TCTL model checking of Time Petri Nets

Hanifa Boucheneb, Guillaume Gardey, Olivier Henri Roux

TCTL model-checking of Time Petri Nets

Hanifa Boucheneb[1], Guillaume Gardey[2,3] and Olivier H. Roux[2]
Affiliations :
[1]: École polytechnique de Montréal, C.P. 6079, succ. Centre-ville Montréal H3C3A7 Canada
[2]: IRCCyN, Université de Nantes 1 rue de la Noë, BP 92101 44321 Nantes cedex 03 France
[3]: currently at Thales Information Systems Security : nCipher Product Line - Cambridge, UK.
hanifa.boucheneb@polymtl.ca , glinmac@gmail.com, olivier-h.roux@irccyn.ec-nantes.fr

Abstract

We consider *Time Petri Nets* (TPN) for which a firing time interval is associated with each transition. State space abstractions for TPN preserving various classes of properties (LTL, CTL, CTL*) can be computed, in terms of so called state classes. Some methods were proposed to check quantitative timed properties but are not suitable for effective verification of properties of real-life systems.

In this paper, we consider *subscript* TCTL for TPN (*TPN-TCTL*) for which temporal operators are extended with a time interval, specifying a time constraint on the firing sequences. We prove the decidability of *TPN-TCTL* on bounded TPN and give its theoretical complexity.

We propose a zone based state space abstraction that preserves marking reachability and traces of the TPN. As for Timed Automata (TA), the abstraction may use an over-approximation operator on zones to enforce the termination. A coarser (and efficient) abstraction is then provided and proved exact w.r.t. marking reachability and traces (LTL properties).

Finally, we consider a subset of *TPN-TCTL* properties (*TPN-TCTL*$_S$) for which it is possible to propose efficient on-the-fly model-checking algorithms. Our approach consists in computing and exploring the zone based state space abstraction. On a practical point of view, the method is integrated in ROMEO [24], a tool for Time Petri Nets edition and analysis. In addition to the old features it is now possible to effectively verify a subset of TCTL directly on Time Petri Nets.

Index Terms

Time Petri Net, Model-checking, TCTL, zone based state space abstraction.

I. INTRODUCTION

Framework

Verification of concurrent systems is a complex task that requires powerful models and efficient analysis techniques. Model checking is one of the most popular verification techniques of concurrent systems. In this framework, the behavior of a system is represented by a finite (or infinite) transition system, and the properties to be verified are expressed in either a standard temporal logic (LTL, CTL, CTL*), or in its time extension (MITL, TCTL). Properties are checked by exploring the transition system.

This paper aims to apply the TCTL model checking techniques to Time Petri Nets.

The two main time extensions of Petri Nets are Time Petri Nets (TPN) [33] and Timed Petri Nets [37]. 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). Concerning the timing analysis of these three models ((T,P,A)–TPN), few studies have been realized about model-checking.

The class T-TPN is the most commonly-used subclass of TPN and in this paper we focus on this subclass that will be henceforth referred to as TPN. For classical TPN, boundedness and reachability are undecidable, and works on this model report undecidability results, or decidability under the assumption that the TPN is bounded (e.g. reachability in [36]).

Another common formalism to model timed systems is the one of Timed Automaton (TA). TA consists in a time extension of finite state automaton with clocks. Clocks constraints can be associated with transitions (guards) or with states (invariants) and are interpreted as firing constraints of the transition or as a delay constraint for idling in

a state. The model-checking of TCTL properties was proven decidable [3] and some efficient tools were developed to model-check both qualitative and quantitative time properties (*i.e.:* UPPAAL [30], KRONOS [41], HYTECH [29]).

Related Work

State space abstractions for TPN: We can find in the literature several state space abstractions for the *TPN* model. The well known are: the *State Class Graph (SCG)* [10], the *Geometric Region Graph (GRG)* [40], the *Strong State Class Graph (SSCG)* [13], the *Zone Based Graph (ZBG)* [25] and the *Atomic State Class Graphs ASCGs* [40], [13], [15]. These abstractions may differ from the class of properties they preserve. State Class Graphs provide finite representations for the state space of bounded TPN preserving either the LTL properties [10] or CTL* properties [13], [40], [15] of the state space. The Geometric Region Graph [40] preserves LTL properties and the region based algorithm in [34] preserves CTL properties on safe T-TPN without ∞ as latest firing time. The Zone Based Graph marking preserves reachability or LTL properties according to the convergence criterion selected to compute the state graph.

TCTL model checking of TPN: Given these abstractions, some methods were proposed to check quantitative timed properties but are not suitable for effective verification of properties on real-life systems. The first approach to verify quantitative properties was the translation of the TPN into a timed bisimilar TA. [19], [20] proposes a structural approach: each transition is translated into a TA using the same pattern. In [31], [32], the authors propose a method for building the State Class Graph of a bounded TPN as a TA. Such translations show that TCTL and CTL are decidable for bounded TPN and that developed algorithms on TA may be extended to TPN. Recently, Virbitskaite and Pokozy in [39] proposed a method to model-check TCTL properties on TPN. The method is based on the region graph and is similar to the one proposed by Alur and Dill [1] extended by Yovine and Tripakis [38], and Penczeck [35] for TA. The TCTL formula is translated into a CTL formula and then model-checked using classical CTL algorithms on a modified TA. However, the region graph is known to be a theoretical method which is not applicable in practice because of its lack of efficiency. In [28], authors proposed, using the observer and the state class graph methods, an on-the-fly model checking for a subclass of TCTL properties. However, the state class graph method is known [13] to be not suitable for the classical CTL model checking such as fix point technique.

Our contribution

We define a TCTL for TPN (*TPN-TCTL*) based on Generalized Mutual Exclusion Constraints (*GMEC*) [27]. A direct proof of the decidability of the model-checking of *TPN-TCTL* on bounded TPN is given as well as its complexity.

We propose a zone based state space abstraction that preserves marking reachability and traces of the TPN. As for Timed Automata (TA), the abstraction may use an over-approximation operator on zones to enforce the termination. A coarser (and efficient) abstraction is then provided and proved exact w.r.t. marking reachability and traces.

We finally consider a subset of *TPN-TCTL* properties (*TPN-TCTL*$_{\mathcal{S}}$) for which it is possible to propose efficient "on-the-fly" model-checking algorithms, that is, model-checking the property while the state space is computed. Thus, it allows to stop the construction as soon as the truth-hood of the property is known. The method, based on the zone based state space abstraction, is implemented and integrated in the tool ROMEO [24].

