Franck Cassez and Olivier H. Roux. From Time Petri Nets to Timed Automata. Research Report 1496, RI-2003-4, IRCCyN/CNRS, Nantes, June 2003. 20 pages. Available at www.irccyn.ec-nantes.fr
Updated May 2004.

From Time Petri Nets to Timed Automata

Franck Cassez and Olivier-H. Roux

IRCCyN/CNRS UMR 6597
BP 92101
1 rue de la Noë
44321 Nantes Cedex 3 France,
email: Firstname.Surname@irccyn.ec-nantes.fr

Abstract. In this paper, we consider Time Petri Nets (TPN) where time is associated with transitions. We give a formal semantics for TPNs in terms of Timed Transition Systems. Then, we propose a translation from TPNs to Timed Automata (TA) that preserves the behavioural semantics (timed bisimilarity) of the TPNs. For the theory of TPNs this result is two-fold: i) reachability problems and more generally TCTL model-checking are decidable for bounded TPNs; ii) allowing strict time constraints on transitions for TPNs preserves the results described in i). The practical applications of the translation are: i) one can specify a system using both TPNs and Timed Automata and a precise semantics is given to the composition; ii) one can use existing tools for analysing timed automata (like KRONOS or UPPAAL or CMC) to analyse TPNs.

Keywords : Time Petri Nets, Timed Automata, Model-Checking, Temporal logics.

1 Introduction

Classes of Petri Nets with Time. The two main extensions of Petri Nets with time are Time Petri Nets (TPNs) [Mer74] and Timed Petri Nets [Ram74]. For TPNs a transition can fire within a time interval whereas for Timed Petri Nets it fires as soon as possible. Among Timed Petri Nets, time can be considered relative to places or transitions [Sif80, PY99]. The two corresponding subclasses namely P-Timed Petri Nets and T-Timed Petri Nets are expressively equivalent [Sif80, PY99]. The same classes are defined for TPNs i.e. T-TPNs and P-TPNs, but both classes of Timed Petri Nets are included in both P-TPNs and T-TPNs [PY99]. P-TPNs and T-TPNs are incomparable [KDCD96]. Finally TPNs form a subclass of Time Stream Petri Nets [DS94] which were introduced to model multimedia applications.

For classical transition-time Petri nets, boundedness is undecidable, and works on this model report undecidability results, or decidability under the assumption that the TPN is bounded (as for reachability decidability [Pop91]). Boundedness and other results are obtained by computing the state-space. Recent work [AN01, dFERA00] consider timed arc Petri nets where each token has a clock representing his "age". They prove that coverability and boundedness are decidable for this class of Petri nets by applying a backward exploration technique [AJ01, FS98]. However, they assume a lazy (non-urgent) behavior of the net. This means that the firing of transitions may be delayed, even if that implies that some transitions are disabled because their input tokens become too old.

Reachability for Time Petri Nets. As for TA, 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 $d \in \mathbb{R}_{\geq 0}$. Then a sequence of transitions $\omega = (t_1, d_1)(t_2, d_2) \cdots (t_n, d_n)$ indicates that t_1 is fired after d_1 time units, then t_2 is fired after d_2 time units, 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 iff there is a timed firing sequence ω leading from the initial marking M_0 to M. It is common for TA to define the *untimed* sequence from the timed one: if $\omega = (t_1, d_1)(t_2, d_2) \cdots (t_n, d_n)$ then $Untimed(\omega) = t_1 t_2 \cdots t_n$. Reachability analysis of TPNs relies on the construction of the so-called States Classes Graph (SCG) that was introduced in [BM83] and later refined in [BD91]. It has been recently improved in [Lil99] by using partial-order reduction methods.

For bounded TPNs, the SCG construction obviously solves marking reachability. State reachability can be decided using the alternative *strong state classes* constructions introduced in [BV03], and liveness can be decided using the *atomic state classes* introduced in the same paper. Strong state classes differ from the state classes in that they canonically represent state sets, while the latter represent state sets modulo some equivalence relation. Atomic state classes are obtained by partition refinement of strong state classes, their graph is bisimilar with the state graph of the net (with temporal annotations omitted), so they preserve branching properties (these include CTL properties).

The nodes of the SCG are sets of *states* (a state is a pair consisting of a marking and a firing constraint) of the TPN and the edges are labeled with transitions' names from T (the set of transitions of the TPN). C_0 is the states class containing the initial marking M_0. The SCG defined in [BD91] has the following property: there is a path $C_0 \xrightarrow{\sigma} C$ in the SCG with $\sigma \in T^*$ iff there is a timed path $\omega \in (T \times \mathbb{R}_{\geq 0})^*$ s.t. $M_0 \xrightarrow{\omega} M$ with $M \in C$ and $Untimed(\omega) = \sigma$. This can be rephrased in terms of language acceptance for TA, assuming all markings are final states. If $L \subseteq T^*$ is the language accepted by the SCG and L' is the timed language accepted by the TPN, then $L = Untimed(L')$. Henceforth the SCG can only be used to check untimed reachability properties[1] but is not 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".

Timed Automata. Timed Automata (TA) were introduced by Alur & Dill [AD94] and have since been extensively studied [LRT94, LPY95, AFH99, BDFP00].

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

This model is an extension of finite automata with (dense time) *clocks* and enables one to specify real-time systems. It has been shown that model-checking for TCTL properties is decidable [ACD93, HNSY94] for TA and some extensions of TA [BDFP00]. There also exist several efficient tools like UPPAAL [LPY97, PL00], KRONOS [Yov97] and CMC [LL98] for model-checking TA and many real-time industrial applications have been specified and successfully verified with them [BGK$^+$96, MY96, LPY98].

Related Work. The relationship between TPNs and TA has not been often investigated. In [SY96] 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 compositions.

In [BST98] 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).

Sava [Sav01] considers bounded TPN where the associated underlying Petri net is not necessarily safe and proposes an algorithm to compute the region graph of a TPN. The result is a timed automaton with a clock for each transition of the original TPN. This automaton is then restricted for the command by supervision. However, they do not give any result to stop the automaton computation when the TPN is not bounded (which one does not know *a priori*) and when it is bounded, they do not prove that the algorithm stop.

Lime and Roux [LR03] propose an extension of the state class graph construction that allows to build the state class graph as a timed automaton. They prove that this timed automaton and the TPN are timed-bisimilar and they also prove a relative minimality of the number of clocks of 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 TPN.

In this article, we consider a structural translation from TPN (not necessary bounded) to TA. Our purpose is focussed on the semantics of TPNs, on their relations with TAs and on the verification of temporal properties on Time Petri Nets.

