Structural Translation from Time Petri Nets to Timed Automata

Franck Cassez¹  Olivier-H. Roux²

IRCCyN/CNRS UMR 6597
BP 92101
1 rue de la Noë
44321 Nantes Cedex 3 France

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, UPPAAL or CMC) to analyse TPNs.

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

1 Introduction

Petri Nets with Time. The two main extensions of Petri Nets with time are Time Petri Nets (TPNs) [20] and Timed Petri Nets [24]. 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 [26,22]. The two corresponding subclasses namely P-Timed Petri Nets and T-Timed Petri Nets are expressively equivalent [26,22].

¹ Email: Franck.Cassez@irccyn.ec-nantes.fr
² Email: Olivier-H.Roux@irccyn.ec-nantes.fr
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 [22]. P-TPNs and T-TPNs are proved to be incomparable in [16]. Finally TPNs form a subclass of Time Stream Petri Nets [13] which were introduced to model multimedia applications.

The class T-TPNs is the most commonly-used subclass of TPNs and in this paper we focus on this subclass that will be henceforth referred to as TPN. For classical TPNs, boundedness is undecidable, and works on this model report undecidability results, or decidability under the assumption that the TPN is bounded (e.g. reachability in [23]). Recent work [1,12] consider timed arc Petri nets where each token has a clock representing his “age”. The authors prove that coverability and boundedness are decidable for this class of timed arc Petri nets by applying a backward exploration technique. However, they assume a lazy (non-urgent) behavior of the net: 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. 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 \(\omega = (t_1, d_1)(t_2, d_2) \ldots (t_n, d_n) \ldots\) 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 if there is a timed firing sequence \(\omega\) from the initial marking \(M_0\) to \(M\). Reachability analysis of TPNs relies on the construction of the so-called States Classe Graph (SCG) that was introduced in [5] and later refined in [4]. It has been recently improved in [18] by using partial-order reduction methods.

For bounded TPNs, the SCG construction obviously solves the marking reachability problem (Given a marking \(M\), “Can we 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 sufficient and an alternative graph, the strong state class graph is introduced for this purpose in [6]. The two previous graphs allow for checking LTL properties. Another graph can be constructed that preserves CTL* properties. Anyway none of the previous graphs is a good\(^3\) 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 \(n\)

\(^3\) The 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.
Timed Automata. Timed Automata (TA) were introduced by Alur & Dill [2] and have since been extensively studied. 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 [2,15] for TA and some of their extensions [10]. There also exist several efficient tools like UPPAAL [21], Kronos [29] and Cmc [17] for model-checking TA and many real-time industrial applications have been specified and successfully verified with them.

Related Work. The relationship between TPNs and TA has not been much investigated. In [27] 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 [7] 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 [8] 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 [25] considers bounded TPN 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 [19] 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 TPN. In this article, we consider a structural translation from TPN (not necessary bounded) to TA. 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 analyse the TPN.

**Our Contribution.** We first give a formal semantics for Time Petri Nets [20] in terms of Timed Transition Systems. 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 [2,15] 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, Uppaal or Cmc); ii) allowing strict time constraints on transitions for TPNs preserves the previous result: this leads to an extension of the original TPN model for which TCTL properties can be decided; 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 timed bisimilar to a TPN. We show how it enables us to check for real-time properties expressed in TCTL in Section 4. Finally we conclude with our ongoing work and perspectives in Section 5.
2 Time Petri Nets and Timed Automata

