⁵If we except the difference with (Lilius, 1998) in the definition of the reset instants for newly enabled transitions.

- (3) The \bar{k} -boundedness problem: Given $\bar{k} = (k_1, k_2, \dots, k_p) \in \mathbb{N}^p$, is it true that for all $m \in \text{ReachMark}(\mathcal{T})$, $m \leq \bar{k}$?
- (4) The state reachability problem: Given $(M, \nu) \in \mathbb{N}^P \times \mathbb{R}_{\geq 0}^n$, is (M, ν) in $\text{ReachState}(\mathcal{T})$?
- (5) The liveness problem: For $t \in T$, $(M, \nu) \in \text{ReachState}(\mathcal{T})$, is there any run $(M, \nu) \xrightarrow{*} (M', \nu')$ such that $(M', \nu') \xrightarrow{t} (M'', \nu'')$?
- (6) The emptiness problem: Is the language accepted by \mathcal{T} empty i.e., $\mathcal{L}(S_{\mathcal{T}}) = \emptyset$?
- (7) The universal problem: Does \mathcal{T} accept all the finite (resp. infinite) timed words over T i.e., $\mathcal{L}^*(\mathcal{T}) = \text{TW}^*(T)$ (resp. $\mathcal{L}^\omega(\mathcal{T}) = \text{TW}^\omega(T)$)?

Problem (1) was proved undecidable for TPNs in (Jones et al., 1977). It follows that all problems (1–2) and (4–5) are undecidable for TPNs. There are however a number of sufficient conditions for the boundedness property of TPNs, the stronger being that the underlying Petri net is bounded, and the latter is known decidable. For Time Petri nets, we have the following results:

Theorem 1 (Berthomieu and Diaz, 1991) \bar{k} -boundedness (3) is decidable for TPNs. State reachability (4) and liveness (5) are decidable for bounded TPNs.

Theorem 2 (Cassez and Roux, 2006) The emptiness problem (6) is PSPACE-complete for bounded TPNs.

Theorem 3 (Bérard et al., 2005a) The universal problem (7) is undecidable for bounded TPNs.

Theorem 2 is obtained by reducing the emptiness problem for bounded TPNs to the emptiness problem for TA (using the result of Section 4). Theorem 3 is more difficult to obtain and consists in translating a TA into an equivalent TPN (using the results of Table 1). In the next two subsections we describe two ways of solving the state reachability (4) problem.

3.3. State Reachability Using The State Class Method

To decide problem 4, we can compute a finite representation of the state space of a bounded TPN. If, on this representation G , we can decide whether a state (M, ν) belongs to G we have an algorithm to check state reachability.

The first method to compute the state space of a TPN is based on the aggregation of states into classes and was introduced by BERTHOMIEU and DIAZ in (Berthomieu and Diaz, 1991).

Definition 8 (State Class) A State Class C of a TPN is a pair (M, D) where M is a marking and D is a set of inequalities, over a set of variables X , called the firing domain. The value of a variable $x_i \in X$ of the firing domain represents the firing time of the enabled transition t_i relatively to the time when the class C was entered.

To obtain an abstract representation of the state space of a TPN, it is possible to compute a graph of state classes called the State Class Graph (SCG). An edge in the SCG from a class to another is defined by:

Definition 9 (State Class Transition) Given a class $C = (M, D)$ and a transition t_j enabled in (M, D) , the t_j -successor class, $C' = (M', D')$, of C is computed as follows:

1. Compute the new marking $M' = M - \bullet t_j + t_j \bullet$;

2. Add the constraints $x_j \leq x_i$ for each $i \neq j$ to D . Then substitute in D each variable $x_i, i \neq j$ by $x'_i + x_j$ where x'_i are new fresh variables. The new domain obtained this way is D' ;
3. Eliminate x_j from D' using for instance the Fourier-Motzkin method;
4. Replace x'_i by x_i in D' .
5. Compute a canonical form of D' using for instance the Floyd-Warshall algorithm.

Computing all the edges of the SCG from the initial class of a TPN is called the *State-Class Method*. In the state class method, the domain associated with a class is relative to the time when the class was entered and as the transformation (we reset the time origin) is irreversible, absolute values of clocks cannot be obtained easily. The graph produced is an abstraction of the state space for which temporal information has been lost. Often, the graph has more classes than the number of markings of the TPN. Edges between classes are no longer labeled with a firing constraint but only with the name of the fired transition: the state class graph accepts the untimed (prefix closed) language of the TPN. If the TPN is bounded the SCG of a TPN is finite. The SCG computation is implemented in the tool TINA (Berthomieu and Vernadat, 2006a).

As a consequence of the SCG construction, sophisticated temporal properties are not easy to check. Indeed, the domain associated with a marking is made of relative values of clocks and the function to compute domains is not bijective. Consequently, domains cannot be easily used to verify properties involving constraints on clocks.

In order to get rid of these limitations, several papers have proposed to construct a different state class graph by modifying the equivalence relation between classes. To our knowledge, the methods proposed in (Berthomieu and Vernadat, 2003) depend on the property to check. Checking LTL or CTL properties will lead to different state class graphs.

Another limitation of the methods and associated tools to check properties of TPN using the SCG, is the need to compute the whole state graph while only the reachability of a given marking is needed (safety properties). The graph is then analyzed by a model checker for finite state systems. Using observers is even more costly: actually, for each property to be checked, a new state class graph has to be built and the observer can dramatically increase the size of the state space.

3.4. State Reachability Using a Zone Based Abstraction

Another method to compute a finite representation of the state space of a bounded TPN was recently proposed by GARDEY *et al.* in (Gardey *et al.*, 2003; Gardey *et al.*, 2006). It is based on the Region Graph introduced for Timed Automata (Alur and Dill, 1994; Rokicki, 1993).

A *zone* is a convex union of regions as defined by ALUR and DILL (Alur and Dill, 1994). For short, considering n clocks, a zone is a convex subset of $\mathbb{R}_{\geq 0}^n$. A zone can be represented by a conjunction of constraints on pairs of clocks: $x_i - x_j \sim c$ where $\sim \in \{<, \leq, =, \geq, >\}$ and $c \in \mathbb{N}$.

The graph which is computed in this case is a *simulation graph* of a TPN which is an abstract and *symbolic* representation of the state space of the TPN. Given the initial marking M_0 and an initial zone Z_0 (the values of clocks for Z_0 are $\mathbf{0}$), time and discrete successors are iteratively computed by letting time pass or by firing transitions. Let M be a marking and Z a zone. The computation of the reachable markings from (M, Z) is done as follows:

1. Compute the possible states reachable by time elapsing: we let \vec{Z} be the set of such states. It is obtained by setting all upper bounds of constraints on clocks defining Z to infinity;

2. Select only the possible valuations of clocks for which M could exist, i.e., the valuations of clocks are smaller than the latest firing time of any enabled transitions;

$$Z' = \vec{Z} \cap \bigwedge_{t_i \in Enabled_{M,Z}} \{x_i \leq \beta_i\}$$

where $\{x \leq \beta\}$ denotes the zone defined by the constraint $x \leq \beta$. Z' is the maximal zone starting from Z for which the marking M exists.

3. Determine the firable transitions in (M, Z') : t_i is firable if $Z' \cap \{x_i \geq \alpha_i\}$ is a non empty zone.
4. For each firable transition t_i leading to a marking M_i , compute the zone obtained when we enter the new marking M_i as follows:

$$Z_i = (Z' \cap \{x_i \geq \alpha_i\})[X_e := 0]$$

where X_e is the set of newly enabled clocks. This means that each transition which is newly enabled has its clock reset. Then, Z_i is a zone for which the new marking M_i is reachable.

It is then possible to compute all the reachable pairs (M, Z) reachable from (M_0, Z_0) using the previous method. This way we obtain a forward algorithm to compute the simulation graph for a bounded TPN.

An algorithm to enumerate reachable markings for a bounded TPN could be based on the previous iterative process but, in some cases, it will lead to a non-terminating computation. Though the number of reachable markings is finite for a bounded TPN, the number of zones in which a marking is reachable is not necessarily finite as shown by the TPN \mathcal{T}_0 in Fig. 1.

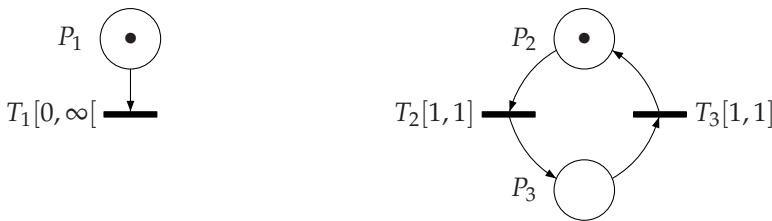


Figure 1. A TPN \mathcal{T}_0 with an Unbounded Number of Zones

The initial zone of \mathcal{T}_0 is Z_0 is $\{x_1 = 0 \wedge x_2 = 0 \wedge x_3 = 0\}$ (where x_i is the clock associated to T_i) and the initial marking $M_0 = (P_1, P_2, P_3) = (1, 1, 0)$. Consider the infinite firing sequence: $(T_2.T_3)^\omega$. By letting time pass, M_0 is reachable until $x_2 = 1$. When $x_2 = x_1 = 1$ the transition T_2 has to be fired. The zone corresponding to these clock values is: $Z_0 = \{0 \leq x_1 \leq 1 \wedge x_1 - x_2 = 0\}$. By firing T_2 and then T_3 , \mathcal{T}_0 reaches its initial marking M_0 . When it enters M_0 , the values of (new) clocks are: $x_1 = 2, x_2 = 0$ and $x_1 - x_2 = 2$. Indeed, T_1 remains enabled while T_2 and T_3 are fired and x_2 is reset when T_3 is fired because T_2 became newly enabled. Given these new values, the initial marking can exist while $x_2 \leq 1$ i.e., for the zone: $Z_1 = \{2 \leq x_1 \leq 3 \wedge x_1 - x_2 = 2\}$. By applying infinitely the sequence $T_2.T_3$, there exists an infinite number of zones for which the initial marking is reachable.