Outline of the paper

We first recall the definitions and semantics of TPN in section II. Section III is dedicated to *TPN-TCTL*, its decidability and its complexity. In section IV, we present a zone based forward method to compute the state space of a bounded TPN and in section V, some coarser approximations to reduce the effective cost of its computation. Finally, we consider in section VI the on-the-fly model-checking of a subset of *TPN-TCTL* (*TPN-TCTL*$_{\mathcal{S}}$) and provide algorithms based on the Zone Based Graph method. Practical examples are then considered in section VII. Finally, we conclude with our ongoing works and perspectives in Section VIII.

II. DEFINITIONS

A. Notations

We denote A^X the set of mappings from X to A. If X is finite and $|X| = n$, an element of A^X is also a vector in A^n. The usual operators $+, -, <$ and $=$ are used on vectors of A^n with $A = \mathbb{N}, \mathbb{Q}, \mathbb{R}$ and are the point-wise extensions of their counterparts in A. For a *valuation* $\nu \in A^X, d \in A$, $\nu+d$ denotes the vector $(\nu+d)(x) = \nu(x)+d$, and for $X' \subseteq X$, $\nu[X' \leftarrow 0]$ denotes the valuation ν' with $\nu'(x) = 0$ for $x \in X'$ and $\nu'(x) = \nu(x)$ otherwise. Finally, we denote $\mathbb{Q}_{\geq 0}$ (respectively $\mathbb{R}_{\geq 0}$) the set of positive (or null) rational (respectively real) numbers.

B. Timed Transition Systems

Definition 1 (Timed Transition Systems): A *timed transition system (TTS)* over the set of actions Σ is a tuple $S = (Q, Q_0, \Sigma, \longrightarrow)$ where Q is a set of states, $Q_0 \subseteq Q$ is the set of initial states, Σ is a finite set of actions disjoint from $\mathbb{R}_{\geq 0}$, $\longrightarrow \subseteq Q \times (\Sigma \cup \mathbb{R}_{\geq 0}) \times Q$ is a set of edges. We also write $q \xrightarrow{e} q'$ for $(q, e, q') \in \longrightarrow$.

In the case of $q \xrightarrow{d} q'$ with $d \in \mathbb{R}_{\geq 0}$, d denotes a delay and not an absolute time (q' is also denoted $q + d$). We assume that in any TTS there is a transition $q \xrightarrow{0} q'$ and in this case $q = q'$. Let $q_i \in Q$ be a state of S. A *run* ρ of q_i is a maximal (finite or infinite) sequence of alternating time and discrete transitions of the form:

$$\rho = q_i \xrightarrow{d_i} q_i + d_i \xrightarrow{a_i} q_{i+1} \xrightarrow{d_{i+1}} q_{i+1} + d_{i+1} \xrightarrow{a_{i+1}} \cdots$$

The total elapsed time of ρ, denoted $time(\rho)$, is the sum $\sum_{k=0}^{l} d_{i+k}$, l being the number of time transitions in ρ ($l = \infty$ if ρ is an infinite run). In case ρ is infinite, it is said to be *zeno* iff $time(\rho) < \infty$. The trace of ρ is the sequence $w_{untimed}(\rho) = (a_i)(a_{i+1}) \cdots$ and the timed trace of ρ is the sequence $w_{timed}(\rho) = (d_i, a_i)(d_{i+1}, a_{i+1}) \cdots$. ρ is a run of S iff $q_i \in Q_0$. The set of runs of q_i is denoted $\pi(q_i)$. The set of runs of S is the union of runs of its initial states ($\bigcup_{q_0 \in Q_0} \pi(q_0)$). Similarly, the set of traces (resp. timed traces) of S is the union of traces (resp. timed traces) of its initial states.

C. Time Petri Nets

Time Petri Nets (TPN) are a time extension of classical Petri nets. Informally, with each transition of the net is associated an implicit clock and an explicit time interval. The clock 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 2 (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,
- $\alpha : T \to \mathbb{Q}_{\geq 0}$ is the function giving the earliest firing times of transitions,
- $\beta : T \to \mathbb{Q}_{\geq 0} \cup \{\infty\}$ is the function giving the latest firing times of transitions.

As a shorthand, we note α_i and β_i for $\alpha(t_i)$ and $\beta(t_i)$.

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.

Let M be a marking of a TPN. M *enables* a transition t if $M \geq {}^\bullet t_i$. The set of transitions enabled by a marking M is noted $enabled(M)$.

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 $M - {}^\bullet t_i$ does not enable t_k. If t_i remains enabled after its firing then t_i is considered newly enabled. The set of transitions newly enabled by the firing of a transition t_i for a marking M is noted $\uparrow enabled(M, t_i)$.

We note x_i the clock associated with a transition $t_i \in T$ of the TPN. $v \in (\mathbb{R}_{\geq 0})^T$ is a *valuation* of the system. For any $t_i \in T$, $v(t_i)$ (also noted v_i), is the time elapsed since the transition t_i has been newly enabled *i.e.* the valuation of the clock x_i. $\bar{0}$ is the initial valuation with $\forall i \in [1..n], \bar{0}_i = 0$.

The semantics of a TPN is defined as a Timed Transition System (TTS).

A state of the TTS is a couple $q = (M, v)$ where M is a marking of the TPN and v a valuation of the system.

Definition 3 (Semantics of a TPN): The semantics of a TPN $\mathcal{N} = (P, T, {}^\bullet(.), (.)^\bullet, \alpha, \beta, M_0)$ is defined by the Timed Transition System $\mathcal{S}_\mathcal{N} = (Q, \{q_0\}, \Sigma, \rightarrow)$:

- $Q = \mathbb{N}^P \times (\mathbb{R}_{\geq 0})^T$
- $q_0 = (M_0, \bar{0})$
- $\Sigma = T$
- $\rightarrow \in Q \times (T \cup \mathbb{R}_{\geq 0}) \times Q$ is the transition relation including a discrete transition and a continuous transition.
  - The continuous transition is defined $\forall d \in \mathbb{R}_{\geq 0}$ by:

$$(M, v) \xrightarrow{d} (M, v') \text{ iff } \begin{cases} v' = v + d \\ \forall k \in [1..n],\ M \geq {}^\bullet t_k \Rightarrow v'_k \leq \beta_k \end{cases}$$

  - The discrete transition is defined $\forall t_i \in T$ by:

$$(M, v) \xrightarrow{t_i} (M', v') \text{ iff } \begin{cases} M \geq {}^\bullet t_i \\ M' = M - {}^\bullet t_i + t_i^\bullet \\ \alpha_i \leq v_i \leq \beta_i \\ \forall k \in [1, n],\ v'_k = \begin{cases} 0 \text{ if } t_k \in \uparrow \textit{enabled}\,(M, t_i) \\ v_k \text{ otherwise} \end{cases} \end{cases}$$

The Time Petri Net runs are those of its Timed Transition System. A TPN model is said to be *zeno* if at least one of its runs is zeno. The set of *reachable markings* of $\mathcal{N}$ is denoted $Reach(\mathcal{N})$. If the set $Reach(\mathcal{N})$ is finite we say that $\mathcal{N}$ is *bounded*. As a shorthand, we write $(M, v) \xrightarrow{(d,t)} (M', v')$ for a sequence of time elapsing and discrete steps like $(M, v) \xrightarrow{d} (M, v + d) \xrightarrow{t} (M', v')$.

