From Time Petri Nets to Timed Automata

Franck Cassez\textsuperscript{1} Olivier-H. Roux\textsuperscript{2}

IRCCyN/CNRS UMR 6597
BP 92101
1 rue de la Noé
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 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) \cite{30} and Timed Petri Nets \cite{34}. 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 \cite{36,32}. The two corresponding subclasses namely P-Timed Petri Nets and T-Timed Petri Nets are expressively equivalent \cite{36,32}. The same classes are defined for TPNs i.e. T-TPNs and P-TPNs, but both classes of Timed

\textsuperscript{1} Email: Franck.Cassez@irccyn.ec-nantes.fr
\textsuperscript{2} Email: Olivier-H.Roux@irccyn.ec-nantes.fr

\copyright 2004 Published by Elsevier Science B. V.
Petri Nets are included in both P-TPNs and T-TPNs [32]. P-TPNs and T-TPNs are incomparable [20]. Finally TPNs form a subclass of Time Stream Petri Nets [16] 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 [33]). Boundedness and other results are obtained by computing the state-space. Recent work [2,15] consider timed arc Petri nets where each token has a clock representing its “age”. They prove that coverability and boundedness are decidable for this class of Petri nets by applying a backward exploration technique [1,17]. 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 \(\omega\) 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 \(\text{Untimed}(\omega) = t_1t_2 \cdots t_n\).

Reachability analysis of TPNs relies on the construction of the so-called States Classes Graph (SCG) that was introduced in [10] and later refined in [9]. It has been recently improved in [25] 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 [11], 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 [9] 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 \(\text{Untimed}(\omega) = \sigma\). 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
\[ L = \text{Untimed}(L'). \] Henceforth the SCG can only be used to check untimed reachability properties\(^3\) but is not accurate enough for checking \textit{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 [4] and have since been extensively studied [21,24,3,13]. This model is an extension of finite automata with (dense time) \textit{clocks} and enables one to specify real-time systems. It has been shown that model-checking for TCTL properties is decidable [5,19] for TA and some extensions of TA [13]. There also exist several efficient tools like UPPAAL [23,31], KRONOS [38] and CMC [22] for model-checking TA and many real-time industrial applications have been specified and successfully verified with them [8,29,27].

**Related Work.**

The relationship between TPNs and TA has not been often investigated. In [37] J. Sifakis and S. Yovine are mainly concerned with \textit{compositional} 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 [12] 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 [35] 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 \textit{a priori}) and when it is bounded, they do not prove that the algorithm stop. Lime and Roux [26] 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

---

\(^3\) The use of \textit{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.
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 focused 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 \([30]\) in terms of Timed Transition Systems \([24]\). 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 \([5,19]\) 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 \(+,-,\langle,\rangle,=\) 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 \(\nu'\) with \(\nu'(x) = 0\) for \(x \in A'\) and \(\nu'(x) = \nu(x)\) otherwise. We denote \(C(V)\) for the simple constraints over a set of variables \(V\). \(C(V)\) is defined to be the set of boolean combinations (with the connectives \{\&\&\,\|\|\,\neg\}) of terms of the form
\[ v - v' \succ c \text{ or } v \succ c \text{ for } v, v' \in V \text{ and } c \in \mathbb{N} \text{ and } \in \{ <, \leq, =, \geq, > \}. \text{ For a} \text{ transition system we write transitions as } s \xrightarrow{a} s' \text{ and sequence of transitions of the form } s_0 \xrightarrow{a_1} s_1 \to \cdots \xrightarrow{a_n} s_n \text{ as } s_0 \xRightarrow{w} s_n \text{ with } w = a_1a_2 \cdots a_n. \]

2.1 Time Petri Nets

The model.

Time Petri Nets were introduced in [30] and extend Petri Nets with timing constraints on the firings of transitions.