Actually, the number of zones is not bounded because infinity is used as latest firing time for T_1 . If for all the transitions t_i of a TPN, $\beta(t_i) \in \mathbb{Q}_{\geq 0}$, i.e., the upper bound is finite, we say that the TPN is *t-bounded*. If a TPN is *t-bounded*, all the clocks in the simulation graph are bounded and so, the number of different zones is bounded (Alur and Dill, 1994). The algorithm computing the simulation graph terminates in this case and it gives a finite (exact) representation of the state space of a bounded TPN.

We now present a more general algorithm which computes the state space of a TPN as defined in section 2, i.e., even if the TPN is not *t-bounded*. It is based on the use of an operator on zones which constructs equivalence classes. The resulting equivalence relation will be of finite index. A common operator on zones is the *k-approx* operator. For a given integer k , the use of this operator allows to create a finite set of distinct zones as presented in (Alur and Dill, 1994). To compute the simulation graph, we refine step 4 of the previous computation algorithm by applying the *k-approx* operator on the zone resulting from this last step.

This approximation is based on the fact that once the clock associated with an “unbounded” transition ($[\alpha, \infty]$) has reached the value α , its precise value does not matter. Using *k-approx* (with $k = \alpha$) allows to group all zones $[x, \infty[$, $x \geq \alpha$ in one equivalence class.

Previous papers on Timed Automata (Bouyer, 2004; Bouyer, 2003) have proved that this operator generally leads to a strict upper-approximation of the reachable state space. Nevertheless, for a given class of TA called *diagonal-free* TA, there is no upper-approximation of the reachable markings (Bouyer, 2004; Bouyer, 2003), and this also holds for TPNs:

Theorem 4 *For a bounded TPN, the (forward) algorithm to compute the simulation graph using k-approx on zones is exact (with respect to marking reachability) and terminates.*

In other words, checking whether $M \in \text{ReachMark}(\mathcal{T})$ is equivalent to checking whether there is a state class $C = (M, D)$ in the simulation graph. As the approximation is only needed for TPNs where some transitions have infinity as latest firing time, the following corollary holds:

Corollary 1 *For a bounded and t-bounded TPN, the (forward) algorithm to compute the simulation graph using zones is exact (with respect to marking reachability) and terminates.*

4. Comparison of Time Petri Nets and Timed Automata

In this section we give some results concerning the expressive power of TPNs and *Timed automata* (Alur and Dill, 1994). Timed automata (TA) are very similar to TPNs and a lot of theoretical results have been obtained for TA. Moreover efficient tools have been developed to check real-time properties on this model. It is thus important to compare the two formalisms and see if they can provide new insights for TPNs.

4.1. Timed Automata and Products of Timed Automata

Timed automata were studied by ALUR and DILL (Alur and Dill, 1994) and are used to model systems which combine *discrete* and *continuous* evolutions.

Definition 10 (Timed Automaton) *A Timed Automaton H is a tuple $(N, l_0, C, A, E, \text{Inv})$ where:*

- N is a finite set of locations;
- $l_0 \in N$ is the initial location;
- X is a finite set of positive real-valued clocks;
- A is a finite set of actions;

- $E \subseteq N \times \mathcal{C}(C) \times A \times 2^X \times N$ is a finite set of edges, $e = \langle l, \gamma, a, R, l' \rangle \in E$ represents an edge from the location l to the location l' with the guard γ , the label a and the reset set $R \subseteq X$;
- $Inv \in \mathcal{C}(X)^N$ assigns an invariant to any location. We restrict the invariants to conjuncts of terms of the form $c \leq r$ for $c \in C$ and $r \in \mathbb{N}$.

The semantics of a timed automaton is a timed transition system.

Definition 11 (Semantics of a Timed Automaton) *The semantics of a timed automaton $H = (N, l_0, X, A, E, Inv)$ is given by a timed transition system $S_H = (Q, q_0, \rightarrow)$ with $Q = N \times \mathbb{R}_{\leq 0}^X$, $q_0 = (l_0, \mathbf{0})$ is the initial state and \rightarrow consists of the discrete and continuous transition relations:*

- the discrete transition relation is defined for all $a \in A$ by $(l, v) \xrightarrow{a} (l', v')$ if:

$$\exists (l, \gamma, a, R, l') \in E \text{ s.t. } \begin{cases} \gamma(v) = \mathbf{tt}, \\ v' = v[R \mapsto 0] \\ Inv(l')(v') = \mathbf{tt} \end{cases}$$

- the continuous transitions is defined for all $t \in \mathbb{R}_{\geq 0}$ by $(l, v) \xrightarrow{t} (l', v')$ if:

$$\begin{cases} l = l' & v' = v + t \text{ and} \\ \forall 0 \leq t' \leq t, Inv(l)(v + t') = \mathbf{tt} \end{cases}$$

A run of a timed automaton H is an initial run in S_H starting in q_0 . The set of runs of H is denoted by $Runs(H)$. If we add two sets of locations $F \subseteq N$ and $R \subseteq N$ we can define the timed languages accepted by a TA H . We let $\mathcal{L}(H)$, $\mathcal{L}^*(H)$ and $\mathcal{L}^\omega(H)$ be the different timed languages accepted by H .

Modularity is important for modeling systems and it is convenient to describe a system as a parallel composition of timed automata. To this end, we use the classical composition notion based on a *synchronization function* à la Arnold-Nivat. Let $X = \{x_1, \dots, x_n\}$ be a set of clocks, H_1, \dots, H_n be n timed automata with $H_i = (N_i, l_{i,0}, X, A, E_i, Inv_i)$. 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 $(H_1 | \dots | H_n)_f$ the parallel composition of the H_i 's w.r.t. f . The configurations of $(H_1 | \dots | H_n)_f$ are pairs (\mathbf{l}, \mathbf{v}) with $\mathbf{l} = (l_1, \dots, l_n) \in N_1 \times \dots \times N_n$ and $\mathbf{v} = (v_1, \dots, v_n)$ where each v_i is the value of the clock $x_i \in X$. Then the semantics of a synchronized product of timed automata is also a timed transition system: the synchronized product can do a discrete transition if all the components agree to do so, and time can progress in the synchronized product also if all the components agree to do so. This is formalized by the following definition:

Definition 12 (Semantics of a Product of Timed Automata) *Let H_1, \dots, H_n be timed automata with $H_i = (N_i, l_{i,0}, X, A, E_i, Inv_i)$, and f a (partial) synchronization function $(A \cup \{\bullet\})^n \hookrightarrow A$. The semantics of $(H_1 | \dots | H_n)_f$ is a timed transition system $S = (Q, q_0, A, \rightarrow)$ with $Q = N_1 \times \dots \times N_n \times \mathbb{R}_{\geq 0}^X$, q_0 is the initial state $((l_{1,0}, \dots, l_{n,0}), \mathbf{0})$ and \rightarrow is defined by:*

- $(\mathbf{l}, \mathbf{v}) \xrightarrow{b} (l', v')$ if there exists $(a_1, \dots, a_n) \in (A \cup \{\bullet\})^n$ s.t. $f(a_1, \dots, a_n) = b$ and for any i we have:

- . If $a_i = \bullet$, then $\mathbf{l}'[i] = \mathbf{l}[i]$ and $\mathbf{v}'[i] = \mathbf{v}[i]$,
 - . If $a_i \in A$, then $(\mathbf{l}[i], \mathbf{v}[i]) \xrightarrow{a_i} (\mathbf{l}'[i], \mathbf{v}'[i])$.
- $(\mathbf{l}, \mathbf{v}) \xrightarrow{t} (\mathbf{l}', \mathbf{v}')$ if for all $i \in [1..n]$, every H_i agrees on time elapsing i.e., $(\mathbf{l}[i], \mathbf{v}[i]) \xrightarrow{t} (\mathbf{l}'[i], \mathbf{v}'[i])$.

We could equivalently define the product of n timed automata syntactically, building a new timed automaton from the n initial ones. In the sequel, we consider a product $(H_1 | \dots | H_n)_f$ to be a timed automaton the semantics of which is timed bisimilar to the semantics of the product we have given in Definition 12.

4.2. Expressiveness of TA vs TPNs

In this subsection, we define some criteria to compare the expressive power of TA and TPNs. We then show how to translate a TPN into an equivalent TA.

4.2.1. Expressiveness and Equivalence Problems

If B, B' are either TPNs or TA, we write $B \approx_S B'$ (resp. $B \approx_W B'$) for $S_B \approx_S S_{B'}$ (resp. $S_B \approx_W S_{B'}$). Let \mathcal{C} and \mathcal{C}' be two classes of TPNs or TA.