Our Contribution. First we give a formal semantics for Time Petri Nets [Mer74] in terms of Timed Transition Systems [LPY95]. Then we present a structural translation of a TPN into a synchronized product of timed automata that preserves the semantics (in the sense of *timed bisimilarity*) of the TPN. This yields theoretical and practical applications of this translation : i) TCTL [ACD93, HNSY94] model-checking is decidable for bounded TPNs and TCTL properties can now be checked (efficiently) for TPNs with existing tools for analyzing timed automata (like KRONOS or UPPAAL or CMC) We have developed a tool for translating TPNs to UPPAAL input format; ii) allowing strict time constraints on transitions for TPNs preserves the results described in i); iii) one can specify a system using both TPNs and Timed Automata and a precise semantics is given to the composition; iv) as the translation is structural, one can use unboundedness testing methods to detect behavior leading to the unboundedness of a TPN.

Outline of the paper. Section 2 introduces the semantics of TPNs in terms of timed transition systems and the basics of TA. In section 3 we show how to build a synchronized product of TA that is equivalent to a TPN. We also prove that the semantics of the TPN and its associated product of TA are timed bisimilar. This enables us to check for real-time properties expressed in TCTL in section 4, where we also address the problem of reducing the number of clocks used in the translation. We present our translator and an example of model-checking with UPPAAL. Finally we conclude with our ongoing work and perspectives in section 5.

2 Time Petri Nets and Timed Automata

Notations. We denote B^A 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 $+, -, <, =$ used on vectors of A^n with $A = \mathbb{N}, \mathbb{Q}, \mathbb{R}$ are the point-wise extensions of their counterparts in A. For a *valuation* $\nu \in A^n, d \in A$, $\nu + d$ denotes the vector $(\nu + d)_i = \nu_i + d$, and for $A' \subseteq A$, $\nu[A' \mapsto 0]$ denotes the valuation ν' with $\nu'(x) = 0$ for $x \in A'$ and $\nu'(x) = \nu(x)$ otherwise. We denote $\mathcal{C}(V)$ for the *simple constraints* over a set of variables V. $\mathcal{C}(V)$ is defined to be the set of boolean combinations (with the connectives $\{\wedge, \vee, \neg\}$) of terms of the form $v - v' \bowtie c$ or $v \bowtie c$ for $v, v' \in V$ and $c \in \mathbb{N}$ and $\bowtie \in \{<, \leq, =, \geq, >\}$. For a transition system we write transitions as $s \xrightarrow{a} s'$ and sequence of transitions

$$s_0 \xrightarrow{a_1} s_1 \longrightarrow \cdots \xrightarrow{a_n} s_n$$

as $s_0 \overset{w}{\Longrightarrow} s_n$ with $w = a_1 a_2 \cdots a_n$.

2.1 Time Petri Nets

The model. Time Petri Nets were introduced in [Mer74] and extend Petri Nets with timing constraints on the firings of transitions.

Definition 2.1. A *Time Petri Net* $\mathcal{T}$ is a tuple $(P, T, {}^\bullet(.), (.)^\bullet, M_0, (\alpha, \beta))$ where:

- $P = \{p_1, p_2, \cdots, p_m\}$ is a finite set of *places*,
- $T = \{t_1, t_2, \cdots, t_n\}$ is a finite set of *transitions*,

– ${}^\bullet(.) \in (\mathbb{N}^P)^T$ is the *backward* incidence mapping,
– $(.)^\bullet \in (\mathbb{N}^P)^T$ is the *forward* incidence mapping,
– $M_0 \in \mathbb{N}^P$ is the *initial* marking,
– $\alpha \in (\mathbb{Q}_{\geq 0})^T$ and $\beta \in (\mathbb{Q}_{\geq 0} \cup \{\infty\})^T$ are respectively the *earliest* and *latest* firing time mappings. $\square$

Semantics of Time Petri Nets. The semantics of TPNs can be given in term of Timed Transition Systems (TTS) [LPY95] which are usual transition systems with two types of labels: discrete labels for events and positive reals labels for time elapsing.

$\nu \in (\mathbb{R}_{\geq 0})^n$ is a *valuation* such that each value ν_i is the elapsed time since the last time transition t_i was enabled. $\overline{0}$ is the initial valuation with $\forall i \in [1..n], \overline{0}_i = 0$. 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 is *enabled* in a marking M iff $M \geq {}^\bullet t$. $\uparrow \mathit{enabled}(t_k, M, t_i) \in \mathbb{B}$ 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 [BD91, AL00] 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 $M' = M - {}^\bullet t_i + t_i{}^\bullet$" [BD91].

Formally this gives:

$$\uparrow \mathit{enabled}(t_k, M, t_i) = \left(M - {}^\bullet t_i + t_i{}^\bullet \geq {}^\bullet t_k\right) \wedge \left((M - {}^\bullet t_i < {}^\bullet t_k) \vee (t_k = t_i)\right) \quad (1)$$

Definition 2.2. The semantics of a TPN $\mathcal{T}$ is a timed transition system $S_{\mathcal{T}} = (Q, q_0, \rightarrow)$ where:

– $Q = \mathbb{N}^P \times (\mathbb{R}_{\geq 0})^n$,
– $q_0 = (M_0, \overline{0})$,
– $\longrightarrow \in Q \times (T \cup \mathbb{R}_{\geq 0}) \times Q$ consists of the discrete and continuous transition relations:
  - the discrete transition relation is defined $\forall t_i \in T$:

$$(M, \nu) \xrightarrow{t_i} (M', \nu') \textit{ iff } \begin{cases} M \geq {}^\bullet t_i \wedge M' = M - {}^\bullet t_i + t_i{}^\bullet \\ \alpha(t_i) \leq \nu_i \leq \beta(t_i) \\ \nu'_k = \begin{cases} 0 & \textit{if } \uparrow \mathit{enabled}(t_k, M, t_i), \\ \nu_k & \textit{otherwise.} \end{cases} \end{cases}$$

  - the continuous transition relation is defined $\forall d \in \mathbb{R}_{\geq 0}$:

$$(M, \nu) \xrightarrow{\epsilon(d)} (M, \nu') \textit{ iff } \begin{cases} \nu' = \nu + d \\ \forall k \in [1..n], \left(M \geq {}^\bullet t_k \Longrightarrow \nu'_k \leq \beta(t_k)\right) \end{cases}$$

A *run* of a time Petri net $\mathcal{T}$ is a path in $S_{\mathcal{T}}$ starting in q_0. The set of runs of $\mathcal{T}$ is denoted $[\![\mathcal{T}]\!]$. The set of *reachable markings* of $\mathcal{T}$ is denoted $\mathit{Reach}(\mathcal{T})$. If the set $\mathit{Reach}(\mathcal{T})$ is finite we say that $\mathcal{T}$ is *bounded.* As a shorthand we write $(M, \nu) \longrightarrow^d_e (M', \nu')$ for a sequence of time elapsing and discrete steps like $(M, \nu) \xrightarrow{\epsilon(d)} (M'', \nu'') \xrightarrow{e} (M', \nu')$. $\square$

This definition appeals for several comments. Our semantics is based on the common definition of [BD91, AL00] for safe TPNs.