III. TCTL for Time Petri Nets

In this section, we will introduce a TCTL for the TPN (*TPN-TCTL*) and prove that the model-checking of a *TPN-TCTL* formula on any bounded TPN is decidable and is a PSPACE-complete problem.

A. TCTL for Time Petri Nets

We propose a definition of TCTL [1], for the TPN, based on Generalized Mutual Exclusion Constraints (*GMEC*)[27]. A definition without *GMEC* was already introduced in [20] and translated into a standard TCTL for Timed Automata; the TCTL property is then checked on a structural translation of the TPN into TA.

The following definition extends the Generalized Mutual Exclusion Constraints (*GMEC*) introduced in [27].

Definition 4 (GMEC): Assume a TPN with m places $P = \{p_1, p_2, \cdots, p_m\}$. A Generalized Mutual Exclusion Constraint (*GMEC*) is inductively defined by:

$$\textit{GMEC} ::= \left(\sum_{i=1}^{m} a_i * \mathbf{M}(p_i) \right) \bowtie c \mid \varphi \vee \psi \mid \varphi \wedge \psi \mid \varphi \Rightarrow \psi$$

where $a_i \in \mathbb{Z}$, $\mathbf{M}$ is a keyword representing a marking, $p_i \in P$, $\bowtie\ \in \{<, \leq, =, >, \geq\}$, $c \in \mathbb{N}$ and $\varphi, \psi \in$ *GMEC*. Operators $\vee$,$\wedge$ and $\Rightarrow$ are defined as usual.

Intuitively, $\mathbf{M}(p_i) \bowtie c$ means that the current marking of the place p_i is in relation $\bowtie$ with c.

Given a marking M and a *GMEC* γ, we denote $M \models \gamma$ (respectively $M \not\models \gamma$) when the GMEC γ is true (respectively false) for the marking M.

Definition 5 (TCTL for TPN): The temporal logic *TPN-TCTL* is inductively defined by:

$$\textit{TPN-TCTL} ::= \textit{GMEC} \mid \mathbf{false} \mid \neg\varphi \mid \varphi \Rightarrow \psi \mid \exists \varphi\, \mathcal{U}_I \psi \mid \forall \varphi\, \mathcal{U}_I \psi$$

where **false** is a keyword, $\varphi, \psi \in$ *TPN-TCTL*, $I \in \{[a,b],[a,b),(a,b],(a,b)\}$ with $a \in \mathbb{N}$ and $b \in \mathbb{N} \cup \{\infty\}$.

Operators ($\neg$, $\Rightarrow$, $\wedge$, $\vee$ are defined as usual. We also use the following familiar abbreviations: $\mathbf{true} = \neg\mathbf{false}$, $\exists\Diamond_I\phi = \exists\mathbf{true}\,\mathcal{U}_I\phi$, $\forall\Diamond_I\phi = \forall\mathbf{true}\,\mathcal{U}_I\phi$, $\exists\Box_I\phi = \neg\forall\Diamond_I\neg\phi$, $\forall\Box_I\phi = \neg\exists\Diamond_I\neg\phi$.

The semantics of *TPN-TCTL* is defined on timed transition systems. Let us consider a TPN $\mathcal{N} = (P, T, {}^\bullet(.), (.)^\bullet, \alpha, \beta, M_0)$ and its semantics $S_\mathcal{N} = (Q, \{q_0\}, T, \rightarrow)$. The truth value of a *TPN-TCTL* formula for a state $q = (M, v)$ is given in the figure 1.

$$
\begin{array}{lll}
q \models \textit{GMEC} & \text{iff} & M \models \textit{GMEC} \\
q \not\models \mathbf{false} & & \\
q \models \neg\varphi & \text{iff} & q \not\models \varphi \\
q \models \varphi \Rightarrow \psi & \text{iff} & q \not\models \varphi \text{ or } q \models \psi \\
q \models \exists\varphi\,\mathcal{U}_I\psi & \text{iff} & \exists \rho = q_1 \xrightarrow{d_1} q_1 + d_1 \xrightarrow{t_1} q_2 ... \in \pi(q),\ \exists i > 0, \exists \delta \in [0, d_i],\ \text{s.t.} \\
 & & \begin{cases} \left(\sum_{j=0}^{i-1} d_j\right) + \delta \in I \text{ (with } d_0 = 0) \text{ and } q_i + \delta \models \psi \\ (\forall \delta' \in [0, \delta[, q_i + \delta' \models \varphi) \\ (\forall j \text{ s.t. } 0 < \mathrm{j} < \mathrm{i}, \forall \delta' \in [0, \mathrm{d_j}], \mathrm{q_j} + \delta' \models \varphi) \end{cases} \\
q \models \forall\varphi\,\mathcal{U}_I\psi & \text{iff} & \forall \rho = q_1 \xrightarrow{d_1} q_1 + d_1 \xrightarrow{t_1} q_2 ... \in \pi(q),\ \exists i > 0, \exists \delta \in [0, d_i],\ \text{s.t.} \\
 & & \begin{cases} \left(\sum_{j=0}^{i-1} d_j\right) + \delta \in I \text{ (with } d_0 = 0) \text{ and } q_i + \delta \models \psi \\ (\forall \delta' \in [0, \delta[, q_i + \delta' \models \varphi) \\ (\forall j \text{ s.t. } 0 < \mathrm{j} < \mathrm{i}, \forall \delta' \in [0, \mathrm{d_j}], \mathrm{q_j} + \delta' \models \varphi) \end{cases}
\end{array}
$$

Fig. 1. Semantics of *TPN-TCTL*

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

B. Decidability and complexity of TPN-TCTL

We give a short direct proof of the decidability of *TPN-TCTL* based on the *region graph* [1], [3]. Note that the decidability was already proven in [20], [32] as a corollary of bisimilar translations from TPN to TA. Moreover we prove that the model-checking of a *TPN-TCTL* formula on a bounded TPN is a PSPACE-complete problem.