**Definition 2.1** A Time Petri Net \( T \) is a tuple \((P, T, \cdot,\cdot, M_0, (\alpha, \beta))\) where:

- \( P = \{p_1, p_2, \ldots, p_m\} \) is a finite set of places,
- \( T = \{t_1, t_2, \ldots, t_n\} \) is a finite set of transitions,
- \( \cdot,\cdot \in (\mathbb{N}^P)^T \) is the backward incidence mapping,
- \( \cdot,\cdot \in (\mathbb{N}^P)^T \) 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) [24] 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. \( \overline{0} \) 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 \( M \geq \cdot t \). \( \uparrow \text{enabled}(t_k, M, t_i) \in \mathbb{B} \) is true if \( t_k \) is enabled by the firing of transition \( t_i \) from marking \( M \), and false otherwise. This definition of enabledness is based on [9,7] 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 \( M - \cdot t_i \) and is enabled by \( M' = M - \cdot t_i + t_i \cdot \)” [9].

Formally this gives:

\[ \uparrow \text{enabled}(t_k, M, t_i) = (M - \cdot t_i + t_i \cdot \geq t_k) \land \left( (M - \cdot t_i < t_k) \lor (t_k = t_i) \right) \quad (1) \]

**Definition 2.2** The semantics of a TPN \( T \) is a timed transition system \( S_T = (Q, q_0, \rightarrow) \) where:

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

\[
(M, \nu) \xrightarrow{t_i} (M', \nu') \iff \begin{cases}
M \geq \bullet t_i \land M' = M - \bullet t_i + t_i^* \\
\alpha(t_i) \leq \nu_i \leq \beta(t_i) \\
\nu'_k = \begin{cases}
0 & \text{if } \uparrow \text{enabled}(t_k, M, t_i) \\
\nu_k & \text{otherwise}.
\end{cases}
\end{cases}
\]

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

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

A run of a time Petri net \( T \) is a path in \( S_T \) starting in \( q_0 \). The set of runs of \( T \) is denoted \([ T ]\). 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 \( (M, \nu) \xrightarrow{c(d)} (M', \nu') \) for a sequence of time elapsing and discrete steps like \( (M, \nu) \xrightarrow{c(d)} (M'', \nu'') \xrightarrow{c} (M', \nu') \).

This definition appeals for several comments. Our semantics is based on the common definition of [9,7] for safe TPNs.

First, previous formal semantics [9,25,32,7] 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 [9].

Second, some variations can be found in the literature about TPNs concerning the firing of transitions. [32] 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 [30,9,32,7].

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

\(^4\) If we except the difference with [25] in the definition of the reset instants for newly enabled transitions.
2.2 Timed Automata and Products of Timed Automata

Timed automata \cite{4} 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,
- $E \subseteq N \times C(V) \times A \times 2^{C \times N}$ 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 $R$.
- $Inv \in C(V)^N$ 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 $S_H = (\mathcal{Q}, q_0, \rightarrow)$ with $\mathcal{Q} = N \times (\mathbb{R} \geq 0)^C$, $q_0 = (l_0, \mathbf{0})$ is the initial state and $\rightarrow$ is defined by:

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

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

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 $[H]$. \hfill \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 \cite{6}. 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 $(A \cup \{\bullet\})^n \rightarrow A$ 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 $({\vec{l}}, v)$ with ${\vec{l}} = (l_1, \ldots, l_n) \in N_1 \times \ldots \times N_n$ and $v = v_1 \cdots v_n$ with $v_i$ is the restriction of $v$ to $C_i$.\hfill 5
(\mathbb{R}_{\geq 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 \((A \cup \{\cdot\})^n \hookrightarrow A\). The semantics of \((H_1|\ldots|H_n)_f\) is a timed transition system \( S = (Q, q_0, \rightarrow) \) with \( Q = N_1 \times \ldots \times N_n \times (\mathbb{R}_{\geq 0})^C \), \( q_0 \) is the initial state \(((l_{1,0}, \ldots, l_{n,0}), \overline{0})\) and \( \rightarrow \) is defined by:

- \((\overline{l}, v) \xrightarrow{b} (\overline{l'}, v')\) iff there exists \((a_1, \ldots, a_n) \in (A \cup \{\cdot\})^n\) s.t. \( f(a_1, \ldots, a_n) = b \) and for any \( i \) we have:
  - If \( a_i = \cdot \), then \( l_i' = l_i \) and \( v_i' = v_i \).
  - If \( a_i \in A \), then \((l_i, v_i) \xrightarrow{a_i} (l_i', v_i')\).

- \((\overline{l}, v) \xrightarrow{\epsilon(t)} (\overline{l}, v')\) iff \( \forall i \in [1..n] \), we have \((l_i, v_i) \xrightarrow{\epsilon(t)} (l_i', v_i')\).

We could equivalently define the product of \( n \) timed automata syntactically, building a new timed automaton \([24]\) 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 \( T = (P, T, \cdot(.), (.)^*, M_0, (\alpha, \beta)) \) 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). 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 \( M_0 \geq \cdot t_i \) 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 the definition 2.5 but this one is very often extended ([23,31]) 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\(^6\) 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:

\begin{itemize}
  \item $f(!pre, \bullet, \ldots, ?pre, \bullet, \ldots) = pre\_i$ if $?pre$ is the $(i+1)$th argument and all the other arguments are $\bullet$,
  \item $f(!post, \bullet, \ldots, ?post, \bullet, \ldots) = post\_i$ if $?post$ is the $(i+1)$th argument and all the other arguments are $\bullet$,
  \item $f(!update, ?update, ?update, \ldots) = \alpha(t_i) 
  x_i = \beta(t_i) 
  p < \bullet t_i 
  p := p - t_i$
  \end{itemize}

Fig. 1. The automaton $A_i$ for transition $t_i$

\begin{itemize}
  \item $p \geq \bullet t_i$
  \item $p := p + t_i \bullet$
  \item $x_i := 0$
  \item $?update$
  \item $p \geq \bullet t_i$
  \item $p < \bullet t_i$
  \item $?update$
  \item $?pre$
  \item $p := p - t_i \bullet$
  \item $x_i \leq \beta(t_i)$
  \item $?update$
  \item $p < \bullet t_i$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
  \item $?update$
Definition 3.1 [States Equivalence] Let \((M, \nu)\) and \(((s, p), \bar{q}, v)\) be respectively a state of \(S_T\) and a configuration\(^7\) of \(S_{\Delta(T)}\). Then

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

\(\square\)

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 \(T\) and \(S_T = (Q, q_0, \rightarrow)\) its semantics. Let \(A_i\) be the automaton associated with transition \(t_i\) of \(T\) as described by Fig. 1, \(SU\) the supervisor automaton of Fig. 2 and \(f\) the synchronization function defined previously. The semantics of \(\Delta(T) = (SU \times A_1 \times \cdots \times A_n)_f\) is the TTS \(S_{\Delta(T)} = (Q_{\Delta(T)}, \epsilon_{\Delta(T)}, \rightarrow)\).

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

\[
(M, \nu) \xrightarrow{t_i} (M', \nu') \iff \begin{cases} 
  ((0, p), \bar{q}, v) \xrightarrow{w_i} ((0, p'), \bar{q}', v') \text{ with } \\
  (M', \nu') \approx ((0, p'), \bar{q}', v') 
\end{cases} \quad (2)
\]

\[
(M, \nu) \xrightarrow{c(d)} (M', \nu') \iff \begin{cases} 
  ((0, p), \bar{q}, v) \xrightarrow{c(d)} ((0, p'), \bar{q}', v') \text{ and } \\
  (M', \nu') \approx ((0, p'), \bar{q}', v') 
\end{cases} \quad (3)
\]