Definition 13 (Expressiveness w.r.t. Timed Language Acceptance) *The class \mathcal{C} is more expressive than \mathcal{C}' w.r.t. timed language acceptance if for all $B' \in \mathcal{C}'$ there is a $B \in \mathcal{C}$ s.t. $\mathcal{L}(B) = \mathcal{L}(B')$. We write $\mathcal{C}' \leq_{\mathcal{L}} \mathcal{C}$ in this case. If moreover there is some $B \in \mathcal{C}$ s.t. there is no $B' \in \mathcal{C}'$ with $\mathcal{L}(B) = \mathcal{L}(B')$, then $\mathcal{C}' <_{\mathcal{L}} \mathcal{C}$ (read “strictly more expressive”). If both $\mathcal{C}' \leq_{\mathcal{L}} \mathcal{C}$ and $\mathcal{C} \leq_{\mathcal{L}} \mathcal{C}'$ then \mathcal{C} and \mathcal{C}' are equally expressive w.r.t. timed language acceptance, and we write $\mathcal{C} =_{\mathcal{L}} \mathcal{C}'$.*

Definition 14 (Expressiveness w.r.t. Timed Bisimilarity) *The class \mathcal{C} is more expressive than \mathcal{C}' w.r.t. strong (resp. weak) timed bisimilarity if for all $B' \in \mathcal{C}'$ there is a $B \in \mathcal{C}$ s.t. $B \approx_S B'$ (resp. $B \approx_W B'$). We write $\mathcal{C}' \leq_S \mathcal{C}$ (resp. $\mathcal{C}' \leq_W \mathcal{C}$) in this case. If moreover there is a $B \in \mathcal{C}$ s.t. there is no $B' \in \mathcal{C}'$ with $B \approx_S B'$ (resp. $B \approx_W B'$), then $\mathcal{C}' <_S \mathcal{C}$ (resp. $\mathcal{C}' <_W \mathcal{C}$). If both $\mathcal{C}' <_S \mathcal{C}$ and $\mathcal{C} <_S \mathcal{C}'$ (resp. $<_W$) then \mathcal{C} and \mathcal{C}' are equally expressive w.r.t. strong (resp. weak) timed bisimilarity, and we write $\mathcal{C} \approx_S \mathcal{C}'$ (resp. $\mathcal{C} \approx_W \mathcal{C}'$).*

In the sequel we will compare various classes of TPNs and TAs. When referring to language acceptance we assume that two sets F and R have been given for a TPN (see Definition 7) and for a TA. We use the following notations:

- $\text{B-TPN}_{\varepsilon}$ for the set of bounded labeled TPNs with ε -transitions (Definition 7);
- $\text{1-B-TPN}_{\varepsilon}$ for the subset of $\text{B-TPN}_{\varepsilon}$ with at most one token in each place (one safe TPN);
- $\text{B-TPN}(\leq, \geq)$ for the subset of $\text{B-TPN}_{\varepsilon}$ where only closed intervals are used;
- $\mathcal{TA}_{\varepsilon}$ for TA with ε -transitions; $\mathcal{TA}_{(\leq, \geq)}^*$ for the syntactical subclass of TA that is equivalent to $\text{B-TPN}(\leq, \geq)$ (see (Bérard et al., 2005a)).

$\mathcal{TA}_{(\leq, \geq)}^*$ is formally defined by:

Definition 15 *The subclass $\mathcal{TA}_{(\leq, \geq)}^*$ of TA is defined by the set of TA of the form (L, I_0, X, A, E, Inv) where :*

	Timed Language Acceptance	Timed Bisimilarity
B- $\mathcal{TPN}_\varepsilon$	$\leq_{\mathcal{L}} \mathcal{TA}_\varepsilon$ (Cassez and Roux, 2006)	$\leq_{\mathcal{W}} \mathcal{TA}_\varepsilon$ (Cassez and Roux, 2006)
	$=_{\mathcal{L}} \mathcal{TA}_\varepsilon$ (Bérard et al., 2005a)	$<_{\mathcal{W}} \mathcal{TA}_\varepsilon$ (Bérard et al., 2005a)
B- $\mathcal{TPN}(\leq, \geq)$	$=_{\mathcal{L}} \mathcal{TA}_{(\leq, \geq)}^*$ (Bérard et al., 2005b)	$\approx_{\mathcal{W}} 1\text{-B-}\mathcal{TPN}(\leq, \geq)$ (Bérard et al., 2005b) $\approx_{\mathcal{W}} \mathcal{TA}_{(\leq, \geq)}^*$ (Bérard et al., 2005b)

Table 1. Summary of the Expressiveness Results for TPNs vs. TA

- guards are conjunctions of atomic constraints of the form $x \geq c$ and invariants are conjunction of atomic constraints $x \leq c$.
- the invariants satisfy the following property; $\forall e = (\ell, \gamma, a, R, \ell') \in E$, if $x \notin R$ and $x \leq c$ is an atomic constraint in $\text{Inv}(\ell)$, then if $x \leq c'$ is $\text{Inv}(\ell')$ for some c' then $c' \geq c$.

In Table 1, $\preceq_{\mathcal{L}}$ or $\preceq_{\mathcal{W}}$ with $\preceq \in \{<, \leq\}$, respectively means “less expressive than” w.r.t. Timed Language Acceptance and Weak Timed Bisimilarity; the term $=_{\mathcal{L}}$ means “equally expressive as” w.r.t. language acceptance and $\approx_{\mathcal{W}}$ “equally expressive as” w.r.t. weak timed bisimilarity. A consequence of the results in this table is that 1-B- $\mathcal{TPN}_\varepsilon$ and B- $\mathcal{TPN}_\varepsilon$ are equally expressive w.r.t. Timed Language Acceptance i.e., 1-B- $\mathcal{TPN}_\varepsilon =_{\mathcal{L}} \text{B-}\mathcal{TPN}_\varepsilon$. An equivalent result was known for untimed PN (we can always obtain a safe PN that accepts the same language as a PN) but the counterpart for TPN was proved in (Bérard et al., 2005a).

Surprisingly, bounded TPNs are less expressive than timed automata w.r.t. timed bisimulation. We will see in subsection 4.3 how to translate a TPN into a timed bisimilar TA. Thanks to this translation, we will use in section 5 the TA obtained to check TCTL properties of the original TPN.

4.3. From Time Petri Nets to Timed Automata

The relationship between TPNs and TA has not been much investigated before 2000. In (Sifakis and Yovine, 1996) J. SIFAKIS and S. YOVINE are mainly concerned with *compositionality* problems. They show that for a subclass of 1-safe Time Stream Petri Nets, 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 propose Timed Automata with Deadlines and flexible notions of composition. 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 safe TPN into TAD (that can be considered as standard timed automata). In (Cortès et al., 2000) the authors consider an extension of Time Petri Nets (*PRES+*) and propose a translation into hybrid automata. Correctness of the translation is not proved. Moreover the method is defined only for 1-safe nets.

In another line of work, SAVA (Sava, 2001) considers bounded TPNs where the underlying Petri net is not necessarily safe and proposes an algorithm to translate the TPN into a timed automaton (one clock is needed for each transition of the original TPN). However, the author does not give any proof that this translation is correct (i.e., it preserves some equivalence relation between the semantics of the original TPN and the computed TA) and neither that the algorithm terminates (even if the TPN is bounded).

LIME and ROUX proposed an extension in (Lime and Roux, 2006) of the state class graph construction that allows to build the state class graph of a bounded TPN as a timed automaton. They prove that this timed automaton and the TPN are timed bisimilar and they also prove a relative minimality result of the number of clocks needed in the obtained automaton.

The first two approaches are structural but are limited to Petri nets whose underlying net is 1-safe. The last two approaches rely on the computation of the state space of the TPN and are limited to bounded TPNs. In this section, we consider a structural translation from TPN (not necessary bounded) to TA proposed in (Cassez and Roux, 2006). This extends the previous results in the following directions: first, we can easily prove that our translation is correct and terminates as it is a syntactic translation and it produces a timed automaton that is timed bisimilar to the TPN we started with. Notice that the timed automaton contains integer variables that correspond to the marking of the Petri net and that it may have an unbounded number of locations. However timed bisimilarity holds even in the unbounded case. In case the Petri net is bounded, we obtain a timed automaton with a finite number of locations and we can check for TCTL properties of the original TPN. Second, as it is a structural translation it does not need expensive computation (like the State Class Graph) to obtain a timed automaton. This has a practical application as it enables one to use efficient existing tools for TA to analyze TPNs.

4.3.1. Translating Time Petri Nets into Timed Automata

In this subsection, we build a synchronized product of timed automata from a TPN so that the behaviors of the two are in a one-to-one correspondence.

We start with a TPN $\mathcal{T} = (P, T, \bullet(\cdot), (\cdot)\bullet, M_0, (\alpha, \beta))$ with set of places $P = \{p_1, \dots, p_m\}$ and set of transitions $T = \{t_1, \dots, t_n\}$.

Timed Automaton Associated with a Transition

We define one timed automaton \mathcal{A}_i for each transition t_i of T (see Fig. 2.a). This timed automaton has one clock x_i . Also the locations of the automaton \mathcal{A}_i give the state of the transition t_i : in location t the transition is enabled; in location \bar{t} it is disabled and in *Firing* it is being fired. The initial location of each \mathcal{A}_i depends on the initial marking M_0 of the Petri net we want to translate. If $M_0 \geq \bullet t_i$, then the initial location is t otherwise it is \bar{t} . This automaton updates an array of integers \mathbf{p} (s.t. $\mathbf{p}[i]$ is the number of tokens in place p_i) shared by all the \mathcal{A}_i 's. This is not covered by Definition 12, but this extended model with integer arrays is very common (Pettersson and Larsen, 2000) and it does not affect the expressiveness of the model when the variables are bounded.

The Supervisor

The automaton for the supervisor SU is depicted on Fig. 2.b. The locations 1 to 3 subscripted with a "c" are assumed to be committed. *Committed* locations can be simulated by adding an extra variable: see (Tripakis, 1999) Appendix A for details. This means that no time can elapse while visiting them. We denote by $\Delta(\mathcal{T}) = (SU \mid \mathcal{A}_1 \mid \dots \mid \mathcal{A}_n)_f$ the timed automaton associated to the TPN \mathcal{T} . The initial location of the supervisor is 0. Let us define the synchronization function f with $n + 1$ parameters (the first element of the vector refers to the supervisor move) by:

- $f(!pre, \bullet, \dots, ?pre, \bullet, \dots) = pre_i$ if $?pre$ is the $(i + 1)$ th argument and all the other arguments are \bullet ,
- $f(!post, \bullet, \dots, ?post, \bullet, \dots) = post_i$ if $?post$ is the $(i + 1)$ th argument and all the other arguments are \bullet ,
- $f(!update, ?update, \dots, ?update) = update$.