First, previous formal semantics [BD91, Lil99, PY99, AL00] for TPNs usually require 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 TPNs[2]. 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 [BD91].

Second, some variations can be found in the literature about TPNs concerning the firing of transitions. [PY99] 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 [Mer74, BD91, PY99, AL00].

Third, it is possible for the TPN to be zeno or unbounded. In the case it is unbounded, the discrete component of the state space of the timed transition system is infinite. If $\forall i, \alpha(t_i) > 0$ then the TPN is non-zeno and the requirement that time diverges on each run is fulfilled. Otherwise, if the TPN is bounded and at least one lower bound is 0, the zeno or non-zeno property can be decided [HNSY94] for the TPN using the equivalent timed automaton we build in section 3.

2.2 Timed Automata and Products of Timed Automata

Timed automata [AD94] are used to model systems which combine *discrete* and *continuous* evolutions.

Definition 2.3 (Timed Automaton). A *Timed Automaton* H is a 6-tuple (N, l_0, C, A, E, Inv) where:

- N is a finite set of *locations*,
- $l_0 \in N$ is the *initial location*,
- C is a finite set of positive real-valued *clocks*,
- A is a finite set of *actions*,
- $E \subseteq N \times \mathcal{C}(V) \times A \times 2^C \times N$ 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.
- $Inv \in \mathcal{C}(V)^N$ 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 also a timed transition system.

Definition 2.4 (Semantics of a Timed Automaton). The semantics of a timed automaton $H = (N, l_0, C, A, E, Act, Inv)$ is a timed transition system $S_H = (Q, q_0, \rightarrow)$ with $Q = N \times (\mathbb{R}_{\leq 0})^C$, $q_0 = (l_0, \overline{0})$ is the initial state and $\rightarrow$ is defined by:

[2] If we except the difference with [Lil99] in the definition of the reset instants for newly enabled transitions.

$$(l,v) \xrightarrow{a} (l',v') \quad \textit{iff } \exists\,(l,\gamma,a,R,l') \in E \ \textit{s.t.} \begin{cases} \gamma(v) = \mathsf{tt},\ v' = v[R \mapsto 0] \ \textit{and} \\ \mathit{Inv}(l')(v') = \mathsf{tt} \end{cases}$$

$$(l,v) \xrightarrow{\epsilon(t)} (l',v') \ \textit{iff} \begin{cases} l = l' \qquad v' = v + t \quad \textit{and} \\ \forall\, 0 \le t' \le t,\ \mathit{Inv}(l)(v+t') = \mathsf{tt} \end{cases}$$

A *run* of a timed automaton H is a path in S_H starting in q_0. The set of runs of H is denoted $[\![H]\!]$. $\square$

Product of Timed Automata. 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 [Arn94]. Let H_1, $\ldots$, H_n be n timed automata with $H_i = (N_i, l_{i,0}, C_i, A, E_i, Inv_i)$. A *synchronization function* f is a partial function from $(A \cup \{\bullet\})^n \hookrightarrow A$ where $\bullet$ is a special symbol used when an automaton is not involved in a step of the global system. Note that f is a synchronization function with renaming. We denote by $(H_1|\ldots|H_n)_f$ the parallel composition of the H_i's w.r.t. f. The configurations of $(H_1|\ldots|H_n)_f$ are pairs $(\bar{l}, v)$ with $\bar{l} = (l_1, \ldots, l_n) \in N_1 \times \ldots \times N_n$ and $v = v_1 \cdots v_n$ with[3] $v_i \in (\mathbb{R}_{\ge 0})^{C_i}$ (we assume that all sets C_i of clocks are disjoint.) 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 and time can progress in the synchronized product also if all the components agree to. This is formalized by the following definition:

Definition 2.5 (Semantics of a Product of Timed Automata). Let H_1, $\ldots, H_n$ be n timed automata with $H_i = (N_i, l_{i,0}, C_i, A, E_i, Inv_i)$, and f a (partial) synchronization function $(A \cup \{\bullet\})^n \hookrightarrow A$. The semantics of $(H_1|\ldots|H_n)_f$ is a timed transition system $S = (Q, q_0, \rightarrow)$ with $Q = N_1 \times \ldots \times N_n \times (\mathbb{R}_{\ge 0})^C$, q_0 is the initial state $((l_{1,0}, \ldots, l_{n,0}), \overline{0})$ and $\rightarrow$ is defined by:

- $(\bar{l}, v) \xrightarrow{b} (\bar{l'}, v')$ iff there exists $(a_1, \ldots, a_n) \in (A \cup \{\bullet\})^n$ s.t. $f(a_1, \ldots, a_n) = b$ and for any i we have:
  - . If $a_i = \bullet$, then $l'_i = l_i$ and $v'_i = v_i$,
  - . If $a_i \in A$, then $(l_i, v_i) \xrightarrow{a_i} (l'_i, v'_i)$.
- $(\bar{l}, v) \xrightarrow{\epsilon(t)} (\bar{l}, v')$ iff $\forall\, i \in [1..n]$, we have $(l_i, v_i) \xrightarrow{\epsilon(t)} (l_i, v'_i)$ $\square$

We could equivalently define the product of n timed automata syntactically, building a new timed automaton [LPY95] from the n initial ones. In the sequel we consider a product $(H_1|\ldots|H_n)_f$ to be a timed automaton the semantics of which is timed bisimilar to the semantics of the product we have given in definition 2.5.

[3] v_i is the restriction of v to C_i.

3 From Time Petri Nets to Timed Automata

In this section, 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.

3.1 Translating Time Petri Nets into Timed Automata

We start with a time petri net $\mathcal{T} = (P, T, {}^\bullet(.), (.)^\bullet, M_0, (\alpha, \beta))$ with $P = \{p_1, \cdots, p_m\}$ and $T = \{t_1, \cdots, t_n\}$.

Timed Automaton for one Transition. We define one timed automaton $\mathcal{A}_i$ for each transition t_i of T (see Fig. 1). This timed automaton has one clock x_i. Also the states of the automaton $\mathcal{A}_i$ give the state of the transition t_i: in state t the transition is enabled; in state $\bar{t}$ it is disabled and in *Firing* it is being fired. The initial state 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 state is t otherwise it is $\bar{t}$. This automaton updates an array of integers p (s.t. $p[i]$ is the number of tokens in place p_i) which is shared by all the $\mathcal{A}_i$'s. This is not covered by the definition 2.5 but this one is very often extended ([LPY97, PL00]) with integer variables (this does not affect the expressiveness of the model).

The Supervisor. The supervisor SU is depicted on Fig. 2. The locations 1 to 3 subscripted with a c are assumed to be urgent or committed[4] which means that no time can elapse while visiting them. The initial state of the supervisor is 0.