\(\square\)

Proof. We first prove statement 2. Assume \((M, \nu) \approx ((0, p), \bar{q}, v)\). Then as \(t_i\) can be fired from \((M, \nu)\) we have: (i) \(M \geq \bullet t_i\) (ii) \(\alpha(t_i) \leq \nu_i \leq \beta(t_i)\) (iii) \(M' = M - \bullet t_i + \bullet t_i\) (iv) \(\nu_k = 0\) if \(\uparrow \text{enabled}(t_k, M, t_i)\) and \(\nu_k' = \nu_k\) otherwise. From (i) and (ii) and the state equivalence we deduce that \(\bar{q}_i = t\) and \(\alpha(t_i) \leq \nu_i \leq \beta(t_i)\). Hence \(!pre\) is enabled in \(A_i\). In state 0 for the supervisor, \(!pre\) is the only possible transition. As the synchronization function \(f\) allows \(!pre, \bullet, \cdots, \bullet\) the global action \(!pre\) is possible. After this move \(\Delta(T)\) reaches state \(((1, p_1), q_1, v_1)\) with \(q_k = \bar{q}_k, \forall k \neq i\) and \(q_{1i} = \text{Firing}\). 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 \(A_i\)'s synchronize according to \(f\). From

\(\cdots\)
((1, p1), q1, v1) we reach ((2, p2), q2, v2) with p2 = p1, v2 = v1. For k ≠ i, q2k = if p1 ≥ t k and q2k = t otherwise and q2i = Firing.

The next global transition must be a post transition leading to ((3, p3), q3, v3) with p3 = p2 + t i, v3 = v2 and q3k = q2k, ∀ k ≠ i and q3i = i.

From this last state only an update transition leading to ((0, p4), q4, v4) is allowed, with p4 = p3, v4 and q4 given by: q4k = t if p3 ≥ t k and q4k = t otherwise. v4k = 0 if q3k = t and q4k = t and v4k = v1k otherwise. We then just notice that q4k = i if p − t k < t k and q4k = t if p − t i + t i ≥ t k. This entails that v4k = 0 iff enabled(t k, p, t i) and with (iv) gives ν′ k = v4k. As p4 = p3 = p2 + t i = p2 + t i = p − t i + t i using (iii) we have

We now focus on statement 3. According to the semantics of TPNs, a continuous transition (M, ν) −→ (M′, ν′) is allowed iff ν = ν′ + d and ∀k ∈ [1..n], M ≥ t k → ν′ ≤ β(t k)). From the states equivalence (M, ν) ≈ (0, p), q, v), if M ≥ t k then qk = t and the continuous evolution for A k is constrained by the invariant x k ≤ β(t k). Otherwise q k = i 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 in the next section to do TCTL model-checking for TPNs. We denote Δ((M, ν)) = ((0, p), q, v) if (M, ν) ≈ (0, p), q, v) and Δ(t i) = prei.update.posti.update and Δ(e(d)) = 1. Just notice that Δ is one-to-one and we can use Δ−1 as well. Then we extend Δ to transitions as: Δ((M, ν) −→ (M′, ν′)) = Δ((M, ν)) −→ Δ((M′, ν′)) with e ∈ T ∪ [0..) (as Δ(t i) is a word, this transition might be a 4 step transition). Again we can extend Δ to runs: if ρ ∈ T we denote Δ(ρ) the associated run in [Δ(T)]. Notice that Δ−1 is only defined for runs σ of [Δ(T)] the last state of which is of the form (0, p), q, v) where the supervisor is in state 0. We denote this property last(σ) = SU.0.

Corollary 3.3 (ρ ∈ T ∧ σ = Δ(ρ)) if (σ ∈ [Δ(T)] ∧ last(σ) = SU.0).