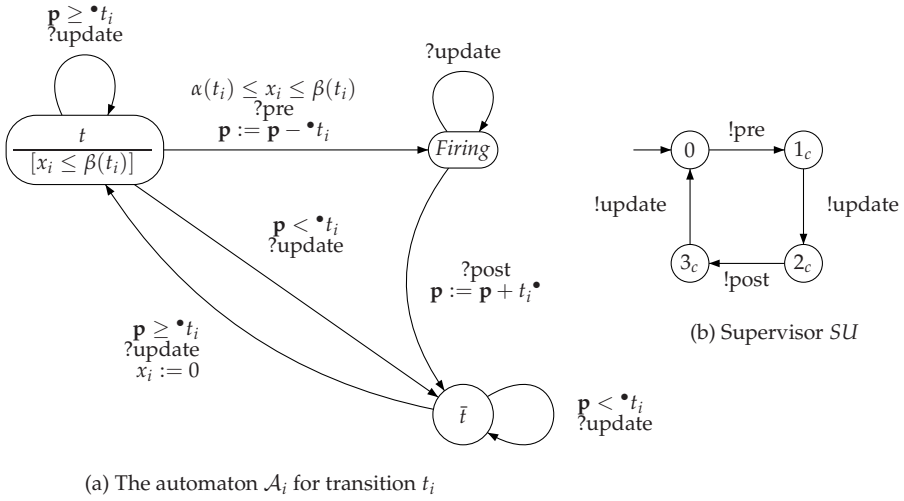


Figure 2. Automata for the Transitions and the Supervisor

In the sequel, $((s, \mathbf{p}), \mathbf{q}, \mathbf{v})$ is such that $(s, \mathbf{p}) \in \{0, 1_c, 2_c, 3_c\} \times \mathbb{N}^m$ is the state of SU , \mathbf{q} gives the product location of $\mathcal{A}_1 \times \cdots \times \mathcal{A}_n$, and $\mathbf{v}[i], i \in [1..n]$ gives the value of the clock x_i . We will prove in the next subsection that the semantics of $\Delta(\mathcal{T})$ is closely related to the semantics of \mathcal{T} . For this we have to relate the states of \mathcal{T} to the states of $\Delta(\mathcal{T})$ and we define the following equivalence:

Definition 16 (State Equivalence) Let (M, ν) and $((s, \mathbf{p}), \mathbf{q}, \mathbf{v})$ be, respectively, a state of $S_{\mathcal{T}}$ and a configuration. Then $(M, \nu) \approx ((s, \mathbf{p}), \mathbf{q}, \mathbf{v})$ if:

$$\begin{cases} s = 0, \\ \forall i \in [1..m], \quad \mathbf{p}[i] = M(p_i), \\ \forall k \in [1..n], \quad \mathbf{q}[k] = \begin{cases} t & \text{if } M \geq \bullet t_k, \\ \bar{i} & \text{otherwise} \end{cases} \\ \forall k \in [1..n], \quad \mathbf{v}[k] = \nu_k. \end{cases}$$

4.3.2. Soundness of the Translation

We now prove that our translation preserves the behaviors of the initial TPN in the sense that the semantics of the TPN and its translation are timed bisimilar. Let \mathcal{T} be a TPN and $S_{\mathcal{T}} = (Q, q_0, T, \rightarrow)$ its semantics. Let \mathcal{A}_i be the automaton associated with transition t_i of \mathcal{T} as described by Fig. 2.a, SU the supervisor automaton of Fig. 2.b and f the synchronization function defined previously. The semantics of $\Delta(\mathcal{T}) = (SU \mid \mathcal{A}_1 \mid \cdots \mid \mathcal{A}_n)_f$ is the TTS $S_{\Delta(\mathcal{T})} = (Q_{\Delta(\mathcal{T})}, q_0^{\Delta(\mathcal{T})}, A(\Delta(\mathcal{T})), \rightarrow)$.

Theorem 5 (Timed Bisimilarity) For $(M, v) \in S_{\mathcal{T}}$ and $((0, \mathbf{p}), \mathbf{q}, \mathbf{v}) \in S_{\Delta(\mathcal{T})}$ such that $(M, v) \approx ((0, \mathbf{p}), \mathbf{q}, \mathbf{v})$ the following holds:

$$(M, v) \xrightarrow{t_i} (M', v') \quad \text{iff} \quad \begin{cases} ((0, \mathbf{p}), \mathbf{q}, \mathbf{v}) \xrightarrow{w_i} ((0, \mathbf{p}'), \mathbf{q}', \mathbf{v}') \text{ with} \\ w_i = \text{pre}_i.\text{update}.\text{post}_i.\text{update} \text{ and} \\ (M', v') \approx ((0, \mathbf{p}'), \mathbf{q}', \mathbf{v}') \end{cases} \quad (2)$$

$$(M, v) \xrightarrow{d} (M', v') \quad \text{iff} \quad \begin{cases} ((0, \mathbf{p}), \mathbf{q}, \mathbf{v}) \xrightarrow{d} ((0, \mathbf{p}'), \mathbf{q}', \mathbf{v}') \text{ and} \\ (M', v') \approx ((0, \mathbf{p}'), \mathbf{q}', \mathbf{v}') \end{cases} \quad (3)$$

Proof. We first prove statement (2). Assume $(M, v) \approx ((0, \mathbf{p}), \mathbf{q}, \mathbf{v})$. Then as t_i can be fired from (M, v) we have: (i) $M \geq \bullet t_i$, (ii) $\alpha(t_i) \leq v_i \leq \beta(t_i)$, (iii) $M' = M - \bullet t_i + t_i \bullet$, and (iv) $v'_k = 0$ if $\uparrow \text{enabled}(t_k, M, t_i)$ and $v'_k = v_k$ otherwise. From (i) and (ii) and the state equivalence we deduce that $\mathbf{q}[i] = t$ and $\alpha(t_i) \leq \mathbf{v}[i] \leq \beta(t_i)$. Hence ?pre is enabled in \mathcal{A}_i . In state 0 for the supervisor, !pre is the only possible transition. As the synchronization function f allows $(\text{!pre}, \bullet, \dots, \text{?pre}, \dots, \bullet)$ the global action pre_i is possible. After this move $\Delta(\mathcal{T})$ reaches state $((1, \mathbf{p}_1), \mathbf{q}_1, \mathbf{v}_1)$ such that for all $k \in [1..n]$, $\mathbf{q}_1[k] = \mathbf{q}[k], \forall k \neq i$ and $\mathbf{q}_1[i] = \text{Firing}$. Also $\mathbf{p}_1 = \mathbf{p} - \bullet t_i$ and $\mathbf{v}_1 = \mathbf{v}$.

Now the only possible transition when the supervisor is in state 1 is an update transition where all the \mathcal{A}_i 's synchronize according to f . From $((1, \mathbf{p}_1), \mathbf{q}_1, \mathbf{v}_1)$ we reach $((2, \mathbf{p}_2), \mathbf{q}_2, \mathbf{v}_2)$ with $\mathbf{p}_2 = \mathbf{p}_1, \mathbf{v}_2 = \mathbf{v}_1$. For all $k \in [1..n], k \neq i$, $\mathbf{q}_2[k] = t$ if $\mathbf{p}_1 \geq \bullet t_k$ and $\mathbf{q}_2[k] = \bar{t}$ otherwise. Also $\mathbf{q}_2[i] = \text{Firing}$. The next global transition must be a post_i transition leading to $((3, \mathbf{p}_3), \mathbf{q}_3, \mathbf{v}_3)$ with $\mathbf{p}_3 = \mathbf{p}_2 + t_i \bullet, \mathbf{v}_3 = \mathbf{v}_2$ and for all $k \in [1..n]$, $\mathbf{q}_3[k] = \mathbf{q}_2[k], \forall k \neq i$ and $\mathbf{q}_3[i] = \bar{t}$. From this last state only an update transition leading to $((0, \mathbf{p}_4), \mathbf{q}_4, \mathbf{v}_4)$ is allowed, with $\mathbf{p}_4 = \mathbf{p}_3, \mathbf{v}_4$ and \mathbf{q}_4 given by: for all $k \in [1..n]$, $\mathbf{q}_4[k] = t$ if $\mathbf{p}_3 \geq \bullet t_k$ and \bar{t} otherwise. $\mathbf{v}_4[k] = 0$ if $\mathbf{q}_3[k] = \bar{t}$ and $\mathbf{q}_4[k] = t$ and $\mathbf{v}_4[k] = \mathbf{v}_1[k]$ otherwise. We then just notice that $\mathbf{q}_3[k] = \bar{t}$ iff $\mathbf{p} - \bullet t_i < \bullet t_k$ and $\mathbf{q}_4[k] = t$ iff $\mathbf{p} - \bullet t_i + t_i \bullet \geq \bullet t_k$. This entails that $\mathbf{v}_4[k] = 0$ iff $\uparrow \text{enabled}(t_k, \mathbf{p}, t_i)$ and with (iv) gives $v'_k = \mathbf{v}_4[k]$. As $\mathbf{p}_4 = \mathbf{p}_3 = \mathbf{p}_2 + t_i \bullet = \mathbf{p}_1 - \bullet t_i + t_i \bullet = \mathbf{p} - \bullet t_i + t_i \bullet$ using (iii) we have $\forall i \in [1..m], M'(p_i) = \mathbf{p}_4[i]$. Hence we conclude that $((0, \mathbf{p}_4), \mathbf{q}_4, \mathbf{v}_4) \approx (M', v')$.

The converse of statement (2) is straightforward following the same steps as the previous ones. We now focus on statement (3). According to the semantics of TPNs, a continuous transition $(M, v) \xrightarrow{d} (M', v')$ is allowed iff $v = v' + d$ and $\forall k \in [1..n], (M \geq \bullet t_k \implies v'_k \leq \beta(t_k))$. As $(M, v) \approx ((0, \mathbf{p}), \mathbf{q}, \mathbf{v})$, if $M \geq \bullet t_k$ then $\mathbf{q}[k] = t$ and the continuous evolution for \mathcal{A}_k is constrained by the invariant $x_k \leq \beta(t_k)$. Otherwise $\mathbf{q}[k] = \bar{t}$ and the continuous evolution is unconstrained for \mathcal{A}_k . No constraints apply for the supervisor in state 0. Hence the result. \square

We can now state a useful corollary which enables us to do TCTL model-checking for TPNs in the next section. We write $\Delta((M, v)) = ((0, \mathbf{p}), \mathbf{q}, \mathbf{v})$ if $(M, v) \approx ((0, \mathbf{p}), \mathbf{q}, \mathbf{v})$, $\Delta(t_i) = \text{pre}_i.\text{update}.\text{post}_i.\text{update}$ and also $\Delta(d) = d$. Just notice that Δ is one-to-one and we can use Δ^{-1} as well. Then we extend Δ to transitions by: $\Delta((M, v) \xrightarrow{e} (M', v')) = \Delta((M, v)) \xrightarrow{\Delta(e)} \Delta((M', v'))$ with $e \in T \cup \mathbb{R}_{\geq 0}$ (as $\Delta(t_i)$ is a word, this transition is a four step transition in $\Delta(\mathcal{T})$). Again we can extend Δ to runs: if $\rho \in \text{Runs}(\mathcal{T})$ we denote $\Delta(\rho)$ the associated run in $\text{Runs}(\Delta(\mathcal{T}))$. Notice that Δ^{-1} is only defined for runs σ of $\text{Runs}(\Delta(\mathcal{T}))$, the last state of which is of the form $((0, \mathbf{p}), \mathbf{q}, \mathbf{v})$ where the supervisor is in state 0. We denote this property $\text{last}(\sigma) \models \text{SU.0}$.