Let us define the synchronization function f with $n+1$ parameters defined by:

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

We denote $\Delta(\mathcal{T}) = (SU \times \mathcal{A}_1 \times \cdots \times \mathcal{A}_n)_f$ the timed automaton associated to the TPN $\mathcal{T}$. 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 3.1 (States Equivalence). Let (M, ν) and $((s, p), \overline{q}, v)$ be respec-

[4] in SU, *committed* or *urgent* locations are equivalent; also this type of locations can be simulated by adding a clock x which is reset to 0 when entering the location and adding the invariant $x = 0$ to the location.

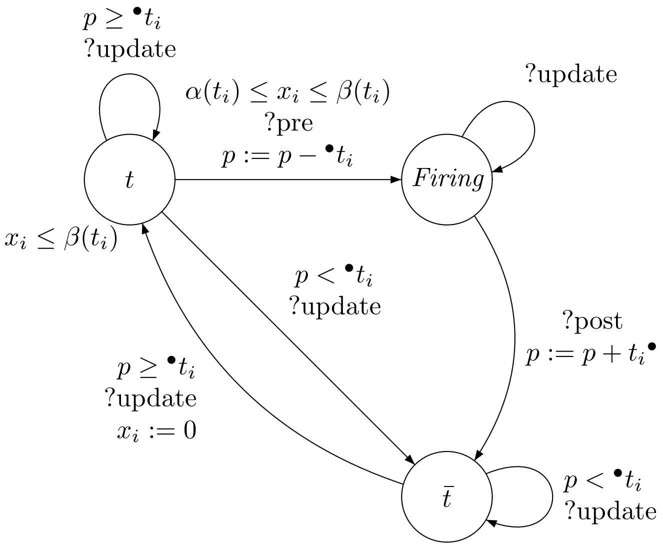

Figure 1. The automaton $\mathcal{A}_i$ for transition t_i

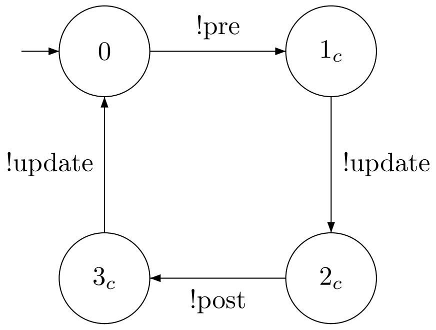

Figure 2. The automaton of the supervisor *SU*

tively a state of $S_{\mathcal{T}}$ and a configuration[5] of $S_{\Delta(\mathcal{T})}$. Then

$$(M,\nu) \approx ((s,p),\bar{q},v) \textit{ iff } \begin{cases} s = 0, \\ \forall i \in [1..m], \quad p[i] = M(p_i), \\ \forall k \in [1..n], \quad q_k = \begin{cases} t \textit{ if } M \geq {}^\bullet t_k, \\ \bar{t} \textit{ otherwise} \end{cases} \\ \forall k \in [1..n], \quad \nu_k = v_k \end{cases} \qquad \square$$

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. We assume a TPN $\mathcal{T}$ and $S_{\mathcal{T}} = (Q, q_0, \rightarrow)$ its semantics. Let $\mathcal{A}_i$ be the automaton associated with transition t_i of $\mathcal{T}$ as described by Fig. 1, SU the supervisor automaton of Fig. 2 and f the synchronization function defined previously. The semantics of $\Delta(\mathcal{T}) = (SU \times \mathcal{A}_1 \times \cdots \times \mathcal{A}_n)_f$ is the TTS $S_{\Delta(\mathcal{T})} = (Q_{\Delta(\mathcal{T})}, q_0^{\Delta(\mathcal{T})}, \rightarrow)$.

Theorem 3.2 (Timed Bisimilarity). *For all $(M,\nu) \in S_{\mathcal{T}}$ and $((0,p),\bar{q},v) \in S_{\Delta(\mathcal{T})}$ such that $(M,\nu) \approx ((0,p),\bar{q},v)$ we have:*

$$(M,\nu) \xrightarrow{t_i} (M',\nu') \textit{ iff } \begin{cases} ((0,p),\bar{q},v) \overset{w_i}{\Longrightarrow} ((0,p'),\bar{q}',v') \textit{ with} \\ w_i = pre_i.update.post_i.update \textit{ and} \\ (M',\nu') \approx ((0,p'),\bar{q}',v') \end{cases} \tag{2}$$

$$(M,\nu) \xrightarrow{\epsilon(d)} (M',\nu') \textit{ iff } \begin{cases} ((0,p),\bar{q},v) \xrightarrow{\epsilon(d)} ((0,p'),\bar{q}',v') \textit{ and} \\ (M',\nu') \approx ((0,p'),\bar{q}',v') \end{cases} \tag{3}$$

$\square$

Proof. We first prove statement 2. Assume $(M,\nu) \approx ((0,p),\bar{q},v)$. Then as t_i can be fired from (M,ν) we have: (i) $M \geq {}^\bullet t_i$ (ii) $\alpha(t_i) \leq \nu_i \leq \beta(t_i)$ (iii) $M' = M - {}^\bullet t_i + t_i{}^\bullet$ (iv) $\nu'_k = 0$ *if* $\uparrow$ *enabled*(t_k, M, t_i) and $\nu'_k = \nu_k$ otherwise. From (i) and (ii) and the state equivalence we deduce that $\bar{q}_i = t$ and $\alpha(t_i) \leq v_i \leq \beta(t_i)$. 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 $(!pre, \bullet, \cdots, ?pre, \cdots, \bullet)$ the global action pre_i is possible. After this move $\Delta(\mathcal{T})$ reaches state $((1,p_1),\bar{q}_1,v_1)$ with $\bar{q}_{1k} = \bar{q}_k, \forall k \neq i$ and $\bar{q}_{1i} = \textit{Firing}$. Also $p_1 = p - {}^\bullet t_i$ and $v_1 = v$.

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 $((1,p_1),\bar{q}_1,v_1)$ we reach $((2,p_2),\bar{q}_2,v_2)$ with $p_2 = p_1$, $v_2 = v_1$. For $k \neq i$, $\bar{q}_{2k} = \textit{if } p_1 \geq {}^\bullet t_k$ and $\bar{q}_{2k} = \bar{t}$ *otherwise* and $\bar{q}_{2i} = \textit{Firing}$.

[5] (s,p) is the state of SU, $\bar{q}$ gives the state of $\mathcal{A}_1 \times \cdots \times \mathcal{A}_n$ (so $\bar{q}_i$ is the state of $\mathcal{A}_1$) and v the values of the clocks $x_i, i \in [1..n]$.

