TLP **6** (3): 301–320, 2006. © 2006 Cambridge University Press
doi:10.1017/S147106840600264X Printed in the United Kingdom

State space computation and analysis of Time Petri Nets

GUILLAUME GARDEY, OLIVIER H. ROUX and OLIVIER F. ROUX

IRCCyN (Institut de Recherche en Communication et Cybernétique de Nantes), UMR CNRS 6597, Université de Nantes, École Centrale de Nantes, École des Mines de Nantes, CNRS, 1, rue de la Noë, B.P. 92101 – 44321 Nantes cedex 3, France
(*e-mail:* {guillaume.gardey,olivier-h.roux,olivier.roux}@irccyn.ec-nantes.fr)

submitted 17 November 2003; revised 13 September 2004; accepted 10 May 2005

Abstract

The theory of *Petri Nets* provides a general framework to specify the behaviors of real-time reactive systems and *Time Petri Nets* were introduced to take also temporal specifications into account. We present in this paper a forward zone-based algorithm to compute the state space of a bounded Time Petri Net: the method is different and more efficient than the classical State Class Graph. We prove the algorithm to be exact with respect to the reachability problem. Furthermore, we propose a translation of the computed state space into a Timed Automaton, proved to be timed bisimilar to the original Time Petri Net. As the method produce a single Timed Automaton, syntactical clocks reduction methods (DAWS and YOVINE for instance) may be applied to produce an automaton with fewer clocks. Then, our method allows to model-check T-TPN by the use of efficient Timed Automata tools.

KEYWORDS: Time Petri nets, timed automata, bisimulation, reachability analysis, zones

1 Introduction

Framework

The theory of Petri Nets provides a general framework to specify the behaviors of real-time reactive systems and time extensions were introduced to take also temporal specifications into account. The two main time extensions of Petri Nets are Time Petri Nets (TPN) (Merlin 1974) and Timed Petri Nets (Ramchandani 1974). While a transition can be fired within a given interval for TPN, in Timed Petri Nets, transitions are fired as soon as possible. There are also numerous ways of representing time. TPN are mainly divided in P-TPN, A-TPN and T-TPN where a time interval is relative to places (P-TPN), arcs (A-TPN) or transitions (T-TPN). Finally, Time Stream Petri Nets (Diaz and Senac 1994) were introduced to model multimedia applications.

Concerning the timing analysis of these three models ((T,P,A)–TPN), few studies have been realized about model-checking.

https://doi.org/10.1017/S147106840600264X Published online by Cambridge University Press

Recent works (Abdulla and Nylén 2001; de Frutos Escrig *et al.* 2000) consider Timed Arc Petri Nets where each token has a clock representing its "age". Using a backward exploration algorithm (Abdulla and Jonsson 1998; Finkel and Schnoebelen 1998), it is proved that the coverability and boundedness are decidable for this class of Petri Nets. However, they assume a lazy (non-urgent) behavior of the net: the firing of a transition may be delayed even if its clock's value becomes greater than its latest firing time, disabling the transition.

Rokicki (1993) and Rokicki and Myers (1994) consider an extension of labeled Petri Nets called Orbitals Nets: each transition of the TPN (safe P-TPN) is labeled with a set of events (actions). The state space is built using a forward algorithm very similar to Alur and Dill region based method. Rokicki finally uses partial order method to reduce time and space requirements for verification purpose. The semantics used is not formally defined and seems to differ from another commonly adopted proposed by Khansa *et al.* (1996) for P-TPN.

In this paper, we consider T-TPN in which a transition can be fired within a time interval. For this model, boundedness is undecidable and works report undecidability results, or decidability under the assumption of boundedness of the T-TPN (as for reachability, decidability (Popova 1991)).

Related work

State Space Computation of a T-TPN. The main method to compute the state space of a T-TPN is the State Class Graph (Menasche 1982; Berthomieu and Diaz 1991). A class C of a T-TPN is a pair (M, D) where M is a marking and D a set of inequalities called the firing domain. The variable x_i of the firing domain represents the firing time of the enabled transition t_i relatively to the time when the class C was entered in and truncated to nonnegative times. The State Class Graph preserves markings (Berthomieu and Vernadat 2003) as well as traces and complete traces but can only be used to check untimed reachability properties and is not accurate enough for checking *quantitative* real-time properties. An alternative approach has been proposed by Yoneda and Ryuba (1998) in the form of an extension of equivalence classes (atomic classes) which allow CTL model-checking. Lilius (1999) refined this approach so that it becomes possible to apply partial order reduction techniques that have been developed for untimed systems. Berthomieu and Vernadat (2003) propose an alternative construction of the graph of atomic classes of Yoneda applicable to a larger class of nets. Okawa and Yoneda (1997) propose another method to perform CTL model-checking on T-TPN , they use a region based algorithm on safe T-TPN without ∞ as latest firing time. Their algorithm is based on the one of Alur and Dill (1994), and aims at computing a graph preserving branching properties. Nevertheless, the algorithm used to construct the graph seems inefficient (their algorithm do code regions) and no result can be exploited to compare with other methods.

From T-TPN to TA. Several approaches aim at translating a Time Petri Net into a Timed Automaton to use efficient existent tools on TA. Cortés *et al.* (2000) propose to transform an extension of T-TPN into the composition of several TA. Each

https://doi.org/10.1017/S147106840600264X Published online by Cambridge University Press

transition is translated into an automaton (not necessarily identical due to conflict problems) and it is claimed that the composition captures the behavior of the T-TPN. Cassez and Roux (2004) propose another structural approach: each transition is translated into a TA using the same pattern. The authors prove the two models are timed bisimilar. Sava and Alla (2001) compute the graph of reachable markings of a T-TPN. The result is a TA. However, they assume the T-TPN is bounded and does not include ∞ as latest firing time. No proof is given of the timed bisimilarity between the two models. Lime and Roux (2003) propose a method for building the State Class Graph of a bounded T-TPN as a TA. They prove the T-TPN to be timed bisimilar to the generated TA.

Considering the translation of T-TPN into TA, to study model's properties, raises the problem of the model-checking feasibility of the resulting TA. The model-checking complexity on TA is exponential in the number of clocks of the TA. The proposed transformation in Cassez and Roux (2004) and Cortés *et al.* (2000) is to build as many TA as the number of transitions of the T-TPN. Consequently, there are as many clocks as in the initial T-TPN. It has also to be considered that the reduction method (Daws and Yovine 1996) cannot be applied to the resulting TA: the parallel composition has to be computed first. Nevertheless, the construction of TA is straightforward and linear in the number of transitions of the T-TPN. Concerning the method in Lime and Roux (2003), the resulting TA has a lower number of clocks. The method we propose produces an automaton with more clocks than the previous method but its computation is faster.

Such translations show that TCTL and CTL are decidable for bounded T-TPN and that developed algorithms on TA may be extended to T-TPN.

Contributions

This paper is devoted to presenting an alternative approach to the state space construction of a T-TPN. The method is mainly based upon the region graph algorithm of Alur and Dill on Timed Automaton. We propose to use a derived method using zones to compute the state space of the T-TPN. The algorithm is proved to be exact with respect to the reachability problem and we propose to translate the state space it computes into a Timed Automaton, bringing so the power of TA model-cheking algorithms to T-TPN.