Proof. The proof is a direct consequence of theorem 3.2. It suffices to notice that all the runs of Δ(T) are of the form

σ = (s0, v0) −→ t w1 (s1, v1) . . . −→ t wn (sn, vn)

with wi = prei.update.posti.update and consequently using theorem 3.2, if the last state satisfy last(σ) = SU.0, there exists a corresponding run ρ in T s.t. σ = Δ(ρ).

Consequently all the runs in Δ(T) ending in SU.0 have a corresponding run in 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 [5.19] for TPNs. The only difference with the versions of [5.19] 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:

\[
TPN-TCTL ::= M \triangleright \hat{V} | \text{false} | t_k+c \leq t_j+d | \neg \varphi | \varphi \rightarrow \psi | \varphi \exists U \triangleright c \psi | \varphi \forall U \triangleright c \psi
\]

where \( M \) and false are keywords, \( \varphi, \psi \in TPN-TCTL \), \( t_k, t_j \in T \), \( c, d \in \mathbb{Z} \), \( \hat{V} \in (Q \cup \{\infty\})^n \) and\(^8\) \( \triangleright \in \{<, \leq, \geq, =, >\} \).

Intuitively the meaning of \( M \triangleright \hat{V} \) is that the current marking vector is in relation \( \triangleright \) with \( \hat{V} \). The meaning of the other operators is the usual one. We use the usual shorthands \( \text{true} = \neg \text{false}, \diamond \triangleright c \phi = \text{true} \exists U \triangleright c \phi \) and \( \square \triangleright c = \neg \diamond \triangleright c \neg \phi \).

The semantics of TPN-TCTL is defined on timed transition systems. Let \( T = (P, T, \cdot, \cdot, M_0, (\alpha, \beta)) \) be a TPN and \( S_T = (Q, q_0, \rightarrow) \) the semantics of \( T \). The truth value of a formula \( \varphi \) of TPN-TCTL for a state \( (M, \nu) \) is given on Tab. 1.

The TPN \( T \) satisfies the formula \( \varphi \) of TPN-TCTL which is denoted \( T \models \varphi \) iff the first state of \( S_T \) satisfies \( \varphi \) 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 \( \varphi \) on a TPN \( T \). Our method consists in using the equivalent timed automata \( \Delta(T) \) defined in section 3. For instance, suppose we want to check \( T \models \square_{\leq 3}(M \geq (1, 2)) \). 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 \( \square_{\leq 3}(SU.0 \rightarrow (p[1] \geq 1 \land p[2] \geq 2)) \) on the equivalent timed automaton. Notice that \( \diamond_{\leq 3}(M \geq (1, 2)) \) reduces to \( \diamond_{\leq 3}(SU.0 \land (p[1] \geq 1 \land p[2] \geq 2)) \). We can then define the translation of a formula in TPN-TCTL to standard TCTL for timed automata.

\(^8\) the use of \( \infty \) in \( \hat{V} \) 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, ν) |= M ⊳⟨ V  
(M, ν) \not\models \text{false}  
(M, ν) |= t_k + c \leq t_j + d  
(M, ν) |= \neg \varphi  
(M, ν) |= \varphi \rightarrow \psi  
(M, ν) |= \varphi \exists U_{\text{loc}} \psi  
(M, ν) |= \varphi \forall U_{\text{loc}} \psi

\Delta(M, \nu) = \bigwedge_{i=1}^{n} (p[i] \models V_i)

\Delta(\text{false}) = \text{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 \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.