The next global transition must be a $post_i$ transition leading to $((3, p_3), \bar{q}_3, v_3)$ with $p_3 = p_2 + t_i{}^\bullet$, $v_3 = v_2$ and $\bar{q_3}_k = \bar{q_2}_k, \forall k \neq i$ and $\bar{q_3}_i = \bar{t}$.

From this last state only an update transition leading to $((0, p_4), \bar{q}_4, v_4)$ is allowed, with $p_4 = p_3$, v_4 and q_4 given by: $\bar{q_4}_k = t$ if $p_3 \geq {}^\bullet t_k$ and $\bar{t}$ otherwise. $v_{4k} = 0$ if $\bar{q_3}_k = \bar{t}$ and $\bar{q_4}_k = t$ and $v_{4k} = v_{1k}$ otherwise. We then just notice that $\bar{q_3}_k = \bar{t}$ iff $p - {}^\bullet t_i < {}^\bullet t_k$ and $\bar{q_4}_k = t$ iff $p - {}^\bullet t_i + t_i{}^\bullet \geq {}^\bullet t_k$. This entails that $v_{4k} = 0$ iff $\uparrow$ *enabled*(t_k, p, t_i) and with (iv) gives $\nu'_k = v_{4k}$. As $p_4 = p_3 = p_2 + t_i{}^\bullet = p_1 - {}^\bullet t_i + t_i{}^\bullet = p - {}^\bullet t_i + t_i{}^\bullet$ using (iii) we have $\forall i \in [1..m], M'(p_i) = p_4[i]$. Hence we conclude that $((0, p_4), \bar{q}_4, v_4) \approx (M', \nu')$.

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 $(M, \nu) \xrightarrow{\epsilon(d)} (M', \nu')$ is allowed iff $\nu = \nu' + d$ and $\forall k \in [1..n], (M \geq {}^\bullet t_k \implies \nu'_k \leq \beta(t_k))$. From the states equivalence $(M, \nu) \approx ((0, p), \bar{q}, v)$, if $M \geq {}^\bullet t_k$ then $\bar{q}_k = t$ and the continuous evolution for $\mathcal{A}_k$ is constrained by the invariant $x_k \leq \beta(t_k)$. Otherwise $\bar{q}_k = \bar{t}$ and the continuous evolution is unconstrained for $\mathcal{A}_k$. No constraints apply for the supervisor in state 0. Hence the result. □

We can now state a useful corollary which enables us in the next section to do TCTL model-checking for TPNs. We denote $\Delta((M, \nu)) = ((0, p), \bar{q}, v)$ if $(M, \nu) \approx ((0, p), \bar{q}, v)$ and $\Delta(t_i) = pre_i.update.post_i.update$ and $\Delta(\epsilon(d)) = \epsilon(d)$. Just notice that Δ is one-to-one and we can use Δ^{-1} as well. Then we extend Δ to transitions as: $\Delta((M, \nu) \xrightarrow{e} (M', \nu')) = \Delta((M, \nu)) \xrightarrow{\Delta(e)} \Delta((M', \nu'))$ with $e \in T \cup \mathbb{R}_{\geq 0}$ (as $\Delta(t_i)$ is a word, this transition might be a 4 step transition). Again we can extend Δ to runs: if $\rho \in [\![\mathcal{T}]\!]$ we denote $\Delta(\rho)$ the associated run in $[\![\Delta(\mathcal{T})]\!]$. Notice that Δ^{-1} is only defined for runs σ of $[\![\Delta(\mathcal{T})]\!]$ the last state of which is of the form $((0, p), \bar{q}, v)$ where the supervisor is in state 0. We denote this property $last(\sigma) \models SU.0$.

Corollary 3.3. $\big(\rho \in [\![\mathcal{T}]\!] \wedge \sigma = \Delta(\rho)\big)$ *iff* $\big(\sigma \in [\![\Delta(\mathcal{T})]\!] \wedge last(\sigma) \models SU.0\big)$. □

Proof. The proof is a direct consequence of theorem 3.2. It suffices to notice that all the runs of $\Delta(\mathcal{T})$ are of the form

$$\sigma = (s_0, v_0) \Longrightarrow^{t_1}_{w_1} (s_1, v_1) \cdots \Longrightarrow^{t_n}_{w_n} (s_n, v_n)$$

with $w_i = pre_i.update.post_i.update$ and consequently using theorem 3.2, if the last state satisfy $last(\sigma) \models SU.0$, there exists a corresponding run ρ in $\mathcal{T}$ s.t. $\sigma = \Delta(\rho)$. □

Consequently all the runs in $\Delta(\mathcal{T})$ ending in $SU.0$ have a corresponding run in $\mathcal{T}$. This property will be used in section 4 when we address the problem of model-checking TCTL for TPNs.

4 TCTL Model-Checking for Time Petri Nets

We can now define TCTL [ACD93, HNSY94] for TPNs. The only difference with the versions of [ACD93, HNSY94] is that the atomic propositions usually associated to states are properties of markings. For practical applications with model-checkers we assume that the TPNs we check are bounded.

TCTL for TPNs.

Definition 4.1 (TCTL for TPN). Assume a TPN with n places, and m transitions it the set $T = \{t_1, t_2, \cdots, t_m\}$. The temporal logics TPN-TCTL is inductively defined by:

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

where $\mathbf{M}$ and **false** are keywords, $\varphi, \psi \in$ *TPN-TCTL*, $t_k, t_j \in T$, $c, d \in \mathbb{Z}$, $\bar{V} \in (\mathbb{Q} \cup \{\infty\})^n$ and[6] $\bowtie \in \{<, \leq, =, >, \geq\}$. □

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 usual shorthands $\mathbf{true} = \neg\mathbf{false}$, $\Diamond_{\bowtie c}\phi = \mathbf{true} \, \exists\mathcal{U}_{\bowtie c} \, \phi$ and $\Box_{\bowtie c} = \neg\Diamond_{\bowtie c}\neg\phi$.

The semantics of TPN-TCTL is defined on timed transition systems. Let $\mathcal{T} = (P, T, {}^\bullet(.), (.)^\bullet, M_0, (\alpha, \beta))$ be a TPN and $S_\mathcal{T} = (Q, q_0, \rightarrow)$ the semantics of $\mathcal{T}$. The truth value of a formula φ of TPN-TCTL for a state (M, ν) is given on Tab. 1.