We first recall the semantics of T-TPN and present a forward zone-based algorithm that computes the state space of a T-TPN. Next, we present the labeling of the state space that produces a TA we proved to be timed bisimilar to the original T-TPN. We then compare our method to other used methods on T-TPN and show its advantages. Finally, some applications are proposed.

2 Time Petri Nets

2.1 Definitions

Time Petri Nets (T-TPN) are a time extension of classical Petri Nets. Informally, with each transition of the Net is associated a clock and a time interval. The clock

https://doi.org/10.1017/S147106840600264X Published online by Cambridge University Press

measures the time since the transition has been enabled and the time interval is interpreted as a firing condition: the transition may fire if the value of its clock belongs to the time interval. Formally:

Definition 1 (*T-TPN*)
A Time Petri Net is a tuple $(P, T, {}^\bullet(.), (.)^\bullet, \alpha, \beta, M_0)$ defined by:

- $P = \{p_1, p_2, \ldots, p_m\}$ is a non-empty set of places,
- $T = \{t_1, t_2, \ldots, t_n\}$ is a non-empty set of transitions,
- ${}^\bullet(.) : T \to \mathbb{N}^P$ is the backward incidence function,
- $(.)^\bullet : T \to \mathbb{N}^P$ is the forward incidence function,
- $M_0 \in \mathbb{N}^P$ is the initial marking of the Petri Net,
- $\alpha : T \to \mathbb{Q}_{\geqslant 0}$ is the function giving the earliest firing times of transitions,
- $\beta : T \to \mathbb{Q}_{\geqslant 0} \cup \{\infty\}$ is the function giving the latest firing times of transitions.

A Petri Net marking M is an element of $\mathbb{N}^P$ such that for all $p \in P$, $M(p)$ is the number of tokens in the place p.

A marking M enables a transition t if: $M \geqslant {}^\bullet t_i$. The set of transitions enabled by a marking M is *enabled*(M).

A transition t_k is said to be *newly* enabled by the firing of a transition t_i if $M - {}^\bullet t_i + t_i^\bullet$ enables t_k and $M - {}^\bullet t_i$ did not enable t_k. If t_i remains enabled after its firing then t_i is newly enabled. The set of transitions newly enabled by a transition t_i for a marking M is noted $\uparrow enabled(M, t_i)$.

$v \in (\mathbb{R}_{\geqslant 0})^T$ is a valuation of the system. v_i is the time elapsed since the transition t_i has been newly enabled.

The semantics of T-TPN is defined as a Timed Transition Systems (TTS). Firing a transition is a discrete transition of the TTS, waiting in a marking, the continuous transition.

Definition 2 (*Semantics of a T-TPN*)
The semantics of a T-TPN is defined by the Timed Transition System $\mathcal{S} = (Q, q_0, \to)$:

- $Q = \mathbb{N}^P \times (\mathbb{R}_{\geqslant 0})^T$
- $q_0 = (M_0, \bar{0})$
- $\to \in Q \times (T \cup \mathbb{R}_{\geqslant 0}) \times Q$ is the transition relation including a discrete transition and a continuous transition.
  - The continuous transition is defined $\forall d \in \mathbb{R}_{\geqslant 0}$ by:
    $$(M, v) \xrightarrow{e(d)} (M, v') \text{ iff } \begin{cases} v' = v + d \\ \forall k \in [1, n]\ M \geqslant {}^\bullet t_k \Rightarrow v'_k \leqslant \beta(t_k) \end{cases}$$
  - The discrete transition is defined $\forall t_i \in T$ by:
    $$(M, v) \xrightarrow{t_i} (M', v') \text{ iff } \begin{cases} M \geqslant {}^\bullet t_i \\ M' = M - {}^\bullet t_i + t_i^\bullet \\ \alpha(t_i) \leqslant v_i \leqslant \beta(t_i) \\ \forall k \in [1, n]\ v'_k = \begin{cases} 0 \text{ if } t_k \in \uparrow enabled(M, t_i) \\ v_k \text{ otherwise} \end{cases} \end{cases}$$

https://doi.org/10.1017/S147106840600264X Published online by Cambridge University Press

2.2 The state class method

The main method for computing the state space of a Time Petri Net is the State Class Method introduced by Berthomieu and Diaz (1991).

Definition 3 (*State Class*)
A State Class C of a T-TPN is a pair (M, D) where M is a marking and D a set of inequalities called the firing domain. The variable x_i of the firing domain represents the firing time of the enabled transition t_i relatively to the time when the class C was entered in.

The State Class Graph is computed iteratively as follows:

Definition 4
Given a class $C = (M, D)$ and a firable transition t_j, the successor class $C' = (M', D')$ by the firing of t_j is obtained by:

1. Computing the new marking $M' = M - {}^\bullet t_j + t_j^\bullet$.
2. Making variable substitution in the domain: $\forall i \neq j,\ x_i \leftarrow x'_i + x_j$.
3. Eliminating x_j from the domain using for instance the Fourier-Motzkin method.
4. Computing a canonical form of D' using for instance the Floyd-Warshall algorithm.

In the state class method, the domain associated with a class is relative to the time when the class was entered in and as the transformation (time origin switching) is irreversible, absolute values of clocks cannot be obtained easily. The produced graph is an abstraction of the state space for which temporal information has been lost and generally, the graph has more states than the number of markings of the T-TPN. Transitions between classes are no longer labeled with a firing constraint but only with the name of the fired transition: the graph is a representation of the untimed language of the T-TPN.

2.3 Limitations of the state class method

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

To get rid of these limitations, several works construct a different State Class Graph by modifying the equivalence relation between classes. To our knowledge, proposed methods (Berthomieu and Vernadat 2003) depend on the property to check. Checking LTL or CTL properties will lead to construct different State Class Graphs.

Another limitation of methods and proposed tools to check properties is the need to compute the whole state space while only the reachability of a given marking is needed (e.g. for safety properties). The graph is then analyzed by a model-checker.

https://doi.org/10.1017/S147106840600264X Published online by Cambridge University Press

The use of T-TPN observers is even more costly: actually, for each property to be checked, a new State Class Graph has to be built and the observer can dramatically increase the size of the state space.

In the next section we will present another method to compute the state space of a bounded T-TPN. It will be used in a later section to propose a Timed Automaton that is timed bisimilar to the original T-TPN. As the graph has exactly as many nodes as the number of reachable markings of the T-TPN , we obtain a compact representation of the state space which may be efficiently model-checked using TA tools.

3 A forward algorithm to compute the state space of a bounded T-TPN

The method we propose in this paper is an adaptation, proved to be exact, of the region based method for Timed Automaton (Alur and Dill 1994; Rokicki 1993). This algorithm starts from the initial state and explores all possible evolutions of the T-TPN by firing transitions or by elapsing a certain amount of time.

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

3.1 Our algorithm: one iteration

Given the initial marking and initial values of clocks (null vector), timing successors are iteratively computed by letting time pass or by firing transitions.