1) Decidability of TPN-TCTL: Let $\mathcal{N} = (P, T, {}^\bullet(.), (.)^\bullet, \alpha, \beta, M_0)$ be a TPN and $S_\mathcal{N} = (Q, \{q_0\}, T, \rightarrow)$ its TTS.

Definition 6 (Clock equivalence – Region): Let $q = (M, v) \in Q$ be a state of a TPN and C_M the set of clocks associated with transitions enabled for marking M. For each clock $x_i \in C_M$ associated with transition t_i, let $c_i = \beta_i$ if $\beta_i \neq \infty$ else $c_i = \alpha_i$. For $a \in \mathbb{R}_{\geq 0}$ we denote $\lfloor a \rfloor$ the integer part of a and $\langle a \rangle$ its fractional part.

The equivalence relation $\simeq$ is defined over the set of states Q. $(M, v) \simeq (M', v')$ iff all the following conditions hold:

1) $M = M'$
2) For all $x_i \in C_M$, either $\lfloor v(x_i) \rfloor$ and $\lfloor v'(x_i) \rfloor$ are the same, or both $v(x_i)$ and $v'(x_i)$ are greater than c_i.
3) For all $x_i, x_j \in C_M$ with $v(x_i) \leq c_i$ and $v(x_j) \leq c_j$, $\langle v(x_i) \rangle \leq \langle v(x_j) \rangle$ iff $\langle v'(x_i) \rangle \leq \langle v'(x_j) \rangle$.
4) For all $x_i \in C_M$ with $v(x_i) \leq c_i$, $\langle v(x_i) \rangle = 0$ iff $\langle v'(x_i) \rangle = 0$.

A *region* is an equivalence class of clock valuations induced by the relation $\simeq$.

Intuitively, the meaning of such a definition is that the integral parts allow to determine whether or not a particular firing condition is satisfied whereas the ordering of the fractional parts is necessary to decide which clock will change its integral part first (and thus the successor region) when the time elapses.

In dimension 2 with $c_i = 3$ and $c_j = 2$ we obtain the partition of the figure 2. The regions are: integer points, open segment between two integer points, inside of the surfaces delimited by the segments.

For example, regions r_1 and r_2 of figure 2 can be described by : $r_1 : (2 < v(x_i) < 3) \wedge (1 < v(x_j) < 2) \wedge 0 < \langle v(x_j) \rangle < \langle v(x_i) \rangle$ and $r_2 : (v(x_i) > 3) \wedge (1 < v(x_j) < 2)$

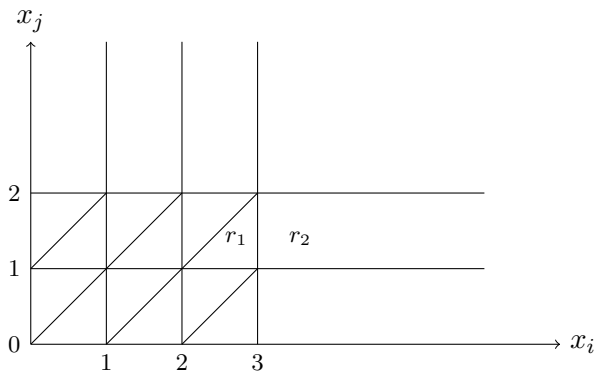

Fig. 2. Partition for 2 clocks x_i and x_j with $c_i = 3$ and $c_j = 2$

Definition 7 (TPN-Region Graph): Given a TPN $\mathcal{N}$ and its timed transitions system $\mathcal{S}_{\mathcal{N}}$, the TPN-region graph is the quotient graph of $\mathcal{S}_{\mathcal{N}}$ with respect to the equivalence relation $\simeq$.

In [2], [1], authors show that TCTL model-checking can be performed by computation of the region graph of timed graph with an additional clock x used to measure time elapsing between configurations. Let φ be any TCTL-formula. For every subscript $\sim c$ appearing in a formula φ let there be a new proposition $p_{\sim c}$. A vertex (s, γ) (where γ is an equivalence class induced by the clock equivalence relation) is labeled with $p_{\sim c}$ if $\gamma \models x \sim c$, else with $\neg p_{\sim c}$. Then vertexes are labelled with syntactic subformulas of φ, or their negation, starting from subformulas of length 1, length 2, and so on. To label the vertex (s, γ) of the region graph satisfying $\exists \varphi_1 U_{\sim c} \varphi_2$, it needs to check the formula $\exists \varphi_1 U (p_{\sim c} \wedge \varphi_2)$ on state $(s, \gamma[x \leftarrow 0])$

The TPN-region graph then allows to prove the following theorems for TPN (we refer the reader to [3], [2], [1] for detailed proofs of these two theorems):

Theorem 1 (TPN-TCTL decidability): TPN-TCTL is decidable for bounded TPN.

Theorem 2: Given a k-bounded TPN $\mathcal{N}$, its number of regions is bounded by:

$$(k+1)^{|P|}.|T|! \cdot 2^{|T|} \cdot \prod_{t_i \in T} (2 \cdot c_i + 2)$$

Proof: For a fixed marking, the number of equivalence classes of clock valuations induced by the relation $\simeq$ is bounded (like in [3]) by $|T|! \cdot 2^{|T|} \cdot \prod_{t_i \in T} (2 \cdot c_i + 2)$. Moreover, the number of markings of a k-bounded TPN, is bounded by $(k+1)^{|P|}$. ■

2) Complexity of TPN-TCTL: To our knowledge reachability and model-checking complexities are still open for TPN. For a state (M, ν), the value of a clock associated with a transition which is not enabled by the marking M does not matter. The region graph proposed here is then exponential in the size of the input i.e. the maximum number of transitions simultaneously enabled + the number of places of a TPN. However, the following theorem holds :

Theorem 3 (PSPACE-completeness): The model-checking of a *TPN-TCTL* formula on a bounded TPN is a PSPACE-complete problem.