$(M,\nu) \models \mathbf{M} \bowtie \bar{V}$	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 $\exists\sigma = (s_0,\nu_0) \longrightarrow^{d_1}_{a_1} \cdots \longrightarrow^{d_n}_{a_n} (s_n,\nu_n) \in [\![\mathcal{T}]\!]$ s.t. $\begin{cases} (s_0,\nu_0) = (M,\nu) \\ \forall i \in [1..n], \forall d \in [0, d_i), (s_i, \nu_i + d) \models \varphi \\ \left(\sum_{i=1}^n d_i\right) \bowtie c \textit{ and } (s_n, v_n) \models \psi \end{cases}$
$(M,\nu) \models \varphi \, \forall\mathcal{U}_{\bowtie c} \, \psi$	iff $\forall\sigma = (s_0,\nu_0) \longrightarrow^{d_1}_{a_1} \cdots \longrightarrow^{d_n}_{a_n} (s_n,\nu_n) \in [\![\mathcal{T}]\!]$ s.t. $\begin{cases} (s_0,\nu_0) = (M,\nu) \\ \forall i \in [1..n], \forall d \in [0, d_i), (s_i, \nu_i + d) \models \varphi \\ \left(\sum_{i=1}^n d_i\right) \bowtie c \textit{ and } (s_n, v_n) \models \psi \end{cases}$

Table 1. Semantics of TPN-TCTL

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

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

We will see that thanks to corollary 3.3, model-checking for TPNs amounts to model-checking for timed automata.

Model-Checking for TPN-TCTL. Let us assume we have to model-check formula φ on a TPN $\mathcal{T}$. Our method consists in using the equivalent timed automata $\Delta(\mathcal{T})$ defined in section 3. For instance, suppose we want to check $\mathcal{T} \models \Box_{\leq 3}(\mathbf{M} \geq (1,2))$. It means that all the states reached within the next 3 time units will have a marking such that p_1 has more than one token and p_2 more than 2. Actually, this is equivalent to checking $\Box_{\leq 3}(SU.0 \rightarrow (p[1] \geq 1 \wedge p[2] \geq 2))$ on the equivalent timed automaton. Notice that $\Diamond_{\leq 3}(\mathbf{M} \geq (1,2))$ reduces to $\Diamond_{\leq 3}(SU.0 \wedge (p[1] \geq 1 \wedge p[2] \geq 2))$. We can then define the translation of a formula in TPN-TCTL to standard TCTL for timed automata.

Definition 4.2 (Translation of TPN-TCTL into TCTL). Let φ be a formula of TPN-TCTL. Then the translation $\Delta(\varphi)$ of φ is inductively defined by:

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

$SU.0$ 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. $\square$

We can now state the following theorem:

Theorem 4.3. *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 $((s,p), \bar{q}, v) = \Delta((M, \nu))$ the equivalent state of $S_{\Delta(\mathcal{T})}$ (i.e. $(M,\nu) \approx ((s,p),\bar{q},v)$). Then*

$$\forall \varphi \in \text{ TPN-TCTL } \quad (M,\nu) \models \varphi \text{ iff } ((s,p),\bar{q},v) \models \Delta(\varphi) \qquad \square$$

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 $[\![\mathcal{T}]\!]$ s.t.:

$$\rho = (s_0, \nu_0) \longrightarrow_{a_1}^{d_1} (s_1, \nu_1) \cdots \longrightarrow_{a_n}^{d_n} (s_n, \nu_n)$$

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 3.3, we conclude that there is a run $\sigma = \Delta(\rho)$ in $[\![S_{\Delta(\mathcal{T})}]\!]$ s.t.

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

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 $(s_n, \nu_n) \models \psi$ iff $((l_n, p_n), \bar{q}_n, v_n)) \models \Delta(\psi)$ and thus conclude that $((l_n, p_n), \bar{q}_n, v_n)) \models \Delta(\psi)$. Moreover as $l_n = 0$ we have $((l_n, p_n), \bar{q}_n, v_n)) \models SU.0 \wedge \Delta(\psi)$. It remains to prove that all intermediate states satisfy $SU.0 \rightarrow \Delta(\varphi)$. 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 $SU.0 \rightarrow \Delta(\psi)$. 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 $\forall i \in [1..n], ((l_i, p_i), \bar{q}_i, v_i)) \approx (s_i, \nu_i)$, 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 3.2, $\forall d \in [0, d_i), ((l_i, p_i), \bar{q}_i, v_i + d)) \approx (s_i, \nu_i + d)$ and applying the induction hypothesis again yields $\forall d \in [0, d_i), ((l_i, p_i), \bar{q}_i, v_i + d)) \models \Delta(\varphi)$. Hence $((l_0, p_0), \bar{q}_0, v_0)) \models (SU.0 \rightarrow \varphi) \, \exists\mathcal{U}_{\bowtie c} \, (SU.0 \wedge \psi)$.

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

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

with

$$((l_n, p_n), \bar{q}_n, v_n)) \models SU.0 \wedge \Delta(\psi) \textit{ and}$$
$$\forall i \in [1..n], \forall d \in [0, d_i), ((l_i, p_i), \bar{q}_i, v_i)) \models (SU.0 \rightarrow \Delta(\varphi))$$

As $((l_n, p_n), \bar{q}_n, v_n)) \models SU.0$, we can use corollary 3.3 and we know there exists a run in $[\![\mathcal{T}]\!]$

$$\rho = \Delta^{-1}(\sigma) = (s_0, \nu_0) \rightarrow_{a_1}^{d_1} (s_1, \nu_1) \cdots \rightarrow_{a_n}^{d_n} (s_n, \nu_n)$$

with $\forall i \in [1..n], ((l_i, p_i), \bar{q}_i, v_i)) \approx (s_i, \nu_i)$. The induction hypothesis on $SU.0 \wedge \Delta(\psi)$ combined with $((l_n, p_n), \bar{q}_n, v_n)) \models \mathcal{S}.0 \wedge \Delta(\psi)$ 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 $(s_0, \nu_0) \models \varphi \, \exists\mathcal{U}_{\bowtie c} \, \psi$. $\square$