Notations. We denote by $B^A$ 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 $+, -, <$ 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 $A$. 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 $\nu'$ with $\nu'(x) = 0$ for $x \in A'$ and $\nu'(x) = \nu(x)$ otherwise. We denote $\mathcal{C}(X)$ for the simple constraints over a set of variables $X$. $\mathcal{C}(X)$ is defined to be the set of boolean combinations (with the connectives $\{\land, \lor, \neg\}$) of terms of the form $x - x' \geq c$ or $x \geq c$ for $x, x' \in X$ and $c \in \mathbb{N}$ and $\geq \in \{<, \leq, =, \geq, >\}$. Given a formula $\varphi \in \mathcal{C}(X)$ and a valuation $\nu \in A^n$, we denote by $\varphi(\nu)$ the truth value obtained by substituting each occurrence of $x$ in $\varphi$ by $\nu(x)$. For a transition system we write transitions as $s \xrightarrow{a} s'$ and a sequence of transitions of the form $s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} s_n$ as $s_0 \xrightarrow{w} s_n$ with $w = a_1a_2\cdots a_n$.

2.1 Time Petri Nets

The model. Time Petri Nets were introduced in [20] and extend Petri Nets with timing constraints on the firings of transitions.

Definition 2.1 [Time Petri Net] A Time Petri Net $T$ is a tuple $(P, T, \cdot, .^\ast, M_0, (\alpha, \beta))$ where: $P = \{p_1, p_2, \cdots, p_m\}$ is a finite set of places and $T = \{t_1, t_2, \cdots, t_n\}$ is a finite set of transitions; $\cdot, .^\ast \in (\mathbb{N}^P)^T$ is the backward incidence mapping; $\cdot, .^\ast \in (\mathbb{N}^P)^T$ 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.

Semantics of Time Petri Nets. The semantics of TPNs can be given in term of Timed Transition Systems (TTS) 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 $\nu_i$ is the elapsed time since the last time transition $t_i$ was enabled. $0$ is the initial valuation with $\forall i \in [1..n], 0_i = 0$. 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 $M \geq .^\ast t$. The predicate $\uparrow enabled(t_k, M, t_i) \in \mathbb{B}$ 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 [4,3] which is the most common one. In this framework, a transition $t_i$ is newly enabled after firing $t_i$ from marking $M$ if “it is not enabled by $M - .t_i$ and is enabled by $M' = M - .t_i + t_i^\ast$” [4].
Formally this gives:
\[
\uparrow\text{enabled}(t_k, M, t_i) = (M - \bullet t_i + t_i \bullet \geq \bullet t_k) \land ((M - \bullet t_i < \bullet t_k) \lor (t_k = t_i)) \quad (1)
\]

**Definition 2.2** [Semantics of TPN] The semantics of a TPN $T$ is a timed transition system $S_T = (Q, q_0, \rightarrow)$ where: $Q = N^P \times (\mathbb{R}_{\geq 0})^n$, $q_0 = (M_0, 0)$, $\rightarrow \in Q \times (T \cup \mathbb{R}_{\geq 0}) \times Q$ consists of the discrete and continuous transition relations:

- the discrete transition relation is defined $\forall t_i \in T$:
  \[
  (M, \nu) \xrightarrow{t_i} (M', \nu') \text{ iff } \begin{cases}
    M \geq \bullet t_i \land M' = M - \bullet t_i + t_i \bullet \\
    \alpha(t_i) \leq \nu_i \leq \beta(t_i) \\
    \nu'_k = \begin{cases}
      0 & \text{if } \uparrow\text{enabled}(t_k, M, t_i), \\
      \nu_k & \text{otherwise}.
    \end{cases}
  \end{cases}
  \]

- the continuous transition relation is defined $\forall d \in \mathbb{R}_{\geq 0}$:
  \[
  (M, \nu) \xrightarrow{\epsilon(d)} (M, \nu') \text{ iff } \begin{cases}
    \nu' = \nu + d \\
    \forall k \in [1..n], (M \geq \bullet t_k \implies \nu'_k \leq \beta(t_k))
  \end{cases}
  \]

A run of a time Petri net $T$ is a (finite or infinite) path in $S_T$ starting in $q_0$. The set of runs of $T$ is denoted by $[T]$. The set of reachable markings of $T$ is denoted $\text{Reach}(T)$. If the set $\text{Reach}(T)$ is finite we say that $T$ is bounded. As a shorthand we write $(M, \nu) \xrightarrow{\epsilon(d)} (M', \nu')$ for a sequence of time elapsing and discrete steps like $(M, \nu) \xrightarrow{\epsilon(d)} (M'', \nu'') \xrightarrow{\epsilon} (M', \nu')$.