Corollary 2 $(\rho \in \text{Runs}(\mathcal{T}) \wedge \sigma = \Delta(\rho))$ iff $(\sigma \in \text{Runs}(\Delta(\mathcal{T})) \wedge \text{last}(\sigma) \models \text{SU.0})$.

Proof. The proof is a direct consequence of Theorem 5. It suffices to notice that all the finite runs of $\Delta(\mathcal{T})$ are of the form

$$\sigma = (s_0, v_0) \xrightarrow{\delta_1} (s'_0, v'_0) \xrightarrow{w_1} (s_1, v_1) \cdots \xrightarrow{\delta_n} (s'_{n-1}, v'_{n-1}) \xrightarrow{w_n} (s_n, v_n)$$

with $w_i = \text{pre}_i.\text{update}.\text{post}_i.\text{update}$, $\delta_i \in \mathbb{R}_{\geq 0}$, and using Theorem 5, if $\text{last}(\sigma) \models \text{SU}.0$, there exists a corresponding run ρ in \mathcal{T} s.t. $\sigma = \Delta(\rho)$. \square

This property will be used in Section 5 when we address the problem of model-checking TCTL for TPNs.

5. Model-Checking of TCTL on Time Petri Nets

In this section we introduce a logic to specify properties of real-time systems and show how we can model-check this logic on bounded TPNs.

We define TCTL (Henzinger et al., 1994) for TPNs. The only difference with the versions of (Henzinger et al., 1994) is that the atomic propositions usually associated with states are now properties of markings. For practical applications with model-checkers, we assume that the TPNs we check are bounded.

Definition 17 (TCTL for TPN) Assume we have a TPN with n places, and m transitions $T = \{t_1, t_2, \dots, t_m\}$. The temporal logic TPN-TCTL is inductively defined by:

$$\text{TPN-TCTL} ::= \mathbf{M} \bowtie \bar{V} \mid \mathbf{false} \mid t_k + c \leq t_j + d \mid \neg\varphi \mid \varphi \rightarrow \psi \mid \varphi \exists \mathcal{U}_{\bowtie c} \psi \mid \varphi \forall \mathcal{U}_{\bowtie c} \psi \quad (4)$$

where \mathbf{M} and \mathbf{false} are keywords, $\varphi, \psi \in \text{TPN-TCTL}$, $t_k, t_j \in T$, $c, d \in \mathbb{Z}$, $\bar{V} \in (\mathbb{N} \cup \{\infty\})^n$ and⁶ $\bowtie \in \{<, \leq, =, >, \geq\}$.

Intuitively the meaning of $\mathbf{M} \bowtie \bar{V}$ is that the current marking vector is in relation \bowtie with \bar{V} . The meaning of the other operators is the usual one. We use the familiar shorthands:

$$\begin{aligned} \mathbf{true} &= \neg\mathbf{false} \\ \exists \diamond_{\bowtie c} \phi &= \mathbf{true} \exists \mathcal{U}_{\bowtie c} \phi \\ \forall \square_{\bowtie c} &= \neg \exists \diamond_{\bowtie c} \neg \phi. \end{aligned}$$

The semantics of TPN-TCTL is defined on timed transition systems. Let $\mathcal{T} = (P, T, \bullet(\cdot), (\cdot)^\bullet, M_0, (\alpha, \beta))$ be a TPN with n places and m transitions and $S_{\mathcal{T}} = (Q, q_0, T, \rightarrow)$ the semantics of \mathcal{T} . Let $\sigma = (s_0, v_0) \xrightarrow{a_1} \cdots \xrightarrow{a_n} (s_n, v_n) \in \text{Runs}(\mathcal{T})$. The truth value of a formula φ of TPN-TCTL for a state (M, v) is given in Fig. 3.

The TPN \mathcal{T} satisfies the formula φ of TPN-TCTL, which is denoted by $\mathcal{T} \models \varphi$, iff the first state of $S_{\mathcal{T}}$ satisfies φ , i.e., $(M_0, \mathbf{0}) \models \varphi$.

We will see that thanks to Corollary 2, model-checking TPNs amounts to model-checking timed automata.

Let us assume we have to model-check formula φ on a TPN \mathcal{T} . Our method consists in using the equivalent timed automaton $\Delta(\mathcal{T})$ defined in Section 4.3. For instance, suppose we want to check $\mathcal{T} \models \forall \square_{\leq 3} (\mathbf{M} \geq (1, 2))$. The check means that all the states reached within the next

⁶The use of ∞ in \bar{V} allows us to handle comparisons like $M(p_1) \leq 2 \wedge M(p_2) \geq 3$ by writing $\mathbf{M} \leq (2, \infty) \wedge \mathbf{M} \geq (0, 3)$.

$(M, \nu) \models \mathbf{M} \bowtie \bar{V}$	iff	$M \bowtie \bar{V}$
$(M, \nu) \not\models \mathbf{false}$		
$(M, \nu) \models t_k + c \leq t_j + d$	iff	$\nu_k + c \leq \nu_j + d$
$(M, \nu) \models \neg \varphi$	iff	$(M, \nu) \not\models \varphi$
$(M, \nu) \models \varphi \rightarrow \psi$	iff	$(M, \nu) \models \varphi$ implies $(M, \nu) \models \psi$
$(M, \nu) \models \varphi \exists \mathcal{U}_{\bowtie c} \psi$	iff	$\exists \sigma \in \text{Runs}(\mathcal{T})$ such that:
		$\left\{ \begin{array}{l} (s_0, \nu_0) = (M, \nu) \\ \forall i \in [1..n], \forall d \in [0, d_i], (s_i, \nu_i + d) \models \varphi \\ (\sum_{i=1}^n d_i) \bowtie c \text{ and } (s_n, \nu_n) \models \psi \end{array} \right.$
$(M, \nu) \models \varphi \forall \mathcal{U}_{\bowtie c} \psi$	iff	$\forall \sigma \in \text{Runs}(\mathcal{T})$ we have:
		$\left\{ \begin{array}{l} (s_0, \nu_0) = (M, \nu) \\ \forall i \in [1..n], \forall d \in [0, d_i], (s_i, \nu_i + d) \models \varphi \\ (\sum_{i=1}^n d_i) \bowtie c \text{ and } (s_n, \nu_n) \models \psi \end{array} \right.$

Figure 3. Semantics of TPN-TCTL

3 time units will have a marking such that p_1 has more than one token and p_2 more than 2. Actually, this is equivalent to checking

$$\forall \square_{\leq 3}(SU.0 \rightarrow (\mathbf{p}[1] \geq 1 \wedge \mathbf{p}[2] \geq 2))$$

on the equivalent timed automaton. Notice that $\exists \diamond_{\leq 3}(\mathbf{M} \geq (1, 2))$ reduces to

$$\exists \diamond_{\leq 3}(SU.0 \wedge (\mathbf{p}[1] \geq 1 \wedge \mathbf{p}[2] \geq 2))$$

We can then define the translation of a formula in TPN-TCTL to standard TCTL for timed automata: we denote TA-TCTL the logic TCTL for timed automata.

Definition 18 (From TPN-TCTL to TA-TCTL) Let φ be a formula of TPN-TCTL. Then the translation $\Delta(\varphi)$ of φ is inductively defined by:

$$\begin{aligned} \Delta(\mathbf{M} \bowtie \bar{V}) &= \bigwedge_{i=1}^n (\mathbf{p}[i] \bowtie \bar{V}_i) \\ \Delta(\mathbf{false}) &= \mathbf{false} \\ \Delta(t_k + c \bowtie t_j + d) &= x_k + c \bowtie x_j + d \\ \Delta(\neg \varphi) &= \neg \Delta(\varphi) \\ \Delta(\varphi \rightarrow \psi) &= SU.0 \wedge (\Delta(\varphi) \rightarrow \Delta(\psi)) \\ \Delta(\varphi \exists \mathcal{U}_{\bowtie c} \psi) &= (SU.0 \rightarrow \Delta(\varphi)) \exists \mathcal{U}_{\bowtie c} (SU.0 \wedge \Delta(\psi)) \\ \Delta(\varphi \forall \mathcal{U}_{\bowtie c} \psi) &= (SU.0 \rightarrow \Delta(\varphi)) \forall \mathcal{U}_{\bowtie c} (SU.0 \wedge \Delta(\psi)) \end{aligned}$$

$SU.0$ means that the supervisor is in state 0 and the clocks x_k are the ones associated with every transition t_k in the translation scheme.

Theorem 6 Let \mathcal{T} be a TPN and $\Delta(\mathcal{T})$ the equivalent timed automaton. Let (M, ν) be a state of $S_{\mathcal{T}}$ and $((s, \mathbf{p}), \mathbf{q}, \mathbf{v}) = \Delta((M, \nu))$ the equivalent state of $S_{\Delta(\mathcal{T})}$ (i.e. $(M, \nu) \approx ((s, \mathbf{p}), \mathbf{q}, \mathbf{v})$). Then $\forall \varphi \in \text{TPN-TCTL}$:

$$(M, \nu) \models \varphi \text{ iff } ((s, \mathbf{p}), \mathbf{q}, \mathbf{v}) \models \Delta(\varphi).$$

Proof. The proof is done by structural induction on the formula of TPN-TCTL. The cases of $\mathbf{M} \bowtie \bar{V}$, **false**, $t_k + c \leq t_j + d$, $\neg \varphi$ and $\varphi \rightarrow \psi$ are straightforward. We give the full proof for $\varphi \exists \mathcal{U}_{\bowtie c} \psi$ (the same proof can be carried out for $\varphi \forall \mathcal{U}_{\bowtie c} \psi$).