Proof:

- First, there exists a PSPACE algorithm based on the TPN-region graph to decide *TPN-TCTL* on bounded TPN: For k-bounded TPN, each marking belongs to $([0..k])^P$. By encoding in binary, marking can be represented in $|P|.log_2(k+1)$ bits i.e. in polynomial space. Then, a region can be represented in polynomial space too. Since the number of regions of the TPN region graph is exponential in the size of the input l (the number of transitions simultaneously enabled + the number of places of a TPN), if we construct it entirely, and label its vertices with the subformulas of a TPN-TCTL properties, the algorithm will need space exponential in l. However, we can proceed like in [1]: for formulas of the form $\varphi = \exists \varphi_1 U_I \varphi_2$, it is possible to check TCTL formulas by nondeterministically exploring a path, region by region. The path is labelled on-the-fly by a

polynomial space procedure like in [1] (A recursive procedure $label(vertex, \varphi)$ which returns $true$ if $vertex$ should be labelled with φ otherwise $false$). It just needs to remember the current guess and the previous guess. By Savitch's theorem, the deterministic version of the algorithm uses only polynomial space. The cases $\varphi = GMEC$, $\varphi = false$ and $\varphi = \varphi_1 \Rightarrow \varphi_2$ are straightforward. For the case $\varphi = \forall \varphi_1 U_I \varphi_2$, the negation of φ can be written using existential path-quantifiers.

- The PSPACE-hardness is a consequence that reachability in untimed 1-safe Petri net is a PSPACE-complete problem [21].

■

IV. State space abstraction

Since we consider a dense time semantics for our model of TPN, the set of configurations of its Timed Transitions Systems cannot generally be exhaustively enumerated. The region graph presented in the previous section abstracts finitely and in a correct way the behaviors of timed automata or bounded time Petri nets. However, such a construction does not support a natural implementation as it runs up against the state explosion problem. We will present, in this section, an abstraction of the state space of a bounded TPN based on convex union of regions. This abstraction will allow to develop a complete framework for the analysis and model-checking of reachability properties (reachability of markings), both linear (LTL) and branching (CTL) properties and quantitative branching properties (TCTL).

A. Zone based state space computation

We briefly present the method used to compute the state space of a TPN using a zone based forward algorithm. We refer the reader to [25], [26] for a detailed presentation of the algorithm. Model-checking algorithms will be adaptations of this algorithm.

1) Definitions: We first give some classical definitions.

Definition 8 (Zone): Let X be a finite set of clocks. A *zone* over X is a convex set of clock valuations representing a conjunction of atomic constraints of the form $x_i - x_j \prec c_{ij}$, $x_i \prec c_{i0}$, $-x_j \prec c_{0j}$ where $x_i, x_j \in X$, $c_{ij}, c_{i0}, c_{0j} \in \mathbb{Q} \cup \{-\infty, \infty\}$ and $\prec \in \{\leq, <, =, \geq, >\}$.

Let x_0 be a clock zero[1] and Z a zone over set of clocks X. All atomic constraints of Z can be rewritten as constraints on clock differences $x_i - x_j \prec z_{ij}$, with $x_i, x_j \in X \cup \{x_0\}$, $z_{ij} \in \mathbb{Q} \cup \{\infty\}$ and $\prec \in \{\leq, <\}$, i.e.:

$$Z = \bigcap_{x_i, x_j \in X \cup \{x_0\}} (x_i - x_j \prec z_{ij})$$

A zone is a particular form of polyhedra that can be efficiently encoded using DBM (Difference Bounded Matrices [11], [23]). This data structure allows easier representation and lower complexity algorithms (at most cubic in the number of clocks of the DBM) than general polyhedra.

Though a set of clocks valuations may be expressed by different conjunctions of constraints, there exists a unique form, called canonical form, that allows to compare zones together. A canonical form of a zone Z is the representation with tightest bounds on all clock differences. Its computation is based on the shortest path *Floyd-Warshall*'s algorithm and is considered as the most costly operation (cubic in the number of clocks of the zone). A zone Z is then in canonical form iff $\forall x_i, x_j \in (X \cup \{x_0\}), z_{ij} = Sup_Z(x_i - x_j)$, where $Sup_Z(x_i - x_j)$ is the supremum (i.e. the least upper bound) of $x_i - x_j$ in Z. In all the following, we will only consider the canonical form of a zone, that is any zone Z will stand for its canonical form. Therefore, the set of clocks of Z contains the clock x_0.

Let Z be a zone over a set of clocks X and $X' \subseteq X$ a subset of clocks of X. The restriction of Z to X' denoted $Z_{|X'}$ is the zone Z' obtained by putting Z in canonical form and then eliminating all variables of $X - X'$.

[1]The value of this clock is always 0.

The future of Z, denoted $\overrightarrow{Z}$, is the zone obtained by putting Z in canonical form and then replacing each constraint of the form $x_i - x_0 \prec z_{i0}$, where $x_i \neq x_0$, by $x_i - x_0 < \infty$.

Definition 9 (Symbolic state): A *symbolic state* of a TPN is a couple (M, Z) where M is a marking and Z is a zone whose clocks are those associated with transitions enabled by the marking M.

Informally, the symbolic state (M, Z) represents the set of clock valuations (Z) for which the marking M is reachable according with the TPN semantics.

Definition 10 (Discrete Successors): Let $s = (M, Z)$ be a symbolic state and t_i a transition of $enabled(M)$. Transition t_i is firable from s iff the zone $Z \cap \{x_i \geq \alpha_i\}$ is non-empty. If t_i is firable from s, then the set of *discrete successors* of s by the firing of transition t_i is:

$$Post_{t_i}(s) = \left(M', Z'\right) \text{ with } \begin{cases} M' = M - {}^{\bullet}t_i + t_i^{\bullet} \\ Z' = \left(\left(Z \cap \{x_i \geq \alpha_i\}\right)_{|O}\right) \cap \bigcap\limits_{x_j \in N} \{x_j = 0\} \end{cases}$$

$Z \cap \{x_i \geq \alpha_i\}$ represents the set of valuations for which the transition t_i is firable. O is the set of clocks of transitions which are enabled in M' but not newly enabled. N is the set of clocks newly enabled by the firing of transition t_i in the marking M (*i.e.* the clocks associated with transitions of $\uparrow$*enabled* (M, t_i)).

$Post_{t_k}(s)$ represents all states that are reachable from s by the firing of the discrete transition t_k.

Definition 11 (Time Successors): Let $s = (M, Z)$ be a symbolic state and X the clock set of Z. The set of *time successors* of s is the symbolic state:

$$\overrightarrow{Post}(s) = (M, Z') \text{ with } Z' = \overrightarrow{Z} \cap \bigcap_{x_i \in X - \{x_0\}} \{x_i \leq \beta_i\}$$

$\overrightarrow{Post}(s)$ is the set of states that are reachable from s by elapsing time in the marking M. Time can grow until a clock reaches its latest firing time.