This definition may need some comments. Our semantics is based on the common definition of [4,3] for safe TPNs.

First, previous formal semantics [4,18,22,3] 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\(^4\). 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 [4].

Second, some variations can be found in the literature about TPNs concerning the firing of transitions. The paper [22] 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

\(^4\) If we accept the difference with [18] in the definition of the reset instants for newly enabled transitions.
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 [20,4,22,3].

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 [15] for the TPN using the equivalent timed automaton we build in section 3.

2.2 Timed Automata and Products of Timed Automata

Timed automata [2] are used to model systems which combine discrete and continuous evolutions.

**Definition 2.3** [Timed Automaton] A Timed Automaton $H$ is a tuple $(N, l_0, C, A, E, Inv)$ 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; $E \subseteq N \times C(C) \times A \times 2^X \times N$ is a finite set of edges, $e = (l, \gamma, a, R, l') \in E$ represents an edge from the location $l$ to the location $l'$ with the guard $\gamma$, the label $a$ and the reset set $R \subseteq X$; and $Inv \in C(X)^N$ 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 also a timed transition system.

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

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

$$(l, v) \xrightarrow{e(l)} (l', v') \quad \text{iff} \quad \begin{cases} l = l' \quad v' = v + t \quad \text{and} \\ \forall 0 \leq t' \leq t, \quad Inv(l)(v + t') = \text{tt} \end{cases}$$

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 by $[H]$.

**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.
Let $X = \{x_1, \ldots, x_n\}$ be a set of clocks, $H_1, \ldots, H_n$ 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 $(H_1|\ldots|H_n)_f$ the parallel composition of the $H_i$’s w.r.t. $f$. The configurations of $(H_1|\ldots|H_n)_f$ are pairs $(l, v)$ with $l = (l_1, \ldots, l_n) \in N_1 \times \ldots \times N_n$ and $v = (v_1, \ldots, v_n)$ 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 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 $H_i = (N_i, l_{i,0}, X, A, E_i, Inv_i)$, and $f$ a (partial) synchronization function $(A \cup \{\bullet\})^n \hookrightarrow A$. The semantics of $(H_1|\ldots|H_n)_f$ is a timed transition system $S = (Q, q_0, \rightarrow)$ with $Q = N_1 \times \ldots \times N_n \times (\mathbb{R}_{\geq 0})^X$, $q_0$ is the initial state $((l_{1,0}, \ldots, l_{n,0}), 0)$ and $\rightarrow$ is defined by:

- $(l, v) \xrightarrow{b} (l', v')$ 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 $(l[i], v[i]) \xrightarrow{a_i} (l'[i], v'[i])$.

- $(l, v) \xrightarrow{e(t)} (l, v')$ iff $\forall i \in [1..n]$, we have $(l[i], v[i]) \xrightarrow{e(t)} (l[i], v'[i])$.

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 $(H_1|\ldots|H_n)_f$ 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 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 $T = (P, T, \cdot(\cdot), (\cdot), M_0, (\alpha, \beta))$ with $P = \{p_1, \ldots, p_m\}$ and $T = \{t_1, \ldots, t_n\}$.

**Timed Automaton for one Transition.** We define one timed automaton
\( A_i \) for each transition \( t_i \) of \( T \) (see Fig. 1.a). This timed automaton has one clock \( x_i \). Also the states of the automaton \( A_i \) give the state of the transition \( t_i \): in state \( t \) the transition is enabled; in state \( \overline{t} \) it is disabled and in \textit{Firing} it is being fired. The initial state of each \( A_i \) depends on the initial marking \( M_0 \) of the Petri net we want to translate. If \( M_0 \geq t_i \), then the initial state is \( t \) otherwise it is \( \overline{t} \). 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 \( A_i \)'s. This is not covered by Definition 2.5, which is very often extended ([21]) with integer variables (this does not affect the expressiveness of the model when the variables are bounded).