Only if part.

Assume $(M, \nu) \models \varphi \exists \mathcal{U}_{\bowtie c} \psi$. Then by definition, there is a run ρ in $\text{Runs}(\mathcal{T})$ s.t.:

$$\rho = (s_0, \nu_0) \xrightarrow{d_1} (s_1, \nu_1) \cdots \xrightarrow{d_n} (s_n, \nu_n)$$

and $(s_0, \nu_0) = (M, \nu)$, $\sum_{i=1}^n d_i \bowtie c$, $\forall i \in [1..n], \forall d \in [0, d_i], (s_i, \nu_i + d) \models \varphi$ and $(s_n, \nu_n) \models \psi$. With corollary 2, we conclude that there is a run $\sigma = \Delta(\rho)$ in $\text{Runs}(S_{\Delta(\mathcal{T})})$ s.t.

$$\sigma = ((l_0, p_0), \bar{q}_0, v_0) \Longrightarrow_{w_1}^{d_1} ((l_1, p_1), \bar{q}_1, v_1) \cdots \cdots \Longrightarrow_{w_n}^{d_n} ((l_n, p_n), \bar{q}_n, v_n)$$

and $\forall i \in [1..n], ((l_i, p_i), \bar{q}_i, v_i) \approx (s_i, \nu_i)$ (this entails that $l_i = 0$).

Since $(s_n, \nu_n) \approx ((l_n, p_n), \bar{q}_n, v_n)$, using the induction hypothesis on ψ , we can assume that $(s_n, \nu_n) \models \psi$ iff $((l_n, p_n), \bar{q}_n, v_n) \models \Delta(\psi)$ and thus we can conclude that $((l_n, p_n), \bar{q}_n, v_n) \models \Delta(\psi)$. Moreover as $l_n = 0$ we have $((l_n, p_n), \bar{q}_n, v_n) \models \text{SU.0} \wedge \Delta(\psi)$. It remains to prove that all intermediate states satisfy $\text{SU.0} \rightarrow \Delta(\varphi)$. Just notice that all the intermediate states in σ not satisfying SU.0 between $((l_i, p_i), \bar{q}_i, v_i)$ and $((l_{i+1}, p_{i+1}), \bar{q}_{i+1}, v_{i+1})$ satisfy $\text{SU.0} \rightarrow \Delta(\psi)$. Then we just need to prove that the intermediate states satisfying SU.0 , i.e., the states $((l_i, p_i), \bar{q}_i, v_i)$ satisfy $\Delta(\varphi)$. As for all $i \in [1..n]$, we have $((l_i, p_i), \bar{q}_i, v_i) \approx (s_i, \nu_i)$, with the induction hypothesis on φ , we have $\forall i \in [1..n], ((l_i, p_i), \bar{q}_i, v_i) \models \Delta(\varphi)$. Moreover, again applying theorem 5, we obtain for all $d \in [0, d_i]: ((l_i, p_i), \bar{q}_i, v_i + d) \approx (s_i, \nu_i + d)$; applying the induction hypothesis again we conclude that for all $d \in [0, d_i] ((l_i, p_i), \bar{q}_i, v_i + d) \models \Delta(\varphi)$. Hence $((l_0, p_0), \bar{q}_0, v_0) \models (\text{SU.0} \rightarrow \varphi) \exists \mathcal{U}_{\bowtie c} (\text{SU.0} \wedge \psi)$.

If part.

Assume $((l_0, p_0), \bar{q}_0, v_0) \models (\text{SU.0} \rightarrow \Delta(\varphi)) \exists \mathcal{U}_{\bowtie c} (\text{SU.0} \wedge \Delta(\psi))$. Then there is a run

$$\sigma = ((l_0, p_0), \bar{q}_0, v_0) \Longrightarrow_{w_1}^{d_1} ((l_1, p_1), \bar{q}_1, v_1) \cdots \cdots \Longrightarrow_{w_n}^{d_n} ((l_n, p_n), \bar{q}_n, v_n)$$

with $((l_n, p_n), \bar{q}_n, v_n) \models \text{SU.0} \wedge \Delta(\psi)$ and:

$$\forall i \in [1..n], \forall d \in [0, d_i], ((l_i, p_i), \bar{q}_i, v_i) \models (\text{SU.0} \rightarrow \Delta(\varphi))$$

As $((l_n, p_n), \bar{q}_n, v_n) \models \text{SU.0}$, we can use corollary 2 and we know there exists a run in $\text{Runs}(\mathcal{T})$

$$\rho = \Delta^{-1}(\sigma) = (s_0, \nu_0) \xrightarrow{d_1} (s_1, \nu_1) \cdots \xrightarrow{d_n} (s_n, \nu_n)$$

with $\forall i \in [1..n], ((l_i, p_i), \bar{q}_i, v_i) \approx (s_i, \nu_i)$. The induction hypothesis on $\text{SU.0} \wedge \Delta(\psi)$ and $((l_n, p_n), \bar{q}_n, v_n) \models \text{SU.0} \wedge \Delta(\psi)$ implies $(s_n, \nu_n) \models \psi$. For all the intermediate states of ρ we also apply the induction hypothesis: each $((l_i, p_i), \bar{q}_i, v_i)$ is equivalent to (s_i, ν_i) and all the states $(s_i, \nu_i + d), d \in [0, d_i]$ satisfy φ . Hence $(s_0, \nu_0) \models \varphi \exists \mathcal{U}_{\bowtie c} \psi$. \square

Theorem 6 enables to reduce the model-checking of a TPN-TCTL formula φ against a TPN \mathcal{T} i.e., the problem $\mathcal{T} \models \varphi$ to a model-checking of TCTL against TA:

Corollary 3 $\mathcal{T} \models \varphi \iff \Delta(\mathcal{T}) \models \Delta(\varphi)$.

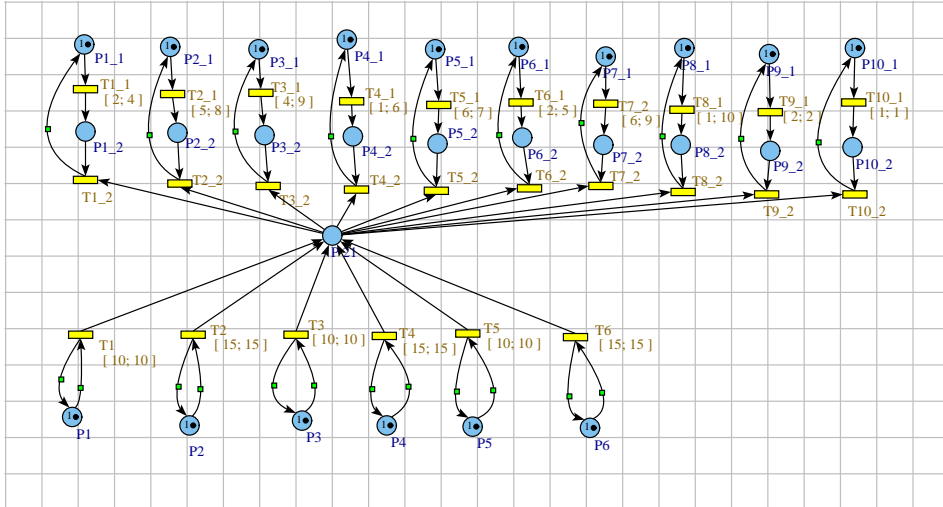


Figure 4. A TPN for a Producer/Consumer example in ROMEO

6. Implementation

In this section, we describe some properties of our translation and important implementation details. Then we report on examples we have checked using our approach and the tool UPPAAL.

6.1. Translation of TPNs to UPPAAL Input Format

The first step in using our approach is to translate an existing TPN into a product of TA. For this we use the TPN tool ROMEO (Gardey et al., 2005) that has been developed for the analysis of TPNs (state space computation and “on-the-fly” model-checking of reachability properties with a zone-based forward method and with the State Class Graph method). ROMEO has a GUI (see Fig. 4) to “draw” Time Petri Nets and an *export to UPPAAL* feature that implements our translation of a TPN into the equivalent TA in UPPAAL input format⁷.

The textual input format for TPNs in ROMEO is XML and the timed automaton is given in the “.xta” UPPAAL input format⁸. The translation gives one timed automaton for each transition and one automaton for the supervisor *SU* as described in Section 4.3. The automata for each transition update an array of integers $M[i]$ (which is the number of tokens⁹ in place i in the original TPN). For example, the enabledness and firing conditions of a transition t_i such that $\bullet t_i = (1, 0, 0)$ and $t_i^\bullet = (0, 0, 1)$, are respectively implemented by $M[0] \geq 1$ and $M[2] := M[2] + 1$. Instead of generating one *template automaton* for each transition, we generate as many templates as types of transitions in the original TPN: the type of a transition is

⁷At least version (3.4.7) of UPPAAL is required to read the files produced by ROMEO.

⁸see <http://www.uppaal.com> for further information about UPPAAL.

⁹The actual meaning of $M[i]$ is given by a table that is available in the ROMEO tool via the “Translate/Indices \implies Place/Transition” menu; the table gives the name of the place represented by $M[i]$ as well as the corresponding information for transitions.

the number of input places and output places. For the example of Fig. 4, there are only three types of transitions (one input place to one output place, one to two and two to one) and three templates in the UPPAAL translation. Then one of these templates is instantiated for each transition of the TPN we started with. An example of a UPPAAL template for transitions having one input place and one output place is given in Fig. 5; integers B1 and F1 give respectively the index of the unique input place of the transition, and the index of the output place. The timing constraints of the transition are given by $dmin$ and $dmax$. We can handle as well transitions with input and output arcs with arbitrary weights (on the examples of Fig. 5 the input and output weights are 1).