Let M_0 be a marking and Z_0 a zone. The computation of the reachable markings from M_0 according to the zone Z_0 is done as follows:

- Compute the possible evolution of time (future): $\overrightarrow{Z_0}$. This is obtained by setting all upper bounds of clocks to infinity.
- Select only the possible valuations of clocks for which M_0 could exist, *i.e.* valuations of clocks must not be greater than the latest firing time of enabled transitions:

$$Z_0' = \overrightarrow{Z_0} \cap \{\bigwedge_i \{x_i \leqslant \beta_i \mid t_i \in enabled\,(M_0)\}\}$$

  So, Z_0' is the maximal zone starting from Z_0 for which the marking M_0 is legal according to the T-TPN semantics.
- Determine the firable transitions: t_i is firable if $Z_0' \cap \{x_i \geqslant \alpha_i\}$ is a non empty zone.
- For each firable transition t_i leading to a marking M_{0i}, compute the zone entering the new marking:

$$Z_i = \left(Z_0' \cap \{x_i \geqslant \alpha_i\}\right)[X_e := 0],$$

  where X_e s the set of clocks of newly enabled transitions.
  This means that each transition which is newly enabled has its clock reset. Then, Z_i is a zone for which the new marking M_{0i} is reachable.

https://doi.org/10.1017/S147106840600264X Published online by Cambridge University Press

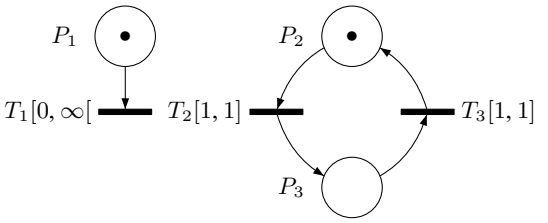


Fig. 1. Time Petri Net with an unbounded number of zones.

3.2 Convergence criterion

To ensure termination, a list of zones is associated with each reachable marking. It will keep track of zones for which the marking was already analyzed or will be analyzed. At each step, we compare the zone currently being analyzed to the ones previously computed. If the zone is included in one of the list there is no need to go further because it has already been analyzed or it will lead to compute a subgraph.

3.3 Unboundedness in T-TPN

An algorithm to enumerate reachable markings for a bounded T-TPN could be based on the described algorithm but, generally, it will lead to a non-terminating computation. Though the number of reachable markings is finite for a bounded T-TPN , the number of zones in which a marking is reachable is not necessarily finite (see Figure 1).

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

Actually, the number of zones is not bounded because infinity is used as latest firing time (T_1). If infinity is not used as latest firing time, all clocks are bounded and so, the number of different zones is bounded (Alur and Dill 1994). The "naive" algorithm is then exact and can be used to compute the state space of a bounded T-TPN.

Consequence 1

For a bounded T-TPN without infinity as latest firing time, this forward analysis algorithm using zones computes the exact state space of the T-TPN.

https://doi.org/10.1017/S147106840600264X Published online by Cambridge University Press

In the next section, we propose a more general algorithm which computes the state space of a T-TPN as defined in section 2, *i.e.* with infinity as latest firing time allowed.

3.4 General algorithm

A common operator on zones is the *k-approx* operator. For a given k value, the use of this operator allows to create a finite set of distinct zones. The algorithm proposed is an extension of the one presented in the previous section. It consists in applying the *k-approx* operator on the zone resulting from the last step:

$$Z_i = k - approx\left(\left(Z_0' \cap \{x_i \geqslant \alpha_i\}\right)[X_e := 0]\right)$$

This approximation is based on the fact that once the clock associated with an unbounded transition ($[\alpha, \infty[$) has reached the value α, its precise value does not matter anymore.

Unfortunately, recent works on Timed Automaton (Bouyer 2002; Bouyer 2003) have proved that this operator generally leads to an overapproximation of the reachable localities of TA. However, for a given class of TA (diagonal-free), there is no overapproximation of the reachable localities.

Results of Bouyer are directly extensible for T-TPN. As computation on zones only involved diagonal-free constraints, the following theorem holds:

Theorem 1
A forward analysis algorithm using *k-approx* on zones is exact with respect to T-TPN marking reachability for bounded T-TPN.

A detailed proof is available in Gardey *et al.* (2003).

3.5 Example

Let us consider the T-TPN of Figure 1. We associate the clock x_i with the transition T_i of the T-TPN and recall that clocks associated with each transition count the time since the transition has been newly enabled.

The algorithm starts from the initial state: $l_0 = (M_0, Z_0)$, with $M_0 = (1 \quad 1 \quad 0)$ and $Z_0 = \{x_1 = x_2 = 0\}$. At marking M_0, transitions T_1 and T_2 are enabled.

The first step is to compute the possible future, *i.e.* the maximal amount of time for which the marking M_0 may exist:

$$\begin{aligned} \overrightarrow{Z_0} \cap Inv(M_0) &= \{x_1 = x_2 \in [0, \infty[\} \cap \{x_1 \leqslant \infty \wedge x_2 \leqslant 1\} \\ &= \{x_1 = x_2 \in [0, 1]\} \end{aligned}$$

From this zone, two transitions are firable: T_1 and T_2.

Firing of T_1

- the new marking is $M_1 = (0 \quad 1 \quad 0)$
- the new zone is obtained by intersecting the previous zone ($\overrightarrow{Z_0} \cap Inv(M_0)$) with the guard $x_1 \geqslant 0$, deleting clocks of transitions that are no longer enabled in

https://doi.org/10.1017/S147106840600264X Published online by Cambridge University Press

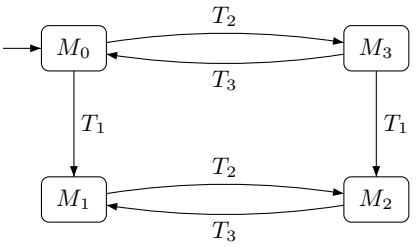


Fig. 2. Graph of reachable markings.

M_1 (x_1) and reseting clocks of newly enabled transitions (none).

$$\begin{aligned} Z_1 = &\ \{x_1 = x_2 \in [0,1]\} \cap \{x_1 \geqslant 0\} && \text{(intersect with guard)} \\ = &\ \{x_1 = x_2 \in [0,1]\} \\ = &\ \{x_2 \in [0,1]\} && \text{(delete } x_1\text{)} \end{aligned}$$

Firing of T_2

- the new marking is $M_3 = (1 \quad 0 \quad 1)$
- the new zone is obtained by intersecting the previous zone ($\overrightarrow{Z_0} \cap Inv(M_0)$) with the guard $x_2 \geqslant 1$, deleting clocks of transitions that are no longer enabled in M_3 (x_2) and reseting clocks of newly enabled transitions (x_3).

$$\begin{aligned} Z_3 = &\ \{x_1 = x_2 \in [0,1]\} \cap \{x_2 \geqslant 1\} && \text{(intersect with guard)} \\ = &\ \{x_1 = x_2 = 1\} \\ = &\ \{x_1 = 1\} && \text{(delete } x_2\text{)} \\ = &\ \{x_1 = 1 \wedge x_3 = 0\} && \text{(reset } x_3\text{)} \end{aligned}$$

We got two new states to analyze: (M_1, Z_1) and (M_3, Z_3). We apply the same algorithm to these two states.