The Supervisor. The supervisor \( SU \) is depicted on Fig. 1.b. The locations 1 to 3 subscripted with a “c” are assumed to be urgent or committed\(^5\) which means that no time can elapse while visiting them. We denote by \( \Delta(T) = (SU | A_1 | \cdots | A_n)_f \) the timed automaton associated to the TPN \( T \). The supervisor's initial state is 0. Let us define the synchronization function\(^6\) \( f \) with \( n+1 \) parameters defined by:

\[
\begin{align*}
\cdot & f(!pre, \bullet, \cdots, ?pre, \bullet, \cdots) = pre_i \text{ if } ?pre \text{ is the } (i+1)\text{th argument and all the other arguments are } \bullet, \\
\cdot & f(!post, \bullet, \cdots, ?post, \bullet, \cdots) = post_i \text{ if } ?post \text{ is the } (i+1)\text{th argument and all the other arguments are } \bullet, \\
\cdot & f(!update, ?update, \cdots, ?update) = update.
\end{align*}
\]

We will prove in the next subsection that the semantics of \( \Delta(T) \) is closely related to the semantics of \( T \). For this we have to relate the states of \( T \) to the states of \( \Delta(T) \) and we define the following equivalence:

\textbf{Definition 3.1} [State Equivalence] Let \((M, \nu)\) and \(((s, p), q, v)\) be, respectively, a state of \( S_T \) and a configuration\(^7\). Then

\[
(M, \nu) \equiv ((s, p), q, v) \text{ iff } \begin{cases}
  s = 0, \\
  \forall i \in [1..m], \quad p[i] = M(p_i), \\
  \forall k \in [1..n], \quad q[k] = \begin{cases}
    t & \text{if } M \geq t_k, \\
    \overline{t} & \text{otherwise}
  \end{cases}
\end{cases}
\]

\[ \forall k \in [1..n], \quad v[k] = \nu_k \]

\(^5\) In \( SU \), committed locations can be simulated by adding an extra variable: see [28] Appendix A for details.

\(^6\) The first element of the vector refers to the supervisor move.

\(^7\) \((s, p) \in \{0, 1_c, 2_c, 3_c\} \times \mathbb{N}^m\) is the state of \( SU \), \( q \) gives the product location of \( A_1 \times \cdots \times A_n \), and \( v[i], i \in [1..n] \) gives the value of the clock \( x_i \).
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 $S_\mathcal{T} = (Q, q_0, \to)$ its semantics. Let $\mathcal{A}_i$ be the automaton associated with transition $t_i$ of $\mathcal{T}$ as described by Fig. 1.a, $SU$ the supervisor automaton of Fig. 1.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 $S_{\Delta(\mathcal{T})} = (Q_{\Delta(\mathcal{T})}, q_0^{\Delta(\mathcal{T})}, \to)$.

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

1. $$(M, \nu) \xrightarrow{t_i} (M', \nu') \text{ iff } \begin{cases} ((0, p), q, v) \xrightarrow{w_i} ((0, p'), q', v') \text{ with } \\ w_i = \text{pre}_i.update.post_i.update \text{ and } \\ (M', \nu') \approx ((0, p'), q', v') \end{cases} (2)$$
2. $$(M, \nu) \xleftarrow{d} (M', \nu') \text{ iff } \begin{cases} ((0, p), q, v) \xleftarrow{e(d)} ((0, p'), q', v') \text{ and } \\ (M', \nu') \approx ((0, p'), q', v') \end{cases} (3)$$