In our translation, each transition of the TPN is implemented by a TA with one clock. The synchronized product thus contains as many clocks as the number of transitions of the TPN. At first sight, one can think that the translation we have proposed is far too expensive w.r.t. to the number of clocks to be of any use when using a model-checker like UPPAAL: indeed the model-checking of TA is exponential in the number of clocks. Nevertheless we do not need to keep track of all the clocks as many of them are not useful in many states.

6.2. Inactive Clocks

When a transition in a TPN is disabled, there is no need to store the value of the clock for this transition: this was already used in the seminal paper (Berthomieu and Diaz, 1991). Accordingly when the TA of a transition is in location \bar{t} (i.e., t is not enabled) we do not need to store the value of the clock: this means that many of the clocks can often be disregarded. In UPPAAL, there is a corresponding notion of *inactive clock*:

Definition 19 (UPPAAL Inactive Clock) *Let \mathcal{A} be a timed automaton. Let x be a clock of \mathcal{A} and ℓ be a location of \mathcal{A} . If on all paths starting from (ℓ, v) in $S_{\mathcal{A}}$, the clock x is always reset before being tested then the clock x is inactive in location ℓ . A clock is active if it is not inactive.*

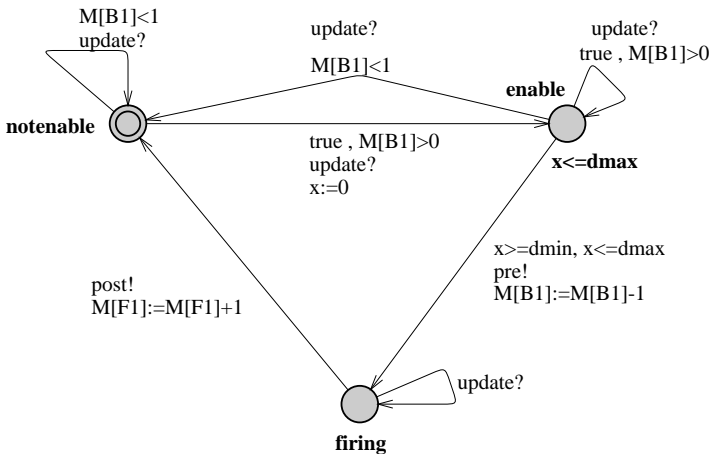


Figure 5. A UPPAAL Template Obtained with the “Export to UPPAAL” Feature of ROMEO

A consequence of the notion of inactive clocks in UPPAAL is that at location ℓ the constraints on the clocks will only contain the active clocks (they can be omitted in the DBM that represents it). The next proposition (which is easy to prove on the timed automaton of a transition) states that our translation is *effective* w.r.t. active clocks reduction i.e., that when a TA of a transition is not in state t (enabled) the corresponding clock is considered inactive by UPPAAL.

Proposition 1 *Let \mathcal{A}_i be the timed automaton associated with transition t_i of a TPN \mathcal{T} (see Fig. 2, page 240). The clock x_i of \mathcal{A}_i is inactive in locations Firing and \bar{t} .*

The recent versions of UPPAAL ($\geq 3.4.7$) computes active clocks syntactically for each automaton of the product. When the product automaton is computed “on-the-fly” (for verification purposes), the set of active clocks for a product location is simply the union of the set of active clocks of each component. Again without difficulty we obtain the following theorem:

Theorem 7 *Let \mathcal{T} be a TPN and $\Delta(\mathcal{T})$ the equivalent product of timed automata (see section 4.3). Let M be a reachable marking of $S_{\mathcal{T}}$ and ℓ the equivalent¹⁰ location in $S_{\Delta(\mathcal{T})}$. The number of active clocks in ℓ is equal to the number of enabled transitions in the marking M .*

Thanks to this theorem and to the *active clocks reduction* feature of UPPAAL the model-checking of TCTL properties on the network of timed automata given by our translation can be efficient. Of course there are still examples with a huge number of transitions, all enabled at any time that we will not be able to analyze, but those examples cannot be handled by any existing tool for TPN.

In the next subsection we apply our translation to some recent and non trivial examples of TPNs that can be found in (Gardey et al., 2005).

6.3. Tools for Analyzing TPNs

One feature of ROMEO is to export a TPN to UPPAAL or KRONOS but it was originally developed to analyze directly TPNs and has many built-in capabilities: we refer to ROMEO STD for the tool ROMEO with these capabilities (Gardey et al., 2005). TINA (Berthomieu and Vernadat, 2006b) is another state-of-the-art tool to analyze TPNs with some more capabilities than ROMEO STD: it allows to produce an Atomic State Class Graph (ASCG) on which CTL* properties can be checked. Using ROMEO STD or TINA is a matter of taste as both tools give similar results on TPNs.

Table 2 gives a comparison in terms of the classes of property (LTL, CTL, TCTL, Liveness) the tools can handle. The columns UPPAAL and KRONOS in ROMEO give the combined capabilities obtained when using our structural translation and the corresponding (timed) model-checker.

Regarding time performance ROMEO STD and TINA give almost the same results. Moreover with ROMEO STD and TINA, model-checking LTL or CTL properties will usually be faster than using ROMEO +UPPAAL: those tools implement efficient algorithms to produce the (A)SCG needed to perform LTL or CTL model-checking. On one hand it is to be noticed that both ROMEO STD and TINA need 1) to produce a file containing the (A)SCG; and then 2) to run a model-checker on the obtained graph to check for the (LTL, CTL or CTL*) property. This can be prohibitive on very large examples (see (Cassez and Roux, 2006)).

On the other hand neither ROMEO STD nor TINA are able to check quantitative properties such as quantitative liveness (like property of equation (5) below) and TCTL which in general

¹⁰See Definition 16.

	TINA	ROMEO		
		ROMEO STD	ROMEO translation from TPN to TA	
			UPPAAL	KRONOS
Marking Reachability	Compute marking graph	ROMEO-TCTL ^c	UPPAAL-TCTL ^c	TCTL
LTL	SCG ^a + MC ^b			
CTL	(CTL*) ASCG ^a + MC ^b			
TCTL	-			

^aSCG = Computation of the State Class Graph ; ASCG = of the atomic SCG.

^bMC = requires the use of a Model-Checker on the SCG.

^cCorresponds to a subset of TCTL and a special type of liveness defined by formulas of the form $\forall \square(\varphi \implies \forall \diamond \Psi)$.

Table 2. What can we do with the different tools and approaches?

cannot be encoded with an observer (when this possible we can translate such a quantitative property into a problem of marking reachability).

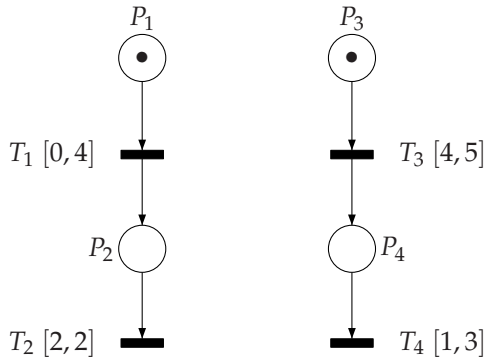


Figure 6. The TPN \mathcal{T}_g

Let us consider the TPN \mathcal{T}_g of Fig. 6. The *response* (liveness) property,

$$\forall \square((M[1] > 0 \wedge M[3] > 0 \wedge T_1.x > 3) \implies \forall \diamond(M[2] > 0 \wedge M[4] > 0)) \tag{5}$$

where $M[i]$ is the marking of the place P_i , cannot be checked with TINA and can easily be checked with our method using the translation and UPPAAL. This property means that if we do not fire T_1 before 3 t.u. then it is unavoidable that at some point in the future there is a marking with a token in P_2 and in P_4 . In UPPAAL we can use the response property template $P \rightarrow Q$ which corresponds to $\forall \square(P \implies \forall \diamond Q)$. Using our TPN-TCTL translation we obtain:

$$(SU.0 \text{ and } M[1]>0 \text{ and } M[3]>0 \text{ and } T_1.x>3) \text{ --> } (SU.0 \text{ and } M[2]>0 \text{ and } M[4]>0)$$

6.4. Experimental Results

We just point out that our translation is syntactic and the time to translate a TPN into an equivalent product of TA is negligible. This is in contrast with the method used in TINA and ROMEO STD where the whole state space has to be computed in order to build some graph (usually very large) and later on, a model-checker has to be used to check the property on the graph. Reports on experimental results on different types of TPNs (cyclic tasks, producers/consumers and large TPNs) can be found in (Cassez and Roux, 2006).

7. Conclusion

In this chapter, we have presented time Petri Nets (TPNs) and a structural translation from TPNs to TA. Any TPN \mathcal{T} and its associated TA $\Delta(\mathcal{T})$ are timed bisimilar.

Such a translation has many theoretical implications. Most of the positive theoretical results on TA carry over to TPNs. The class of TPNs can be extended by allowing strict constraints (open, half-open or closed intervals) to specify the firing dates of the transitions; for this extended class, the following results follow from our translation and from Theorem 5:

- TCTL model checking is decidable for bounded TPNs. Moreover efficient algorithms used in UPPAAL (Pettersson and Larsen, 2000) and KRONOS (Yovine, 1997) are exact for the class of TA obtained with our translation;
- it is decidable whether a TA is non-zeno or not (Henzinger et al., 1994) and thus our result provides a way to decide non-zenoness for bounded TPNs;
- lastly, as our translation is structural, it is possible to use a model-checker to find sufficient conditions of unboundedness of the TPN.

These results enable us to use algorithms and tools developed for TA to check quantitative properties on TPNs. For instance, it is possible to check real-time properties expressed in the logic TCTL on bounded TPNs. The tool ROMEO (Gardey et al., 2005) that has been developed for the analysis of TPN (state space computation and “on-the-fly” model-checking of reachability properties) implements this translation of a TPN into the equivalent TA in UPPAAL input format.