Considering (M_1, Z_1):

$$\begin{aligned} Z_1' = \overrightarrow{Z_1} \cap Inv(M_1) &= \{x_2 \in [0,1]\} \cap \{x_2 \leqslant 1\} \\ &= \{x_2 \in [0,1]\} \end{aligned}$$

T_2 is firable and leads to the new state: (M_2, Z_2) with $M_2 = \begin{pmatrix} 0 & 0 & 1 \end{pmatrix}$ and $Z_2 = \{x_3 = 0\}$. Analyzing (M_2, Z_2) leads to the new state $(M_1, \{x_2 = 0\})$. As $\{x_2 = 0\} \subset Z_1$, the algorithm stops and get a new state to analyze: (M_3, Z_3).

Considering (M_3, Z_3):

$$\begin{aligned} Z_3' = \overrightarrow{Z_3} \cap Inv(M_3) &= \{x_1 - x_3 = 1,\ x_1 \in [0, \infty[\} \cap \{x_1 \leqslant \infty \wedge x_3 \leqslant 1\} \\ &= \{x_1 - x_3 = 1 \wedge x_3 \leqslant 1\} \end{aligned}$$

T_3 and T_1 are firable...

The analysis is performed until no new states are created. We then build the graph of reachable markings shown in Figure 2.

In this section we have presented an algorithm that exactly computes the reachable markings of a bounded T-TPN with ∞ as latest firing time. The graph computed is not suitable to verify time logic properties. So, in the next section we present

https://doi.org/10.1017/S147106840600264X Published online by Cambridge University Press

a transformation of the graph into a Timed Automaton we proved to be timed bisimilar to the original T-TPN. Consequently, model-checking methods on TA become available for the model-checking of T-TPN.

4 Marking timed automaton of Time Petri Net

We first recall the definition of Timed Automata, introduced by Alur and Dill (1994) and their semantics.

4.1 Timed automaton: definitions

Timed Automata are an extension of classical automata providing timing constraints. A transition can occur if clocks valuations satisfy constraints called "guard". Actions on clocks (reset for instance) are associated with transition. The system can idle in a locality if valuations of clocks satisfy some constraints called "invariant".

Definition 5 (*Constraints*)
Let V be a set of clocks, $\mathscr{C}(V)$ is the set of timing constraints upon V i.e. the set of expressions δ defined by:

$$\delta := v \sim c \mid v - v' \sim c \mid \neg\, \delta_1 \mid \delta_1 \wedge \delta_2$$

with $v, v' \in V$, $\sim \in \{<, \leqslant, =, \geqslant, >\}$ and $c \in \mathbb{N}$.

Definition 6 (*TA*)
A Timed Automaton is a tuple (L, l_0, C, A, E, Inv) defined by:

- L a finite set of locations,
- $l_0 \in L$ the initial location,
- C a finite set of positive real-valued clocks,
- A a finite set of actions,
- $E \subset L \times \mathscr{C}(C) \times A \times 2^C \times L$ a finite set of transitions. $e = (l, \gamma, a, R, l')$ is the transition from location l to location l' with the guard γ, the label a and the set of clocks to reset R,
- $Inv : L \times \mathscr{C}(C) \rightarrow \{true, false\}$, a function assigning to each location an invariant.

The semantics of a Timed Automaton is given by a Timed Transition System (TTS).

Definition 7 (*Semantics of a TA*)
The semantics of a Timed Automaton is the Timed Transition System $\mathscr{S} = (Q, Q_0, \rightarrow)$ where:

- $Q = L \times (\mathbb{R}_{\geqslant 0})^C$,
- $Q_0 = (l_0, \bar{0})$,
- $\rightarrow$ is the transition relation including a discrete transition and a continuous transition.

https://doi.org/10.1017/S147106840600264X Published online by Cambridge University Press

- The discrete transition is defined $\forall a \in A$ by:

$$(l,v) \xrightarrow{a} (l',v') \text{ iff } \exists (l,\gamma,a,R,l') \in E \text{ such as } : \begin{cases} \gamma(v) = true \\ v' = v[R \leftarrow 0] \\ Inv(l')(v') = true \end{cases}$$

- The continuous transition is defined $\forall d \in \mathbb{R}_{\geqslant 0}$ by:

$$(l,v) \xrightarrow{\epsilon(d)} (l,v') \text{ iff } \begin{cases} v' = v + d \\ \forall t' \in [0,d],\ Inv(l)(v+t') = true \end{cases}$$

4.2 Labeling algorithm

The algorithm given in section 3 represents the marking graph of the T-TPN. We show here that it can easily be labeled to generate a Timed Automaton timed bisimilar to the T-TPN.

Let $\mathcal{G} = (M, T)$ be the graph produced by the algorithm where:

- M is the set of reachable markings of the T-TPN : $M_0, \ldots, M_p$
- T is the set of transitions: $T_0, \ldots, T_q$.

The Timed Automaton will be obtained by associating to each marking an invariant and to each transition a guard and some clocks assignments.

4.2.1 Invariant

First, an invariant is associated with each marking M_k. By construction, in each marking, only the possible evolution of time is computed: the entering zone is intersected with the set of constraints $\{x_i \leqslant \beta_i\}$, where x_i are clocks of transitions enabled by the marking M_k. Then, the invariant associated with each marking M_k is defined by:

$$I(M_k) = \{x_i \leqslant \beta_i \mid t_i \in enabled(M_k)\}$$

4.2.2 Guard

Each transition T_k of the graph $\mathcal{G}$ corresponds to the firing of a transition t_i. Then we label T_k by:

- the action name t_i,
- the guard: $x_i \geqslant \alpha_i$,
- the clocks assignments: $x_k \leftarrow 0$ for all clocks x_k associated with a newly enabled transition t_k

4.3 Marking timed automaton

The Timed Automaton we obtain is then defined as follows:

Definition 8 (*Marking Timed Automaton*)

- $L = \{M_0, \ldots, M_p\}$ is the set of localities *i.e.* the set of reachable markings of the T-TPN.

https://doi.org/10.1017/S147106840600264X Published online by Cambridge University Press

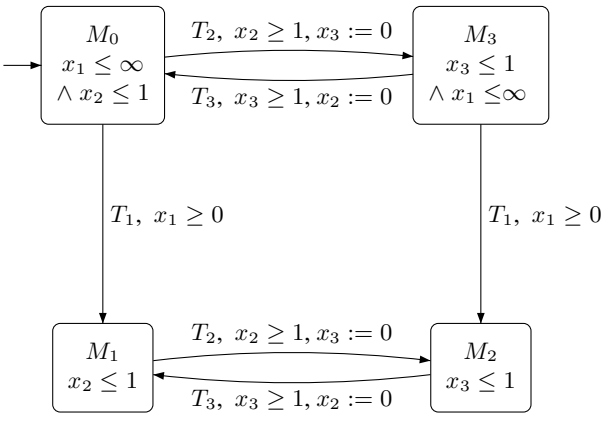


Fig. 3. Time marking automaton.