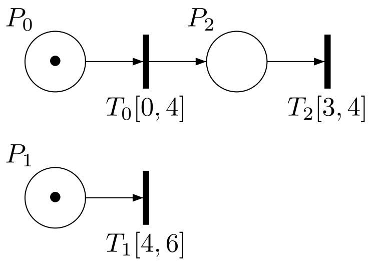

Figure 3. A time Petri net

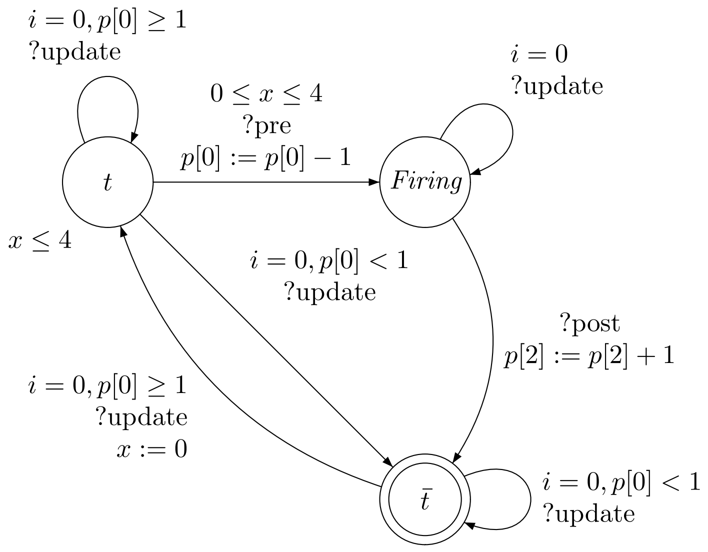

Figure 4. The automaton $\mathcal{A}_0$ for transition t_0

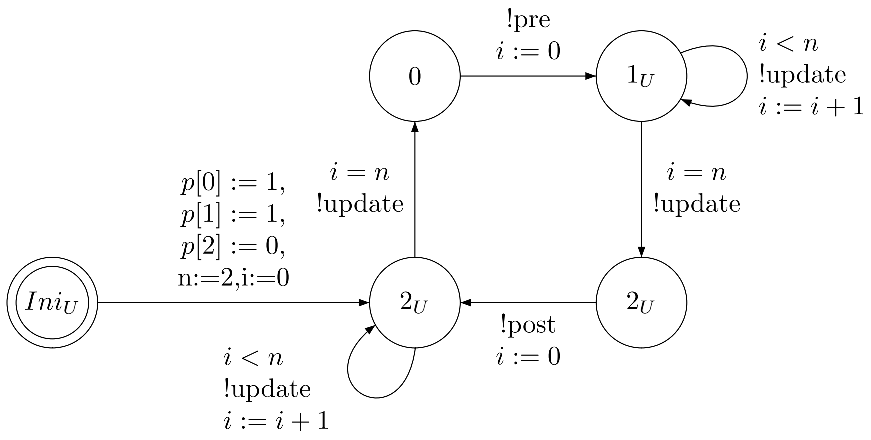

Figure 5. The automaton of the supervisor *SU*

Example of model-checking. If we want to model-check a TPN we obviously have to assume it is bounded. We can ensure this by computing the SCG for instance. Here after we give an example of a TPN and its translation to a timed automaton obtained with our translator [GLR03]. Roméo is a tool for TPNs analysis which can perform the translation of a TPN into the equivalent TA in the UPPAAL input format. It is then possible to check efficiently real-time properties. Let us focus on the TPN of the figure 3 and its translation with Roméo. The translation gives one timed automaton for each transition (see the automaton of the transition t_0 Fig. 4) and one automaton for the supervisor SU (see Fig. 5). These automata update an array of integer $p[i]$ (the number of tokens in place i). The first action of the supervisor gives the initial marking $p[0] = 1, p[1] = 1, p[2] = 0$. UPPAAL accepts only two synchronized actions, then the synchronization function defined in section 3 is implemented with a loop in the supervisor where i is the index of the transition and $n + 1$ the number of transition. The automaton of the transition t_0 is depicted on Fig. 4. As ${}^\bullet t_0 = (1, 0, 0)$ and ${t_0}^\bullet = (0, 0, 1)$, then $p \geq {}^\bullet t_0$ (t_0 is enabled) is implemented by $p[0] \geq 1$; $p := p + {t_0}^\bullet$ is implemented by $p[2] := p[2] + 1$ and so on.

It is now possible to check efficiently properties with UPPAAL in using the equivalent timed automaton. For instance we can check properties as

```
A[] ( ( SU.0 and T0.x>4 ) imply p[0]<1 )
```
or
```
E<> (T2.firing and T1.x<4)
```

The first property means that all the states reached after 4 time units will have a marking such that p_0 has no token. The second property means that it's possible to fire t_2 while t_1 is not firable. Note that this property can't be checked with the SCG that just allows to check that it's possible to fire t_1 after the firing of t_2.

5 Conclusion

In this paper, we have given 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 theoretical results on TA carry on 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 3.2:

- TCTL model checking is decidable for bounded TPNs. Moreover efficient algorithms used in UPPAAL [LPY97, PL00] and KRONOS [Yov97] are exact for TPNs (see recent results [Bou03] by P. Bouyer);
- it is decidable whether a TA is non-zeno or not [HNSY94] and thus our result provides a way to decide non-zenoness for bounded TPNs;
- the controller synthesis problem is partially solvable for TPNs: there is an algorithm [MPS95] for computing the winning states of a Timed Automaton Game and thus we can use it on the timed automaton $\Delta(\mathcal{T})$ associated to a time Petri net $\mathcal{T}$. This way we can compute a controller that is timed automaton for our time Petri net $\mathcal{T}$. This does not prove that there exists a controller for $\mathcal{T}$ that is a time Petri net. In this sense we partially solve the controller synthesis problem for time Petri nets;
- 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 properties on TPNs. For instance, it is possible to check real-time properties expressed in TCTL on bounded TPNs. We have implemented our translation in the tool [GLR03] which can perform the translation of a TPN into the equivalent TA in UPPAAL input format. It is then possible to check efficiently real-time properties with this tool, and to benefit from all the advantages of it: efficiency, counter example generation when a property is false, simulation environment. Also, it bridges a gap between TPNs and TA and for instance enables one to specify a real-time system as a mixture of TPNs and TA, and then derive a TA modeling the behavior of the whole system.

References

ACD93. Rajeev Alur, Costas Courcoubetis, and David Dill. Model-checking in dense real-time. *Information and Computation*, 104(1):2–34, May 1993.

AD94. R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science B*, 126:183–235, 1994.

AFH99. Alur, Fix, and Henzinger. Event-clock automata: A determinizable class of timed automata. *TCS: Theoretical Computer Science*, 211, 1999.

AJ01. Parosh Aziz Abdulla and Bengt Jonsson. Ensuring completeness of symbolic verification methods for infinite-state systems. *Theoretical Computer Science*, 256:145–167, 2001.

AL00. Aura and Lilius. A causal semantics for time petri nets. *TCS: Theoretical Computer Science*, 243, 2000.

AN01. Parosh Aziz Abdulla and Aletta Nylén. Timed petri nets and bqos. In *22nd International Conference on Application and Theory of Petri Nets (ICATPN'01)*, volume 2075 of *Lecture Notes in Computer Science*, pages 53–72, Newcastle upon Tyne, United Kingdom, june 2001. Springer-Verlag.

Arn94. André Arnold. *Finite Transition System.* Prentice Hall, 1994.

BD91. B. Berthomieu and M. Diaz. Modeling and verification of time dependant systems using time petri nets. *IEEE Transactions on Software Engineering*, 17(3):259–273, March 1991.

BDFP00. P. Bouyer, C. Dufourd, E. Fleury, and A. Petit. Are timed automata updatable ? In *Proc. 12th Int. Conf. Computer Aided Verification (CAV'2000), Chicago, IL, USA, July 2000*, volume 1855 of *Lecture Notes in Computer Science*, pages 464–479. Springer, 2000.