**Proof.** We first prove statement (2). Assume $(M, \nu) \approx ((0, p), q, v)$. Then as $t_i$ can be fired from $(M, \nu)$ we have: (i) $M \geq t_i$, (ii) $\alpha(t_i) \leq \nu_i \leq \beta(t_i)$, (iii) $M' = M - t_i + t_i^*$, and (iv) $\nu'_k = 0$ if $\nabla enabled(t_k, M, t_i)$ and $\nu'_k = \nu_k$ otherwise. From (i) and (ii) and the state equivalence we deduce that $q[i] = t$ 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, \( l_{\text{pre}} \) is the only possible transition. As the synchronization function \( f \) allows \( \{(l_{\text{pre}}, \bullet, \cdots, ?l_{\text{pre}}, \cdots, \bullet) \) the global action \( l_{\text{pre}} e_i \) is possible. After this move \( \Delta(T) \) reaches state \( ((1, p_1), q_1, v_1) \) such that for all \( k \in [1..n] \), \( q_1[k] = q[k], \forall k \neq i \) and \( q_1[i] = \text{Firing} \). Also \( p_1 = p - t_i \) and \( v_1 = v \).

Now the only possible transition when the supervisor is in state 1 is an update transition where all the \( A_i \)'s synchronize according to \( f \). From \( ((1, p_1), q_1, v_1) \) we reach \( ((2, p_2), q_2, v_2) \) with \( p_2 = p_1, v_2 = v_1 \). For all \( k \in [1..n], k \neq i, q_2[k] = t \) if \( p_1 \geq t_k \) and \( q_2[k] = \bar{t} \) otherwise. Also \( q_2[i] = \text{Firing} \). The next global transition must be a post\(_i \) transition leading to \( ((3, p_3), q_3, v_3) \) with \( p_3 = p_2 + t_i^*, v_3 = v_2 \) and for all \( k \in [1..n], \), \( q_3[k] = q_2[k], \forall k \neq i \) and \( q_3[i] = \bar{t} \).

From this last state only an update transition leading to \( ((0, p_4), q_4, v_4) \) is allowed, with \( p_4 = p_3, v_4 \) and \( q_4 \) given by: for all \( k \in [1..n], q_4[k] = t \) if \( p_3 \geq t_k \) and \( \bar{t} \) otherwise. \( v_4[k] = 0 \) if \( q_3[k] = \bar{t} \) and \( q_4[k] = t \) and \( v_4[k] = v_1[k] \) otherwise. We then just notice that \( q_4[k] = t \) if \( p - t_i < t_k \) and \( q_4[k] = \bar{t} \) if \( p - t_i + t_i^* \geq t_k \). This entails that \( v_4[k] = 0 \) iff \( \text{enabled}(t_k, p, t_i) \) and with (iv) gives \( v'_k = v_4[k] \). As \( p_4 = p_3 = p_2 + t_i^* = p_1 - t_i + t_i^* = p - t_i + t_i^* \) using (iii) we have \( \forall i \in [1..m], M'(p_i) = p_4[i] \). Hence we conclude that \( ((0, p_4), q_4, v_4) \approx (M', \nu') \).

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

We can now state a useful corollary which enables us to do TCTL model-checking for TPNs in the next section. We write \( \Delta((M, \nu)) = ((0, p), q, v) \) if \( (M, \nu) \approx ((0, p), q, v) \). By definition \( \Delta(t_i) = \text{pre}_i, \text{update}, \text{post}_i, \text{update} \) and \( \Delta(e(d)) = e(d) \). Just notice that \( \Delta \) is one-to-one and we can use \( \Delta^{-1} \) as well. Then we extend \( \Delta \) 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 is a four step transition in \( \Delta(T) \)). Again we can extend \( \Delta \) to runs: if \( \rho \in \llbracket T \rrbracket \) we denote \( \Delta(\rho) \) the associated run in \( \llbracket \Delta(T) \rrbracket \). Notice that \( \Delta^{-1} \) is only defined for runs \( \sigma \) of \( \llbracket \Delta(T) \rrbracket \), the last state of which is of the form \( ((0, p), q, v) \) where the supervisor is in state 0. We denote this property \( \text{last}(\sigma) \models SU.0 \).