- $l_0 = M_0$ is the initial locality.
- $C = \{x_1, \ldots, x_q\}$ is the set of clocks *i.e.* the set of all clocks associated with a transition.
- $A = \{t_1, \ldots, t_q\}$ is the set of actions *i.e.* the transitions of the T-TPN.
- $E \subset L \times \mathscr{C}(C) \times A \times 2^C \times L$ is the finite set of transitions. Let $e = \left(M_i, \gamma, a, R, M_j\right)$ a transition, e is defined as follows:
  - $a = t_k$
  - $\gamma = x_k \geqslant \alpha_k$
  - $R = \{x_i \mid t_i \in \uparrow enabled\,(M_i, t_k)\}$
- $Inv : L \times \mathscr{C}(C) \rightarrow \{true, false\}$, with:

$$Inv(M_i) = \{x_i \leqslant \beta_i \mid t_i \in enabled\,(M_i)\}$$

Example Considering the T-TPN of Figure 1, the resulting Timed Automaton is as shown in Figure 3.

4.4 Bisimulation

Definition 9
As defined in the time transition system for a T-TPN $\mathscr{T}$, we note $\mathscr{Q}_{\mathscr{T}}$ the set of states of $\mathscr{T}$. $\mathscr{Q}_{\mathscr{A}}$ is the set of states of a TA $\mathscr{A}$.

Definition 10
Let $\mathscr{R} \subset \mathscr{Q}_{\mathscr{T}} \times \mathscr{Q}_{\mathscr{A}}$ be the relation between a state of the Timed Automaton and a state of the Time Petri Net defined by:

$$\begin{cases} \forall (M, v) \in \mathscr{Q}_{\mathscr{T}} \\ \forall (l, \bar{v}) \in \mathscr{Q}_{\mathscr{A}} \end{cases}, (M, v)\mathscr{R}(l, \bar{v}) \Leftrightarrow \begin{cases} M = \mathbf{M}(l) \\ v = \bar{v} \end{cases}$$

where $\mathbf{M}$ is the function giving the associated marking of a TA state l.

Two states are in relation if their "markings" and their clocks valuations are equals.

https://doi.org/10.1017/S147106840600264X Published online by Cambridge University Press

Theorem 2

$\mathscr{R}$ is a bisimulation:

For all (M,v), $(l,\bar{v})$ such that $(M,v)\,\mathscr{R}\,(l,\bar{v})$:

- $(M,v) \xrightarrow{t_i} (M',v') \Leftrightarrow \begin{cases} (l,\bar{v}) \xrightarrow{t_i} (l',\bar{v}') \\ (M',v')\mathscr{R}(l',\bar{v}') \end{cases}$
- $(M,v) \xrightarrow{\delta} (M,v') \Leftrightarrow \begin{cases} (l,\bar{v}) \xrightarrow{\delta} (l,\bar{v}') \\ (M,v')\mathscr{R}(l,\bar{v}') \end{cases}$

Proof

Continuous transition – time elapsing

Let $(M,v_{\mathscr{T}}) \in \mathscr{Q}_{\mathscr{T}}$, $(l,v_{\mathscr{A}}) \in \mathscr{Q}_{\mathscr{A}}$, and $\delta \in \mathbb{R}^{\geqslant 0}$.

We prove that if the T-TPN can idle in a state, this is allowed on the constructed TA *i.e.* if the system can idle for any δ such that $\forall k \in [1,n]\ M \geqslant {}^\bullet t_k \Rightarrow v_{\mathscr{T}}(t_k) + \delta \leqslant \beta(t_k)$ then the automaton verifies: $\forall t \in [0,\delta]\ Inv(l)(v_{\mathscr{A}} + t) = true$.

By construction, the invariant of the location l is obtained by the conjunction of the latest firing times of enabled transitions. So $Inv(l) = \bigwedge \{x_i \leqslant \beta(t_i)\}$ where $t_i \in enabled(\mathbf{M}(l))$. $(M,v_{\mathscr{T}})$ and $(l,v_{\mathscr{A}})$ are in relation so $v_{\mathscr{T}} = v_{\mathscr{A}}$. As $v_{\mathscr{T}}(t_i)+\delta \leqslant \beta(t_i)$ then for all $t \in [0,\delta]$ $v_{\mathscr{A}}(t_i)+t \leqslant \beta(t_i)$. This means that $\forall t \in [0,\delta]\ Inv(l)(v_{\mathscr{A}}+t) = true$.

To conclude, the automaton can idle in the state and $(M,v_{\mathscr{T}} + \delta)\mathscr{R}(l,v_{\mathscr{A}} + \delta)$.

Symmetrically, we prove that if the TA can idle for a time δ, the T-TPN can idle for the same time δ.

According to the semantics of T-TPN , a continuous transition can occur if and only if $\forall t_k \in enabled(M)$, $v_{\mathscr{T}}(t_k) + \delta \leqslant \beta(t_k)$. As $(M,v_{\mathscr{T}})$ and $(l,v_{\mathscr{A}})$ are in relation, $v_{\mathscr{T}} = v_{\mathscr{A}}$. The TA can idle in the state for all $t \in [0,\delta]$ $v_{\mathscr{A}}(t_i) + t \leqslant \beta(t_i)$ by construction of the invariant. Then, $t = \delta$ prove the result.

The T-TPN can idle in the marking and $(M,v_{\mathscr{T}} + \delta)\mathscr{R}(l,v_{\mathscr{A}} + \delta)$.

Concerning continuous transitions, $\mathscr{R}$ is a bisimulation.

Discrete transition – firing a transition t_i

Let $(M,v_{\mathscr{T}}) \in \mathscr{Q}_{\mathscr{T}}$ and $(l,v_{\mathscr{A}}) \in \mathscr{Q}_{\mathscr{A}}$ be two states in relation.

We prove that if a transition is firable for the T-TPN , it is firable for the TA and the two resulting states are in relation.

A transition t_i of the T-TPN can be fired if: $M \geqslant {}^\bullet t_i$ and $\alpha(t_i) \leqslant v_{\mathscr{T}}(t_i) \leqslant \beta(t_i)$. The resulting marking is $M' = M - {}^\bullet t_i + t_i^\bullet$ and the resulting valuation is $v'_{\mathscr{T}}(t_k) = 0$ for all newly enabled transition t_k, all others valuations remain unchanged.

The corresponding action is allowed on the constructed TA if and only if

$$\exists (l,\gamma,a,R,l') \in E \text{ such as } : \begin{cases} \gamma(v) = true \\ v_{\mathscr{A}} = v_{\mathscr{A}}[R \leftarrow 0] \\ Inv(l')(v'_{\mathscr{A}}) = true \end{cases}$$

As t_i is firable, it exits by construction a transition of the TA from l, such that $\mathbf{M}(l) = M$, to a location l' such that $\mathbf{M}(l') = M'$. The guard is by construction, $\gamma = x_i \geqslant \alpha(t_i)$. Thus, as t_i is firable $\gamma(v_{\mathscr{A}}) = true$.

https://doi.org/10.1017/S147106840600264X Published online by Cambridge University Press

Also by construction, the clocks to be reset for the TA are the same clocks to be reset for the T-TPN. Thus, $v'_{\mathscr{A}} = v'_{\mathscr{T}}$.