In addition we note :

$$\overrightarrow{Post}_{t_i}(s) = \overrightarrow{Post}\left(Post_{t_i}(s)\right)$$

2) State Space Computation Algorithm: The aim of the algorithm is to iteratively compute from a symbolic state the set of reachable symbolic states. Starting from the initial symbolic state $\overrightarrow{Post}((M_0, Z_0))$ (where M_0 is the initial marking and clocks of Z_0 are all equal to 0) of a TPN, successor symbolic states are iteratively computed as follows: Let $s = (M, Z)$ be a reachable symbolic state. For each transition t_k firable in the symbolic state s:

1) Compute the discrete successor $s' = Post_{t_k}(s)$, that is the symbolic state resulting by firing transition t_k.
2) Compute the time successor $s" = \overrightarrow{Post}(s')$, that is all states (including those of s') reachable by time elapsing from those of s'.

The construction of the ZBG is given in Algorithm 1, where $Waiting$ is a list to store the symbolic states to be analyzed and $Passed$ symbolic states that were analyzed. According to the criterion of convergence selected (equality of symbolic states or inclusion *i.e.* how $s \notin Passed$ is coded), one can compute the marking graph of the TPN or a graph preserving markings and traces of the TPN (LTL properties).

3) k-approximation: If the net contains some transitions with ∞ as latest firing time, a last step consisting in applying an approximation operator is needed to ensure the convergence, else the algorithm exactly computes the state space of the TPN. Actually, if we consider the TPN of figure 3, the infinite firing sequence $(t_2, t_3)^*$ generates the infinite sequences of zones $Z_i = \{2i \leq x_1 \leq 2i + 1 \wedge x_1 - x_2 = 2i\}$, even if the model is bounded (i.e. has a finite number of reachable markings).

Algorithm 1 Zone Based Graph Algorithm

$s_0 = (M_0, Z_0)$
$Waiting \leftarrow \{\overrightarrow{Post}(s_0)\}$
$Passed \leftarrow \emptyset$
while $Waiting \neq \emptyset$ **do**
  $s = pop(Waiting)$
  if $s \notin Passed$ **then**
    for all $t \in enabled(s)$ **do**
      $s' = \overrightarrow{Post}(Post_t(s))$
      $Waiting \leftarrow Waiting \cup \{s'\}$
    end for
    $Passed \leftarrow Passed \cup \{s\}$
  end if
end while

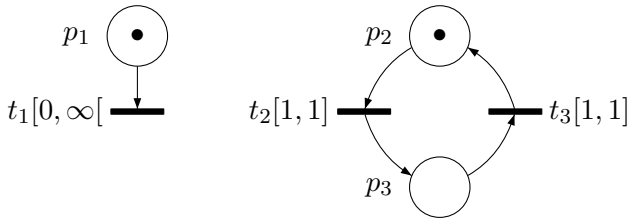

Fig. 3. Bounded TPN with an unbounded number of zones

V. Approximations of zones

A. Introduction

Approximation of zones is needed, for TPN with unbounded transitions to ensure the finiteness for clock based abstractions (region as well as zone based). This operation takes into account the constants with which clocks are compared and consists of extending domains of clocks whilst preserving reachable markings and traces of the model (i.e. each state added by an approximation can be simulated by a state within the originate zone).

Moreover, the coarser we may choose the abstraction, the more efficient the method will be.

Related work: The classical abstraction method for timed automata, known under the names *normalization*, *extrapolation* or *k-approximation*, takes into account the maximum constants to which the various clocks are compared. [22] describes this method and a number of additional abstraction techniques including active clock reduction and the over-approximating convex hull abstraction. In [4], the maximum constants used in the abstraction do not only depend on the clocks but also on the particular locations of the timed automata. In [5], [6], authors show that, by distinguishing maximal lower and upper bounds, significant coarser abstractions can be obtained.

Most of the above zone based abstractions require that guards are restricted to conjunctions of simple lower or upper bounds on individual clocks. We refer the reader to [18] to learn more about zone based abstractions in the presence of difference constraints, and to [4] for a solution based on zone splitting.

Specificities of TPN: Contrary to TA, in TPN, a transition cannot be disabled by elapsing time [8]. It means that if a transition is enabled in a particular state q, for all runs starting from q, either this transition is fired or is disabled by the firing of another transition. A consequence is that the notion of active clocks can obviously be used by considering in a zone only the enabled transitions. Moreover, we can search improvement of *k-approximation* by focussing our attention on the following case : a valuation of a clock cannot exceed its maximal bound.

In this section, we apply the *k-approximation* to TPN (as in [25], [26]) and we propose

- an adaptation of the well known improvement on TA [22], [7] based on a different value of k for each unbounded transition.

- an improvement of the method leading to an abstraction that we show coarser than that of [6].

Moreover, we prove that these abstractions are exact w.r.t. reachable markings and traces.

B. k-approximation

The first approximation proposed for timed automata [22] consists of selecting an integer k equal to the maximum finite constant appearing in the time constraints of the model. This approximation is based on the fact that once the clock has over passed the value k, its precise value does not matter anymore. In the context of time Petri nets, k is equal to the maximum finite constant appearing in the firing time intervals of the TPN. Each symbolic state (M, Z) is then approximated in (M, Z') using the following $k\text{-}approx$ function:

Definition 12 (k-approx): Let Z be a zone (in canonical form) over a set of clocks X. $k\text{-}approx\,(Z) = Z'$ is defined by:

$$\forall x_i, x_j \in X, \quad z'_{ij} = \begin{cases} \infty & \text{if } z_{ij} > k \\ -k & \text{if } z_{ij} < -k \\ z_{ij} & \text{otherwise.} \end{cases}$$

The algorithm then computes at each iteration:

$$\overrightarrow{Post}^k_{t_i}(s) = k\text{-}approx\left(\overrightarrow{Post}_{t_i}(s)\right)$$

Recent works on Timed Automaton [17], [18] proved that this operation generally leads to an over-approximation of the reachable localities of TA. For TPN, [25], [26] prove the correctness of the zone based abstraction with k-approximation w.r.t. the set of reachable markings and traces of the TPN.