**Corollary 3.3** \( \rho \models \llbracket T \rrbracket \land \sigma = \Delta(\rho) \) iff \( \sigma \models \llbracket \Delta(T) \rrbracket \land \text{last}(\sigma) \models SU.0 \). \( \square \)
Proof. The proof is a direct consequence of Theorem 3.2. It suffices to notice that all the finite runs of $\Delta(T)$ 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 $w_i = \text{pre}_i \cdot \text{update} \cdot \text{post}_i \cdot \text{update}$, $\delta_i \in \mathbb{R}_{\geq 0}$, and using Theorem 3.2, if $\text{last}(\sigma) \models SU.0$, there exists a corresponding run $\rho$ in $T$ s.t. $\sigma = \Delta(\rho)$. \qed

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 [15] for TPNs. The only difference with the versions of [15] 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 $T = \{t_1, t_2, \cdots, t_m\}$. The temporal logic TPN-TCTL is inductively defined by:

$$\text{TPN-TCTL} ::= M \bowtie \bar{V} | \text{false} | t_{k+c} \leq t_{j+d} | \neg \varphi | \varphi \to \psi | \varphi \exists U_{\bowtie c} \psi | \varphi \forall U_{\bowtie c} \psi$$

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

Intuitively the meaning of $M \bowtie \bar{V}$ 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 $\text{true} = \neg \text{false}$, $\exists U_{\bowtie c} \varphi = \text{true} \exists U_{\bowtie c} \varphi$ and $\forall U_{\bowtie c} = \neg \exists U_{\bowtie c} \neg \varphi$.

The semantics of TPN-TCTL is defined on timed transition systems. Let $T = (P, T, \cdot, ., (.), M_0, (\alpha, \beta))$ be a TPN and $S_T = (Q, q_0, \rightarrow)$ the semantics of $T$. Let $\sigma = (s_0, v_0) \xrightarrow{a_1} \cdots \xrightarrow{a_n} (s_n, v_n) \in \llbracket T \rrbracket$. The truth value of a formula $\varphi$ of TPN-TCTL for a state $(M, \nu)$ is given in Fig. 2.

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

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

---

8 The use of $\infty$ in $\bar{V}$ allows us to handle comparisons like $M(p_1) \leq 2 \land M(p_2) \geq 3$ by writing $M \leq (2, \infty) \land M \geq (0, 3)$. 
\( (M, \nu) \models M \triangleright V \) \text{ iff } \( M \triangleright V \)

\( (M, \nu) \not\models \text{false} \)

\( (M, \nu) \models t_k + c \leq t_j + d \) \text{ iff } \( \nu_k + c \leq \nu_j + d \)

\( (M, \nu) \models \neg \varphi \) \text{ iff } \( (M, \nu) \not\models \varphi \)

\( (M, \nu) \models \varphi \rightarrow \psi \) \text{ iff } \( (M, \nu) \models \varphi \) implies \( (M, \nu) \models \psi \)

\( (M, \nu) \models \varphi \exists U_{\triangleleft} \psi \) \text{ iff } \exists \sigma \in \mathbb{T} \text{ such that }
\[
\begin{cases}
(s_0, \nu_0) = (M, \nu) \\
\forall i \in [1..n], \forall d \in [0, d_i), (s_i, \nu_i + d) \models \varphi \\
(\sum_{i=1}^{n} d_i) \triangleleft c \text{ and } (s_n, \nu_n) \models \psi
\end{cases}
\]

\( (M, \nu) \models \varphi \forall U_{\triangleleft} \psi \) \text{ iff } \forall \sigma \in \mathbb{T} \text{ we have }
\[
\begin{cases}
(s_0, \nu_0) = (M, \nu) \\
\forall i \in [1..n], \forall d \in [0, d_i), (s_i, \nu_i + d) \models \varphi \\
(\sum_{i=1}^{n} d_i) \triangleleft c \text{ and } (s_n, \nu_n) \models \psi
\end{cases}
\]