As clocks newly enabled are set to 0, they verifies the inequalities $x_j \leqslant \beta(t_j)$ in the invariant of l'. All other clocks stay unchanged: $v'_{\mathscr{A}}(t_j) \leqslant \beta(t_j)$ for all other enabled clocks. Thus, $Inv(l')(v'_{\mathscr{A}}) = true$.

So the transition on TA is allowed and $(M', v'_{\mathscr{T}})\mathscr{R}(l', v'_{\mathscr{A}})$.

Symmetrically, we prove that if t_i is firable for the TA, it is firable for the T-TPN. The two resulting states are in relation.

A transition $e = (l, t_i, \gamma, R, l')$ of the TA can occur and leads to a new state $(l', v'_{\mathscr{A}})$ if and only if $\gamma(v_{\mathscr{A}}) = true$ and $Inv(l')(v'_{\mathscr{A}}) = true$. Then $v'_{\mathscr{A}} = v_{\mathscr{A}}[R \leftarrow 0]$.

The corresponding action is allowed on the T-TPN and leads to a new state $(M', v'_{\mathscr{T}})$ if and only if:

$$\begin{cases} M \geqslant^{\bullet} t_i \\ M' = M -^{\bullet} t_i + t_i^{\bullet} \\ \alpha(t_i) \leqslant v_i \leqslant \beta(t_i) \\ \forall \text{ transitions } t_k \; v'_{\mathscr{T}}(t_k) = \begin{cases} 0 \text{ if } t_k \in \uparrow enabled(M, t_i) \\ v_{\mathscr{T}}(t_k) \text{ otherwise} \end{cases} \end{cases}$$

By definition of the Marking Timed Automaton, if t_i is firable for the TA, it is for the T-TPN. So $M \geqslant^{\bullet} t_i$ and the resulting marking is by definition $M' = M -^{\bullet} t_i + t_i^{\bullet}$.

$(l, v_{\mathscr{A}})$ and $(M, v_{\mathscr{T}})$ are in relation so $v_{\mathscr{T}} = v_{\mathscr{A}}$.

As, $\gamma(v_{\mathscr{A}}) = true$ and $Inv(l)(v_{\mathscr{A}}) = true$ so, $\alpha(t_i) \leqslant v_{\mathscr{T}}(t_i) \leqslant \beta(t_i)$.

By construction, the clocks to be reset are the clocks of newly enabled transitions *i.e.* the clocks of R. So $v'_{\mathscr{A}} = v'_{\mathscr{T}}$.

To conclude, t_i is firable for the T-TPN and $(M', v'_{\mathscr{T}})$ and $(l', v'_{\mathscr{A}})$ are in relation.

$\mathscr{R}$ is a bisimulation for discrete transitions. □

5 Performances

We have implemented the algorithm to compute all the reachable markings of a bounded T-TPN using DBM (Difference Bounded Matrices) to encode zones. The tool implemented (Mercutio) is integrated into Romeo (Romeo 2003), a software for T-TPN edition and analysis.

As boundedness of T-TPN is undecidable, Mercutio offers stopping criteria: number of reached markings, computation time, bound on the number of tokens in a place. It also provides an on-the-fly reachability test of markings and export the automaton in Kronos or Uppaal syntax. Concerning the on-the-fly reachability test, Mercutio also provides a trace (sequence of transitions and interval in which they are fired) leading to the marking.

5.1 Comparison with other methods

We present here a comparison (Table 1) of three methods to compute the state space of a T-TPN:

https://doi.org/10.1017/S147106840600264X Published online by Cambridge University Press

Table 1. *Time to compute the state space of a T-TPN*

Time Petri Net	T-TPN (p./t.)	TINA	GPN	MERCUTIO
Example 1 (oex15)	16/16	10.5 s	12.9 s	2 s
Example 2 (oex7)	22/20	30.5 s	9.8 s	1.3 s
Example 3 (oex8)	31/21	29 s	12.2 s	1.4 s
Example 4 (P6C7)	21/20	31.6 s	1 min 17 s	7.9 s
Example 5 (P10C10)	32/31	4.2 s	6.8 s	1 s
Example 6 (GC - 3)	20/23	2 s	1.2 s	0.1 s
Example 7 (GC - 4)	24/29	3 min 8 s	1 min 3 s	10.8 s
Example 8 (P6C9)	25/24	2 min 49 s	6 min 2 s	22.9 s
Example 9 (P6C10)	27/26	8 min 53 s	36 min	1 min
Example 10 (P6C11)	29/28	14 min 36 s	1 h 1 min	2 min 20s
Example 11 (P6C12)	31/30	23 min 34 s	2 h 7 min	3 min 59s
Example 12 (P6C13)	33/32	36 min 25 s	×	6 min 3s

- the method proposed in this paper with our tool MERCUTIO.
- the State Class Graph computation (BERTHOMIEU) with the tool TINA.
- the State Class Timed Automaton (LIME and ROUX) with the tool GPN.

Computations were performed on a Pentium 2 (400MHz) with 320MB of RAM.

Examples 1 to 5 come from real-time systems (parallel tasks [1], periodic tasks[2–3], producer-consumer [4–5,8–12]). Examples 7 and 8 are the classical level crossing example (3 and 4 trains).

For this set of examples and for all nets we have tested, our tool performs better than TINA and than GPN. For Example 12, GPN ran out of memory.

5.2 Reducing the number of clocks

A major issue in model checking TA is the number of clocks in the automaton. Time computation is exponential in the number of clocks. Consequently, obtaining an automaton with a reduced number of clocks is of importance.

The algorithm we propose assigns a clock to each transition. Thus, the resulting automaton has as many clocks as transitions of the T-TPN. However we have underlined that for each location, only a reduced number of clocks (active clocks) really matter for the timing evolution of the T-TPN.

Daws and Yovine (1996) proposed a syntactical method to reduce the number of clocks of a TA. As a single Timed Automaton is build with our method (no need to compute parallel composition) we applied this reduction. The table 2 presents the comparison between the clocks of (1) the Timed Automaton obtained, (2) the Timed Automaton obtained after syntactical clocks reduction (we used OPTIKRON from KRONOS (Yovine 1997)), (3) the State Class Timed Automaton using GPN that ensures a minimal number of clocks using classes.

These results are all the more encouraging that, reducing the number of clocks is made syntactically and is made at no cost comparatively to the state space

https://doi.org/10.1017/S147106840600264X Published online by Cambridge University Press

Table 2. *Structure of resulting timed automata*

Time Petri Net	Clocks(1)[1]	Marking(2) TA			State Class TA(3)		
		Cl. [2]	N.[3]	T.[4]	Cl.	N.	T.
Example 1 (oex15)	16	4	361	1095	4	998	3086
Example 2 (oex7)	20	11	637	2284	7	1140	3990
Example 3 (oex8)	21	11	695	2444	7	1277	4344
Example 4 (P6C7)	20	13	449	4175	3	11490	50268
Example 5 (P10C10)	31	4	1088	5245	2	1088	5245
Example 6 (GC - 3)	23	5	94	271	3	286	763
Example 7 (GC - 4)	29	6	318	1221	4	2994	11806
Example 8 (P6C9)	24	15	1299	12674	3	24483	117918
Example 9 (P6C10)	26	16	2596	27336	3	59756	313729
Example 10 (P6C11)	28	17	4268	44620	3	82583	440540
Example 11 (P6C12)	30	18	6846	70856	3	112023	606771
Example 12 (P6C13)	32	19	10646	108842	×	×	×

Number of: [1]clocks of the original T-TPN , [2]clocks of the TA, [3]nodes of the TA, [4]transitions of the TA.

computation. The State Class Timed Automaton always as a lower number of clocks but its construction is not as fast as our method: the Timed Automaton has lower clocks at the price of a greater size. For Example 12, we have not succeeded in computing the State Class Timed Automaton (out of memory).