Our approach turns out to be a good alternative to existing methods for verifying TPNs:

- with our translation and UPPAAL we were able to check safety properties on very large TPNs that cannot be handled by other existing tools;
- we also extend the class of properties that can be checked on TPNs to real-time quantitative properties.

Note also that using our translation, we can take advantage of all the features of a tool like UPPAAL: looking for counter examples is usually much faster than checking a safety property. Moreover if a safety property is false, we will obtain a counter example even for unbounded TPNs (if we use breadth-first search).

There are currently new features being developed for tools like ROMEO that enables one to directly check TCTL properties on a TPN without translating it into a TA.

Acknowledgments

The authors wish to thank Didier Lime for his careful reading of this chapter and useful comments to improve many parts of the text.

8. References

- Abdulla, P. A. and Nylén, A. (2001). Timed Petri nets and BQOs. In *22nd International Conference on Applications and Theory of Petri Nets (ICATPN'01)*, volume 2075 of *Lecture Notes in Computer Science*, pages 53–70, Newcastle upon Tyne, UK. Springer-Verlag.
- Alur, R. and Dill, D. L. (1994). A Theory of Timed Automata. *Theoretical Computer Science*, 126(2):183–235.
- Aura, T. and Lilius, J. (2000). A Causal Semantics for Time Petri Nets. *Theoretical Computer Science*, 243(2):409–447.
- Bérard, B., Cassez, F., Haddad, S., Lime, D., and Roux, O. H. (2005a). Comparison of the Expressiveness of Timed Automata and Time Petri Nets. In *3rd International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS'05)*, volume 3829 of *Lecture Notes in Computer Science*, Uppsala, Sweden. Springer-Verlag.
- Bérard, B., Cassez, F., Haddad, S., Lime, D., and Roux, O. H. (2005b). When are Timed Automata Weakly Timed Bisimilar to Time Petri Nets? In *25th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'05)*, volume 3821 of *Lecture Notes in Computer Science*, Hyderabad, India. Springer-Verlag.
- Bérard, B., Cassez, F., Haddad, S., Roux, O., and Lime, D. (2005c). Comparison of Different Semantics for Time Petri Nets. In Xiaoyu, M., Cardoso, J., and Valette, R., editors, *Proceedings of the Third International Symposium on Automated Technology for Verification and Analysis (ATVA'2005)*, volume 3707 of *Lecture Notes in Computer Science*, pages 293–307, Taipei, Taiwan. Springer-Verlag.
- Berthomieu, B. and Diaz, M. (1991). Modeling and Verification of Time Dependent Systems Using Time Petri Nets. *IEEE Transactions on Software Engineering*, 17(3):259–273.
- Berthomieu, B. and Menasche, M. (1983). An Enumerative Approach for Analyzing Time Petri Nets. In Mason, R. E. A., editor, *Information Processing: proceedings of the IFIP congress 1983*, volume 9 of *IFIP congress series*, pages 41–46.
- Berthomieu, B. and Vernadat, F. (2003). State Class Constructions for Branching Analysis of Time Petri Nets. In *Proc. 9th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'03)*, volume 2619 of *Lecture Notes in Computer Science*, pages 442–457. Springer-Verlag.
- Berthomieu, B. and Vernadat, F. (2006a). TINA. <http://www.laas.fr/tina>.
- Berthomieu, B. and Vernadat, F. (2006b). Time Petri Nets Analysis with TINA. In *Third International Conference on the Quantitative Evaluation of Systems (QEST 2006)*, pages 123–124, Riverside, California, USA. IEEE Computer Society.
- Bornot, S., Sifakis, J., and Tripakis, S. (1998). Modeling Urgency in Timed Systems. In de Roever, W. P., Langmaack, H., and Pnueli, A., editors, *International Symposium on Compositionality: The Significant Difference (COMPOS'97)*, volume 1536 of *Lecture Notes in Computer Science*, pages 103–129, Bad Malente, Germany. Springer-Verlag.
- Bouyer, P. (2003). Untamable Timed Automata! In *20th Annual Symposium on Theoretical Aspects of Computer Science (STACS'03)*, volume 2607 of *Lecture Notes in Computer Science*, pages 620–631, Berlin, Germany. Springer-Verlag.

- Bouyer, P. (2004). Forward Analysis of Updatable Timed Automata. *Formal Methods in System Design*, 24(3):281–320.
- Bouyer, P., Dufourd, C., Fleury, E., and Petit, A. (2000). Are Timed Automata Updatable? In *Proc. 12th International Conference on Computer Aided Verification (CAV'00)*, volume 1855 of *Lecture Notes in Computer Science*, pages 464–479. Springer-Verlag.
- Cassez, F. and Roux, O. H. (2006). Structural Translation from Time Petri Nets to Timed Automata – Model-Checking Time Petri Nets via Timed Automata. *The Journal of Systems and Software*, 79(10):1456–1468.
- Cortès, L. A., Eles, P., and Peng, Z. (2000). Verification of Embedded Systems Using a Petri Net based Representation. In *13th International Symposium on System Synthesis (ISSS 2000)*, pages 149–155, Madrid, Spain.
- de Frutos Escrig, D., Ruiz, V. V., and Alonso, O. M. (2000). Decidability of Properties of Timed-Arc Petri Nets. In *21st International Conference on Applications and Theory of Petri Nets (ICATPN'00)*, volume 1825 of *Lecture Notes in Computer Science*, pages 187–206, Aarhus, Denmark. Springer-Verlag.
- Diaz, M. and Senac, P. (1994). Time Stream Petri Nets: A Model for Timed Multimedia Information. In *15th International Conference on Applications and Theory of Petri Nets (ICATPN'94)*, volume 815 of *Lecture Notes in Computer Science*, pages 219–238, Zaragoza, Spain. Springer-Verlag.
- Emerson, E. A. (1990). Temporal and Modal Logic. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics (B)*, pages 995–1072. Elsevier.
- Gardey, G., Lime, D., Magnin, M., and Roux, O. H. (2005). ROMEO: A Tool for Analyzing Time Petri Nets. In *Proc. 17th International Conference on Computer Aided Verification (CAV'05)*, volume 3576 of *Lecture Notes in Computer Science*, pages 418–423, Edinburgh, Scotland, UK. Springer-Verlag. <http://romeo.rts-software.org>.
- Gardey, G., Roux, O. H., and Roux, O. F. (2003). Using Zone Graph Method for Computing the State Space of a Time Petri Net. In *International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS'03)*, volume 2791 of *Lecture Notes in Computer Science*, Marseille, France. Springer-Verlag.
- Gardey, G., Roux, O. H., and Roux, O. F. (2006). 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*, 6(3):301–320.
- Henzinger, T. A., Nicollin, X., Sifakis, J., and Yovine, S. (1994). Symbolic Model Checking for Real-Time Systems. *Information and Computation*, 111(2):193–244.
- Jones, N. D., Landweber, L. H., and Lien, Y. E. (1977). Complexity of some problems in Petri nets. *Theoretical Computer Science*, 4:277–299.
- Khansa, W., Denat, J.-P., and Collart-Dutilleul, S. (1996). P-Time Petri Nets for Manufacturing Systems. In *International Workshop on Discrete Event Systems (WODES'96)*, pages 94–102, England. IEEE Computer Society.
- Laroussinie, F. and Larsen, K. G. (1998). CMC: A Tool for Compositional Model-Checking of Real-Time Systems. In Budkowski, S., Cavalli, A. R., and Najm, E., editors, *Proceedings of IFIP TC6 WG6.1 Joint International Conference on Formal Description Techniques*

- for *Distributed Systems and Communication Protocols (FORTE'XI) and Protocol Specification, Testing and Verification (PSTV'XVIII)*, volume 135 of *IFIP Conference Proceedings*, pages 439–456, Paris, France. Kluwer Academic Publishers.
- Larsen, K. G., Pettersson, P., and Yi, W. (1997). UPPAAL in a Nutshell. *International Journal of Software Tools for Technology Transfer*, 1(1–2):134–152. <http://www.uppaal.com/>.
- Lilius, J. (1998). Efficient State Space Search for Time Petri Nets. *Electronic Notes in Theoretical Computer Science*, 18.
- Lime, D. and Roux, O. H. (2006). Model Checking of Time Petri Nets Using the State Class Timed Automaton. *Journal of Discrete Events Dynamic Systems - Theory and Applications*, 16(2):179–205.
- Merlin, P. M. (1974). *A Study of the Recoverability of Computing Systems*. PhD thesis, Dep. of Information and Computer Science, Univ. of California, Irvine, CA.
- Pettersson, P. and Larsen, K. G. (2000). UPPAAL2k. *Bulletin of the European Association for Theoretical Computer Science*, 70:40–44.
- Pezzè, M. (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 of 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.
- Rokicki, T. G. (1993). *Representing and Modeling Circuits*. PhD thesis, Stanford University.
- Sava, A. T. (2001). *Sur la synthèse de la commande des systèmes à événements discrets temporisés*. PhD thesis, Institut National polytechnique de Grenoble, Grenoble, France.
- Sifakis, J. (1980). Performance Evaluation of Systems using Nets. In Brauer, W., editor, *Net theory and applications : Proceedings of the advanced course on general net theory, processes and systems*, volume 84 of *Lecture Notes in Computer Science*, pages 307–319, Hamburg, Germany. Springer-Verlag.
- Sifakis, J. and Yovine, S. (1996). Compositional specification of timed systems. In Puech, C. and Reischuk, R., editors, *13th Annual Symposium on Theoretical Aspects of Computer Science (STACS'96)*, volume 1046 of *Lecture Notes in Computer Science*, pages 347–359, Grenoble, France. Springer-Verlag.
- Tripakis, S. (1999). Timed Diagnostics for Reachability Properties. In Cleaveland, R., editor, *Proc. 5th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'99)*, volume 1579 of *Lecture Notes in Computer Science*, pages 59–73, Amsterdam, The Netherlands. Springer-Verlag.
- Yovine, S. (1997). KRONOS: A Verification Tool for Real-Time Systems. *International Journal of Software Tools for Technology Transfer*, 1(1–2):123–133.