C. k_x-approximation

The first improvement of *k-approximation* is to consider a distinct value k_{x_i} for each clock x_i. k_{x_i} is equal to the maximum finite constant appearing in time constraints on clock x_i. The underlying idea is: if a clock x_i grows beyond the value k_{x_i}, its precise value does not matter and the resulting zone has exactly the same behavior than its originate symbolic state (M, Z) w.r.t. traces.

Definition 13 (k_x-approx): Let Z be a zone (in canonical form) over a set of clocks X.
$k_x\text{-}approx(Z) = Z'$ is defined by:

$$\forall x_i, x_j \in X, \quad z'_{ij} = \begin{cases} \infty & \text{if } z_{ij} > k_{x_i} \\ -k_{x_j} & \text{if } z_{ij} < -k_{x_j} \\ z_{ij} & \text{otherwise.} \end{cases}$$

For time Petri nets, clocks are associated with transitions and compared with bounds of static firing intervals of their transitions. Therefore, for a clock x_i of a transition t_i, its k_{x_i} is defined by:

$$k_{x_i} = \begin{cases} \alpha_i & \text{if } \beta_i = \infty \\ \beta_i & \text{otherwise.} \end{cases}$$

In addition, for any bounded transition t_i (i.e. $\beta_i \neq \infty$), its clock cannot overpass β_i. Note that, in this case, $k_{x_i} = \beta_i$. Then, the following relations hold: $\forall t_i, t_j \in enabled(M)$,

$$(1) \quad \beta_i \neq \infty \quad \Rightarrow \quad z_{ij} \leq z_{i0} \leq \beta_i = k_{x_i}$$

$$(2) \quad \beta_j \neq \infty \quad \Rightarrow \quad -k_{x_j} = -\beta_j \leq z_{0j} \leq z_{ij}$$

Using the above relations, we can rewrite the $k_x\text{-}approx$ for time Petri nets as follows:

$$\forall x_i, x_j \in X, \quad z'_{ij} = \begin{cases} \infty & \text{if } z_{ij} \geq \alpha_i \wedge \beta_i = \infty \\ -\alpha_j & \text{if } z_{ij} \leq -\alpha_j \wedge \beta_j = \infty \\ z_{ij} & \text{otherwise.} \end{cases}$$

By convention we suppose that $\alpha_0 = \beta_0 = 0$. Note that since intervals of time Petri nets are closed by the left, we have replaced the strict relation ($<$) between z_{ij}, k_{x_i} and k_{x_j} by a larger one ($\leq$).

By applying the forward algorithm using k_x-$approx$ instead of k-$approx$, we then build a coarser abstraction (and consequently less symbolic states to explore) but still exact with respect to reachable markings and traces. We refer the reader to section V-F for the proof of its exactness.

D. Lower and Upper bounds based approximations

In [6], authors have proposed two other approximations for timed automata, referred in the sequel by LU_x-$Approx$ and LU_x^*-$Approx$. In these approximations, authors distinguish between lower and upper bounds of clocks. They define, for each clock x_i, two maximal bounds denoted respectively $L(x_i)$ and $U(x_i)$. $L(x_i)$ and $U(x_i)$ are respectively the maximal lower bound and the maximal upper bound of clock x_i in the time constraints of the model. More precisely, $L(x_i)$ (resp. $U(x_i)$) is the maximal constant k such that there exists, in the model, a constraint $x_i \geq c$ or $x_i > c$ (resp. $x_i \leq c$ or $x_i < c$). If such a constant does not exist, $L(x_i)$ (resp. $U(x_i)$) is set to $-\infty$.

Definition 14 (LU_x-$Approx$): Let Z be a zone (in canonical form) over a set of clocks X. LU_x-$Approx(Z) = Z'$ is defined by:

$$\forall x_i, x_j \in X, \quad z'_{ij} = \begin{cases} \infty & \text{if } z_{ij} > L(x_i) \\ -U(x_j) & \text{if } -z_{ij} > U(x_j) \\ z_{ij} & \text{otherwise.} \end{cases}$$

Definition 15 (LU_x^*-$Approx$): Let Z be a zone (in canonical form) over a set of clocks X. LU_x^*-$Approx(Z) = Z'$ is defined by:

$$\forall x_i, x_j \in X, \quad z'_{ij} = \begin{cases} \infty & \text{if } z_{ij} > L(x_i) \\ \infty & \text{if } -z_{0i} > L(x_i) \\ \infty & \text{if } -z_{0j} > U(x_j), i \neq 0 \\ -U(x_j) & \text{if } -z_{0j} > U(x_j), i = 0 \\ z_{ij} & \text{otherwise.} \end{cases}$$

In the context of time Petri nets, for each clock x_j, $L(x_j) = \alpha_j$, $U(x_j) = \beta_j$ if $\beta_j \neq \infty$, $U(x_j) = -\beta_j$ otherwise. Moreover, x_j cannot neither be negative nor overpass β_j, it follows that $\forall t_i, t_j \in enabled(M)$,

$$(3) \quad -\beta_j \leq z_{0j} \leq z_{ij} \leq z_{i0}$$

Then, the following relations always holds: $\forall t_i, t_j \in enabled(M)$, $U(x_j) = \beta_j \leq z_{0j} \leq z_{ij}$, if $\beta_j \neq \infty$, $U(x_j) = -\beta_j \leq z_{0j} \leq z_{ij}$ otherwise.
Using the above relations, LU_x-$Approx(Z) = Z'$ can be simplified (by eliminating relations which never hold) and rewritten for time Petri nets as follows:

$$\forall x_i, x_j \in X, \quad z'_{ij} = \begin{cases} \infty & \text{if } z_{ij} > \alpha_i \\ \infty & \text{if } (\beta_j = \infty) \\ z_{ij} & \text{otherwise.} \end{cases}$$

LU_x^*-$Approx(Z) = Z'$ can be simplified and rewritten for time Petri nets as follows:

$$\forall x_i, x_j \in X, \quad z'_{ij} = \begin{cases} \infty & \text{if } z_{ij} > \alpha_i \\ \infty & \text{if } -z_{0i} > \alpha_i \\ \infty & \text{if } (\beta_j = \infty) \\ z_{ij} & \text{otherwise.} \end{cases}$$

E. Our approximation

We propose to use another approximation operation which is exact and leads to much compact graphs. We refer the reader to the section V-F, lemma 2 and theorem 4 for the proof of its exactness.