6 Applications

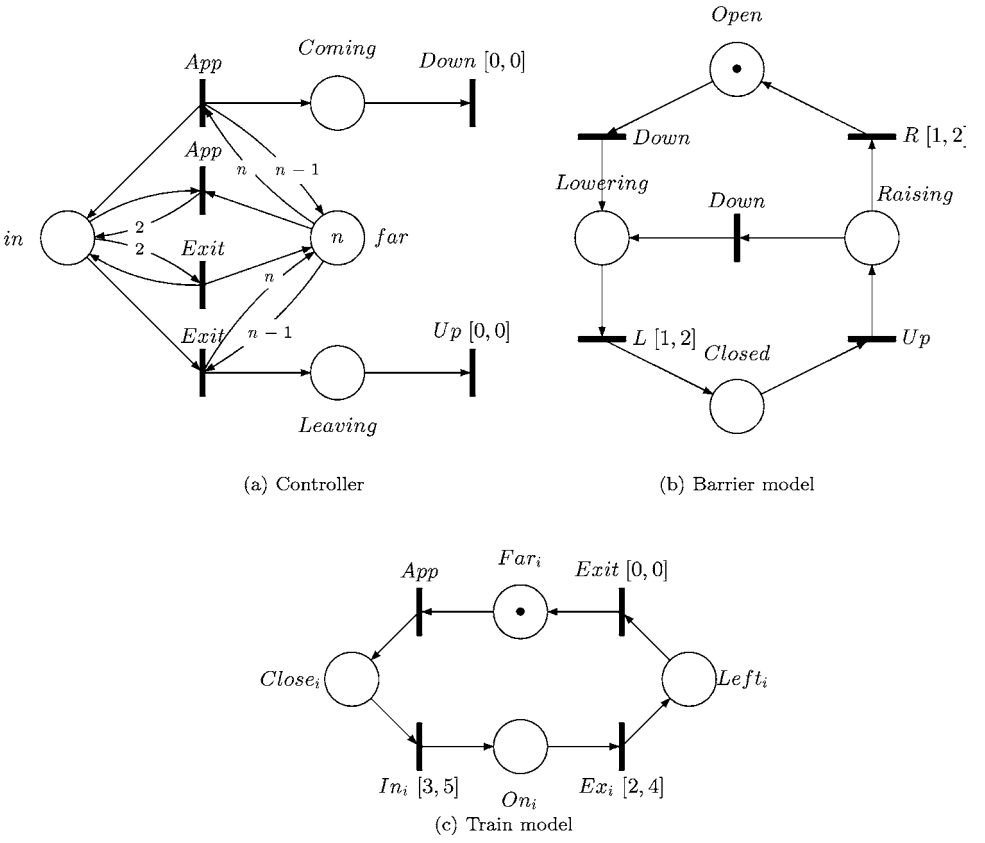
We propose in this section some applications of our method to model-check T-TPN.

6.1 Model checking of quantitative properties

Since they were introduced, Timed Automata are an active research area and several methods and tools have been developed to analyze them. Tools like Uppaal (Larsen *et al.* 1997) or Kronos (Yovine 1997) successfully implement efficient algorithms and data structures to provide model-checking on TA (TCTL model-checking for instance): numerous case studies have been performed with real reactive systems.

Concerning T-TPN , few studies were realized and properties that can be checked are mainly safety untimed properties (reachability). Time or untime properties are mainly verified over T-TPN using "observers". Basically, properties are transformed in an additional T-TPN motif called "observer", and then, the problem is transformed into a reachability test. Such methods are not easy to use: (1) modeling the property with an observer is not easy (it exists some generic observers (Toussaint *et al.* 1997), but for few properties), (2) the observer's size may be as large as the initial T-TPN , (3) due to the increase of the T-TPN 's size, computing the state space will be more time expensive.

https://doi.org/10.1017/S147106840600264X Published online by Cambridge University Press



(a) Controller (b) Barrier model

(c) Train model

Fig. 4. Gate controller.

The method we propose here, is to use existent TA tools to perform model-checking of T-TPN. As a Timed Automaton is produced, model-check a T-TPN (LTL,CTL) becomes possible and verifying quantitative time property (TCTL) is possible. Moreover, as the automaton constructed is a Timed Automaton with diagonal free constraints, model checking could be done using on-the-fly algorithms on TA (UPPAAL (Larsen *et al.* 1997), KRONOS (Yovine 1997)).

Example Let us consider the classical level crossing example. The system is modeled using the three patterns of the Figure 4. This model is made of a controller (4(a)), a barrier model (4(b)) and four identical trains (4(c)). The resulting Petri Net is obtained by the parallel composition of these T-TPN.

The property "the barrier is closed when a train crosses the road" is a safety property and is interpreted as a reachability test: we want to check if there exists a state such that for any train i: $M(On_i) = 1$ and $M(Closed) = 0$. This could be checked directly on the computed graph using MERCUTIO or using UPPAAL to test the property. In UPPAAL, the property is expressed as: `E<>((M[`On_1`]==1 or M[`On_2`]==1 or M[`On_3`]==1 or M[`On_4`]==1) and M[Closed]==0)`. In both cases, the result is **False**, proving that no train may cross the road while the barrier is not closed.

https://doi.org/10.1017/S147106840600264X Published online by Cambridge University Press

Using the automaton, it is possible to model time properties. For instance, "when the train i approaches, the barrier closes within delay δ" may be checked. In TCTL this property is expressed by: $M(close_i) = \uparrow 1 \Longrightarrow \forall\Diamond_{\leqslant\delta} M(closed) = 1$. $M(close_i) = \uparrow 1$ means that only states for which $M(close_i) = 1$ in the state and $M(close_i) = 0$ for all the preceding states. To check this property on the TA using UPPAAL or on the T-TPN using reachability analysis leads to create an observer or modify the model. For instance, to use UPPAAL we have to add an additional clock that starts when a train change its state to $close_i$. By using KRONOS, there is no need to modify or create an observer. Given the TA and a TCTL formula, KRONOS can perform model-checking using classical TCTL forward or backward algorithms.

6.2 Mixing timed automata and Time Petri Nets

The method proposed in this paper provides a common framework for using and analyzing reactive systems modeled with Timed Automata or Time Petri Nets.

Many systems are modeled using T-TPN (FIP, CAN), nevertheless some problems (time controller synthesis for instance) benefit of larger studies and efficient tools. Then, it may be necessary to have a mixed representation of the system.