We can now state the following theorem:

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

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

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

**Only if part.**

Assume $(M, \nu) \models \varphi \exists \mathcal{U}_{\infty} \psi$. Then by definition, there is a run $\rho$ in $[T]$ s.t.:

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

and $(s_0, \nu_0) = (M, \nu)$, $\sum_{i=1}^{n} d_i = c$, $\forall i \in [1..n], \forall d \in [0, d_i), (s_i, \nu_i + d) \models \varphi$ and $(s_n, \nu_n) \models \psi$. With corollary 3.3, we conclude that there is a run $\sigma = \Delta(\rho)$ in $[S_{\Delta(T)}]$ s.t.

$$\sigma = ((l_0, p_0), \bar{q}_0, \nu_0)) \rightarrow_{d_1} ((l_1, p_1), \bar{q}_1, \nu_1)) \cdots \rightarrow_{d_n} ((l_n, p_n), \bar{q}_n, \nu_n))$$

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 $(s_n, \nu_n) \approx ((l_n, p_n), \bar{q}_n, \nu_n))$, using the induction hypothesis on $\psi$, we can assume $(s_n, \nu_n) \models \psi$ iff $((l_n, p_n), \bar{q}_n, \nu_n)) \models \Delta(\psi)$ and thus conclude that $((l_n, p_n), \bar{q}_n, \nu_n)) \models \Delta(\psi)$. Moreover as $l_n = 0$ we have $((l_n, p_n), \bar{q}_n, \nu_n)) \models SU.0 \wedge \Delta(\psi)$. It remains to prove that all intermediate states satisfy $SU.0 \rightarrow \Delta(\varphi)$. Just notice that all the intermediate states in $\sigma$ 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 $\varphi$, 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 $((l_0, p_0), \bar{q}_0, \nu_0)) \models (SU.0 \rightarrow \varphi) \exists \mathcal{U}_{\infty}(SU.0 \wedge \psi)$.

**If part.**

Assume $((l_0, p_0), \bar{q}_0, \nu_0)) \models (SU.0 \rightarrow \Delta(\varphi)) \exists \mathcal{U}_{\infty}(SU.0 \wedge \Delta(\psi))$. Then there is a run

$$\sigma = ((l_0, p_0), \bar{q}_0, \nu_0)) \rightarrow_{d_1} ((l_1, p_1), \bar{q}_1, \nu_1)) \cdots \rightarrow_{d_n} ((l_n, p_n), \bar{q}_n, \nu_n))$$

with

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

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

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

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

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 $((l_n, p_n), \bar{q}_n, \nu_n)) \models SU.0 \wedge \Delta(\psi)$ implies $(s_n, \nu_n) \models \psi$. For
all the intermediate states of $\rho$ we also apply the induction hypothesis: each $((l_i, p_i), \bar{q}_i, v_i))$ is equivalent to $(s_i, \nu_i)$ and all the states $(s_i, \nu_i + d), d \in [0, d_i)$ satisfy $\varphi$. Hence $(s_0, \nu_0) \models \varphi \exists U_{\text{loc}} \psi$. □ □

Example of model-checking.

15

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. Hereafter we give an example of a TPN and its translation to a timed automaton obtained with our translator [18]. 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 $t_0^* = (1, 0, 0)$ and $t_0^* = (0, 0, 1)$, then $p \geq t_0^*$ ($t_0$ is enabled) is implemented by $p[0] \geq 1; p := p + t_0^*$ 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 \text{ and } T0.x>4) \implies p[0]<1 \) \]
\text{ or }
\[ E<> (T2.firing \text{ and } T1.x<4) \]

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 fireable. 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 \( T \) and its associated TA \( \Delta(T) \) 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 [23,31] and KRONOS [38] are exact for TPNs (see recent results [14] by P. Bouyer);
- it is decidable whether a TA is non-zeno or not [19] 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 [28] for computing the winning states of a Timed Automaton Game and thus we can use it on the timed automaton \( \Delta(T) \) associated to a time Petri net \( T \). This way we can compute a controller that is timed automaton for our time Petri net \( T \). This does not prove that there exists
a controller for $T$ 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 [18] 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

[1] Parosh Aziz Abdulla and Bengt Jonsson. Ensuring completeness of symbolic
verification methods for infinite-state systems. Theoretical Computer Science,

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.


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.

systems using time petri nets. IEEE Transactions on Software Engineering,