Fig. 2. Semantics of TPN-TCTL

**Model-Checking for TPN-TCTL.** Let us assume we have to model-check formula \( \varphi \) on a TPN \( T \). Our method consists in using the equivalent timed automaton \( \Delta(T) \) defined in Section 3. For instance, suppose we want to check \( T \models \forall \square_{\leq 3} (M \geq (1, 2)) \). The check 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 \( \forall \square_{\leq 3} (SU.0 \rightarrow (p[1] \geq 1 \land p[2] \geq 2)) \) on the equivalent timed automaton. Notice that \( \exists \Diamond_{\leq 3} (M \geq (1,2)) \) reduces to \( \exists \Diamond_{\leq 3} (SU.0 \land (p[1] \geq 1 \land p[2] \geq 2)) \). 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 \( \varphi \) be a formula of TPN-TCTL. Then the translation \( \Delta(\varphi) \) of \( \varphi \) is inductively defined by:
\[ \Delta(M \triangleright V) = \bigwedge_{i=1}^{n} (p[i] \triangleright V_i) \]

\[ \Delta(\text{false}) = \text{false} \]

\[ \Delta(t_k + c \triangleright t_j + d) = x_k + c \triangleright x_j + d \]

\[ \Delta(\neg \varphi) = \neg \Delta(\varphi) \]

\[ \Delta(\varphi \rightarrow \psi) = SU.0 \land (\Delta(\varphi) \rightarrow \Delta(\psi)) \]

\[ \Delta(\varphi \exists U_{\text{loc}} \psi) = (SU.0 \rightarrow \Delta(\varphi)) \exists U_{\text{loc}} (SU.0 \land \Delta(\psi)) \]

\[ \Delta(\varphi \forall U_{\text{loc}} \psi) = (SU.0 \rightarrow \Delta(\varphi)) \forall U_{\text{loc}} (SU.0 \land \Delta(\psi)) \]

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 4.3** Let \( T \) be a TPN and \( \Delta(T) \) the equivalent timed automaton. Let \( (M, \nu) \) be a state of \( S_T \) and \( ((s, p), q, v) = \Delta((M, \nu)) \) the equivalent state of \( S_{\Delta(T)} \) (i.e. \( (M, \nu) \approx ((s, p), q, v) \)). Then

\[ \forall \varphi \in \text{TPN-TCTL} \quad (M, \nu) \models \varphi \iff ((s, p), q, v) \models \Delta(\varphi) \]

**Proof.** The proof of the theorem can be found in [11].

## 5 Conclusion

In this paper, we have given a structural translation from TPNs to TAs. Any TPN \( T \) and its associated TA \( \Delta(T) \) 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 3.2:

- TCTL model checking is decidable for bounded TPNs. Moreover efficient algorithms used in UPPAAL [21] and KRONOS [29] are exact for TPNs (see recent results [9] by P. Bouyer);
- it is decidable whether a TA is non-zeno or not [15] 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 TAs to check properties on TPNs. For instance, it is possible to check real-time properties expressed in TCTL on bounded TPNs. The tool ROMEO [14] that has been developed for the analysis of TPN (state space computation and “on the
fly” model-checking of reachability properties) implements our translation of a TPN into the equivalent TA in UPPAAL input format. 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. Nevertheless when the TA of a transition is in location $\bar{t}$ we do not need to store the value of the clock. Some of the clocks can then be disregarded. This is known as the active clock reduction in UPPAAL. The current version of UPPAAL computes the active clocks but this is a very expensive step. A future version will feature a syntactical means to declare the active clocks. This will enable us to declare the clock $x_t$ of transition $t$ to be “inactive” in the location $\bar{t}$. When this new version is released we will be able to apply our translation on meaningful case studies.

References