BGK$^+$96. J. Bengtsson, W. O. David Griffioen, K. J. Kristoffersen, K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi. Verification of an audio protocol with bus collision using uppaal. In *Proc. of the 8th International Conference on Computer-Aided Verification, LNCS 1102*, pages 244–256, 1996.

BM83. B. Berthomieu and M. Menasche. An enumerative approach for analyzing time Petri nets. In R. E. A. Mason, editor, *Information Processing: proceedings of the IFIP congress 1983*, volume 9 of *IFIP congress series*, pages 41–46. Elsevier Science Publishers, Amsterdam, 1983.

Bou03. Patricia Bouyer. Untameable timed automata! In *Proc. of the 20*th *Annual Symposium on Theoretical Aspects of Computer Science (STACS'2003)*, number 2607 in Lecture Notes in Computer Science. Springer–Verlag, February 2003.

BST98. Sébastien Bornot, Joseph Sifakis, and Stavros Tripakis. Modeling urgency in timed systems. *Lecture Notes in Computer Science*, 1536:103–129, 1998.

BV03. B. Berthomieu and F. Vernadat. State class constructions for branching analysis of time Petri nets. In *Proc. Tools and Algorithms for the Construction and Analysis of Systems (TACAS'2003), Warsaw, Poland, Springer LNCS 2619*, pages 442–457, 2003.

dFERA00. David de Frutos Escrig, Valentín Valero Ruiz, and Olga Marroquín Alonso. Decidability of properties of timed-arc petri nets. In *21st International Conference on Application and Theory of Petri Nets (ICATPN'00)*, volume 1825 of *Lecture Notes in Computer Science*, pages 187–206, Aarhus, Denmark, june 2000. Springer-Verlag.

DS94. M. Diaz and P. Senac. Time stream Petri nets: a model for timed multimedia information. *Lecture Notes in Computer Science*, 815:219–238, 1994.

FS98. Alain Finkel and Philippe Schnoebelen. Fundamental structures in well-structured infinite transitions systems. In *3rd Latin American Theoretical Informatics Symposium (LATIN'98)*, volume 1380 of *Lecture Notes in Computer Science*, pages 102–118, Campinas, Brazil, april 1998. Springer-Verlag.

GLR03. Guillaume Gardey, Didier Lime, and Olivier H. Roux. Roméo: A tool for Time Petri Nets Analysis, 2003. The tool is available at www.irccyn.ec-nantes.fr/irccyn/d/fr/equipes/TempsReel/logs.

HNSY94. Thomas A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2):193–244, June 1994.

KDCD96. Wael Khasa, J.-P. Denat, and S. Collart-Dutilleul. P-Time Petri Nets for manufacturing systems. In *International Workshop on Discrete Event Systems, WODES'96*, pages 94–102, Edinburgh (U.K.), august 1996.

Lil99. Johan Lilius. Efficient state space search for time petri nets. *Electronic Notes in Theoretical Computer Science*, 18, 1999.

LL98. F. Laroussinie and K. G. Larsen. CMC: A tool for compositional model-checking of real-time systems. In *Proc. IFIP Joint Int. Conf. Formal Description Techniques & Protocol Specification, Testing, and Verification (FORTE-PSTV'98)*, pages 439–456. Kluwer Academic Publishers, 1998.

LPY95. Kim G. Larsen, Paul Pettersson, and Wang Yi. Model-Checking for Real-Time Systems. In *Proc. of Fundamentals of Computation Theory*, number 965 in Lecture Notes in Computer Science, pages 62–88, August 1995.

LPY97. K. G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a Nutshell. *Journal of Software Tools for Technology Transfer*, 1(1/2):134–152, October 1997.

LPY98. Magnus Lindahl, Paul Pettersson, and Wang Yi. Formal Design and Analysis of a Gear-Box Controller. In *Proc. of the 4*th *Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, number 1384 in Lecture Notes in Computer Science, pages 281–297. Springer–Verlag, March 1998.

LR03. Didier Lime and Olivier H. Roux. State class timed automaton of a time Petri net. In *10th International Workshop on Petri Nets and Performance Models, (PNPM'03)*. IEEE Computer Society, September 2003.

LRT94. L. Fix, R. Alur, and T.A. Henzinger. A determinizable class of timed automata. In David L. Dill, editor, *sixth International Conference on Computer-Aided Verification CAV*, volume 818 of *Lecture Notes in Computer Science*, pages 1–13, Standford, California, USA, June 1994. Springer-Verlag.

Mer74. P. M. Merlin. *A study of the recoverability of computing systems.* PhD thesis, Department of Information and Computer Science, University of California, Irvine, CA, 1974.

MPS95. O. Maler, A. Pnueli, and J. Sifakis. On the synthesis of discrete controllers for timed systems. In *Proc. 12th Annual Symposium on Theoretical Aspects of Computer Science (STACS'95)*, volume 900, pages 229–242. Springer, 1995.

MY96. O. Maler and S. Yovine. Hardware timing verification using KRONOS. In *Proc. 7th Israeli Conference on Computer Systems and Software Engineering*, Herzliya, Israel, June 1996.

PL00. Paul Pettersson and Kim G. Larsen. UPPAAL2k. *Bulletin of the European Association for Theoretical Computer Science*, 70:40–44, February 2000.

Pop91. Louchka Popova. On time petri nets. *Journal Information Processing and Cybernetics, EIK*, 27(4):227–244, 1991.

PY99. Mauro Pezzé and Michael Young. Time Petri Nets: A Primer Introduction. Tutorial presented at the Multi-Workshop on Formal Methods in Performance Evaluation and Applications, Zaragoza, Spain, september 1999. Available at www.elet.polimi.it/people/pezze.

Ram74. C. Ramchandani. *Analysis of asynchronous concurrent systems by timed Petri nets.* PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 1974. Project MAC Report MAC-TR-120.

Sav01. Alexandru Tiberiu Sava. *Sur la synthèse de la commande des systèmes à évènements discrets temporisés.* PhD thesis, Institut National polytechnique de Grenoble, Grenoble, France, november 2001.

Sif80. J. Sifakis. Performance Evaluation of Systems using Nets. In W. Brauer, editor, *Net theory and applications : Proceedings of the advanced course on general net theory, processes and systems (Hamburg, 1979)*, volume 84 of *Lecture Notes in Computer Science*, pages 307–319. Springer-Verlag, Berlin, 1980.

SY96. Joseph Sifakis and Sergio Yovine. Compositional specification of timed systems (extended abstract). In *13th Annual Symposium on Theoretical Aspects of Computer Science*, volume 1046 of *lncs*, pages 347–359, Grenoble, France, 22–24 February 1996. Springer.

Yov97. S. Yovine. Kronos: A Verification Tool for real-Time Systems. *Journal of Software Tools for Technology Transfer*, 1(1/2):123–133, October 1997.