We give here some examples of mixing Timed Automata and Time Petri Nets:

Test Case Given a reactive system expressed with a T-TPN , different scenarios may be studied by synchronizing it with a Test Automaton. This Test Automaton represents the sequence of transitions to be fired and the synchronization is made over the firing of transitions.

Controller Given a reactive system expressed with a T-TPN , a controller may be modeled using TA to constraint the execution of the system.

7 Conclusions

In this paper, we proposed an efficient method to compute the state space of a bounded T-TPN. The proposed algorithm performs a forward computation of the state space and we proved it is exact with respect to reachability even for bounded T-TPN with ∞ as latest firing time. We proposed a labeling algorithm of the produced graph to build a Timed Automaton that we proved to be timed bisimilar to the original T-TPN. Some examples were given to show that our tool performs better than two other methods used to compute the state space of a T-TPN : the State Class Timed Automaton (GPN) and the State Class Graph (TINA). Though the number of clocks of our TA is greater than the one of the State Class Timed Automaton, our construction is faster and syntactical clocks reduction algorithms may be successfully applied to reduce it.

Consequently, our method allows the use of Timed Automaton tools to model-check T-TPN. In particular, the Timed Marking Automaton makes TCTL model-checking feasible for bounded T-TPN , which, to our knowledge has not been done before.

We are currently involved in two different research area. First, we think possible to use efficient data structures (BDD-like structure) to improve our implementation

https://doi.org/10.1017/S147106840600264X Published online by Cambridge University Press

and we are studying Partial Order methods to reduce time and space requirements. Finally, it would be useful to develop a full model-checker for T-TPN without having to build the Timed Automaton. Then, a further step in the analysis of real-time reactive systems will be to provide methods for the time controller synthesis problem for T-TPN.

References

Abdulla, P. A. and Jonsson, B. 1998. Ensuring completeness of symbolic verification methods for infinite-state systems. *Theoretical Computer Science 256*, 145–167.

Abdulla, P. A. and Nylén, A. 2001. Timed petri nets and bqos. *22nd International Conference on Application and Theory of Petri Nets (ICATPN'01)*. Lecture Notes in Computer Science, vol. 2075. Springer-Verlag, pp. 53–70.

Alur, R. and Dill, D. L. 1994. A theory of timed automata. *Theoretical Computer Science 126,* 2, 183–235.

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

Berthomieu, B. and Vernadat, F. 2003. State class constructions for branching analysis of time petri nets. *9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2003)*. Lecture Notes in Computer Science, vol. 2619. Springer-Verlag, pp. 442–457.

Bouyer, P. 2002. Timed automata may cause some troubles. Technical report, LSV.

Bouyer, P. 2003. Unteamable timed automata! *Proc. 20th Annual Symposium on Theoretical Aspects of Computer Science (STACS'2003)*. LNCS, vol. 2607. Springer-Verlag, pp. 620–631.

Cassez, F. and Roux, O. H. 2004. Structural translation from time Petri nets to timed automata. *Fourth International Workshop on Automated Verification of Critical Systems (AVoCS'04)*. London (UK).

Corts, L. A., Eles, P. and Peng, Z. 2000. Verification of embedded systems using a petri net based representation. *13th International Symposium on System Synthesis (ISSS 2000)*. Madrid, Spain, pp. 149–155.

Daws, C. and Yovine, S. 1996. Reducing the number of clock variables of timed automata. *17th IEEE Real Time Systems Symposium, RTSS'96*. IEEE Computer Society Press.

de Frutos Escrig, D., Ruiz, V. V. and Alonso, O. M. 2000. Decidability of properties of timed-arc petri nets. *21st International Conference on Application and Theory of Petri Nets (ICATPN'00)*. Lecture Notes in Computer Science, vol. 1825. Springer-Verlag, pp. 187–206.

Diaz, M. and Senac, P. 1994. Time stream petri nets: a model for timed multimedia information. *15th International Conference on Application and Theroy of Petri Nets*. LNCS, vol. 815. Springer-Verlag, pp. 219–238.

Finkel, A. and Schnoebelen, P. 1998. Fundamental structures in well-structured infinite transitions systems. *3rd Latin American Theoretical Informatics Symposium (LATIN'98)*. Lecture Notes in Computer Science, vol. 1380. Springer-Verlag, pp. 102–118.

Gardey, G., Roux, O. H. and F.Roux, O. 2003. Using zone graph method for computing the state space of a time petri net. In *Formal Modeling and Analysis of Timed Systems (FORMATS'2003)*. LNCS. Springer–Verlag.

Khansa, W., Denat, J.-P. and Collart-Dutilleul, S. 1996. P-time petri nets for manufacturing systems. In *International Workshop on Discrete Event Systems, WODES'96*, pp. 94–102.

Larsen, K. G., Pettersson, P. and Yi, W. 1997. Uppaal in a nutshell. *International Journal on Software Tools for Technology Transfer 1,* 1–2, 134–152.

https://doi.org/10.1017/S147106840600264X Published online by Cambridge University Press

Lilius, J. 1999. Efficient state space search for time petri nets. *MFCS Workshop on Concurrency '98*. ENTCS, vol. 18. Elsevier.

Lime, D. and Roux, O. H. 2003. State class timed automaton of a time petri net. *The 10th International Workshop on Petri Nets and Performance Models, (PNPM'03)*. IEEE Computer Society.

Menasche, M. 1982. Analyse des rseaux de petri temporiss et application aux systmes distribus. PhD thesis, Universit Paul Sabatier, Toulouse, France.

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

Okawa, Y. and Yoneda, T. 1997. Symbolic ctl model checking of time petri nets. *Electronics and Communications in Japan*, S. Technica, Ed. Vol. 80. pp. 11–20.

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

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

Rokicki, T. G. 1993. Representing an modeling circuits. PhD thesis, Stanford University.

Rokicki, T. G. and Myers, C. J. 1994. Automatic verification of timed circuits. *6th International Conference on Computer-Aided Verification (CAV'94)*. LNCS, vol. 818. Springer–Verlag, pp. 468–480.

Romeo. 2003. http://www.irccyn.ec-nantes.fr/irccyn/d/fr/equipes/tempsreel/logs. *A tool for Time Petri Nets Analysis*.

Sava, A. T. and Alla, H. 2001. Commande par supervision des systmes vnements discrets temporises. *Modlisation des systmes ractifs (MSR 2001)*, 71–86.

Toussaint, J., Simonot-Lion, F. and Thomesse, J.-P. 1997. Time constraint verifications methods based time petri nets. In *6th Workshop on Future Trends in Distributed Computing Systems (FTDCS'97)*. Tunis, Tunisia, pp. 262–267.

Yoneda, T. and Ryuba, H. 1998. Ctl model checking of time petri nets using geometric regions. *IEICE Transactions on Information and Systems E99-D,* 3, 297–396.

Yovine, S. 1997. Kronos: A verification tool for real-time systems. *International Journal of Software Tools for Technology Transfer 1,* 1–2, 123–133.

https://doi.org/10.1017/S147106840600264X Published online by Cambridge University Press