Definition 16 (k'_x-approx): Let (M, Z) be a symbolic state in canonical form and X the clock set of Z. Our approximation of Z, denoted k'_x-$approx(Z)$, is the canonical form of the DBM Z' defined as follows:

$$\forall x_i, x_j \in X, \quad z'_{ij} = \begin{cases} \infty & \text{if } \beta_j = \infty \\ \infty & \text{if } z_{ij} - \alpha_i \geq z_{0j} \\ z_{ij} & \text{otherwise.} \end{cases}$$

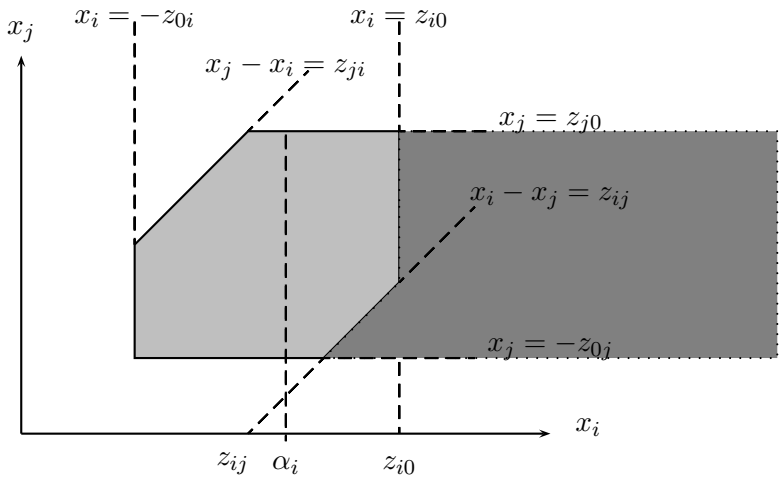

Fig. 4. a DBM (light gray area) and its k'_x-$approx$ (light + dark gray area)

The comparison between [6] is depicted in figure 4. This figure shows that with our approximation the clock domain of x_i is extended earlier ($z_{ij} - z_{0j} \geq \alpha_i$) than with approximations proposed in [6] ($z_{ij} \geq \alpha_i$). Indeed, for the DBM (light gray area) of the figure 4, we have $z_{ij} \ngeq \alpha_i$ and the approximation of [6] can not be applied whereas $z_{ij} - z_{0j} \geq \alpha_i$ leading with our approximation to add the dark gray area. Such a situation can be obtained, for example, with the TPN of figure 5 after firing of the sequence t_1, t_2, t_3.

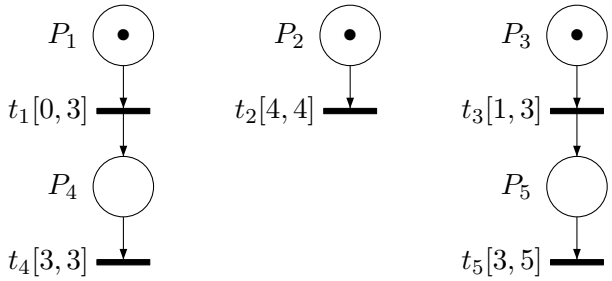

Fig. 5. Time Petri Net leading to the DBM of figure 4.

The following lemma establishes that our approximation leads to coarser graphs than those obtained using LU^*_x-$Approx$. Indeed, our approximation produces larger zones than those obtained by LU^*_x-$Approx(Z)$ and then leads to coarser abstractions.

Lemma 1: For every Z, it holds that LU^*_x-$Approx(Z) \subseteq k'_x$-$Approx(Z)$.

Proof: It suffices to show that $\forall x_i, x_j \in En(M)$, if $z_{ij} > \alpha_i \ \vee \ -z_{0i} > \alpha_i$ then $z_{ij} - \alpha_i \geq z_{0j}$.

1) If $z_{ij} > \alpha_i$ then $z_{ij} - \alpha_i > 0$. Since $z_{0j} \leq 0$, it follows that $z_{ij} - \alpha_i \geq z_{0j}$.

2) If $-z_{0i} > \alpha_i$ then relation $z_{0i} + z_{ij} \geq z_{0j}$ implies that $-z_{0j} + z_{ij} \geq -z_{0i} > \alpha_i$. Then $z_{ij} - \alpha_i \geq z_{0j}$. Moreover, since $\alpha_i \geq 0$, relation $-z_{0i} > \alpha_i$ implies that $i \neq 0$. ■

F. Correctness of our zone based forward method

We now prove that the Zone Based Graph (ZBG) computed with k_x-$approximation$ or k'_x-$approximation$ is exact i.e. both sound[2] and complete.[3]

We have then to show that a symbolic state (M, Z) and its approximation (as proposed in section V-E) have exactly the same traces. To achieve this goal, we prove, in lemma 2, that a zone Z and its approximation have the same firing domain. More precisely, a symbolic state and its approximation map to the same *Berthomieu*'s state class [13]. We first show, in proposition 1, how to compute the firing domain of a zone. The idea of this proposition is borrowed from [14] and adapted to zones. In [14], authors have shown how to compute the firing domain of their state classes. Their state classes are based on past firing instants while *Berthomieu*'s state classes are based on future firing instants.

Definition 17: The firing domain of a symbolic state [14], [13].
Let (M, Z) be a symbolic state (Z is in canonical form) and $q = (M, v)$ one of its states. The firing domain I_v of $q = (M, v)$ is defined by I_q : *enabled* $(M) \rightarrow \mathbb{Q}_{\geq 0} \times (\mathbb{Q}_{\geq 0} \cup \{\infty\})$, $I_q(t_i) = [max(0, \alpha_i - v_i), \beta_i - v_i]$.
The firing domain of (M, Z) is the union of the firing domains of all its states, i.e.: $\{I_q \mid q \in (M, Z)\}$

Proposition 1: Let (M, Z) be a symbolic state (Z is in canonical form) and X the set of clocks of Z. The canonical form of its firing domain D is:

$$\forall x_i, x_j \in X, \quad d_{ij} = \begin{cases} \beta_i + min(z_{0i}, z_{ji} - \alpha_j) & \text{if } i \neq j \\ 0 & \text{otherwise.} \end{cases}$$

Proof: First we have to show that the firing domain D of (M, Z) is characterizable by a set of atomic constraints. The following steps show how to do it:
Let $Y = \{y_i \mid x_i \in X\}$ and $Y' = \{y'_i \mid x_i \in X\}$ be two disjoint sets of new variables ($Y \cap X = Y' \cap X = Y \cap Y' = \emptyset$). We suppose that $x_0 = y_0 = y'_0 = 0$.

1) Initialize D with
$$Z \cap \bigcap_{x_i \in X - \{x_0\}} \{0 \leq y_i \wedge \alpha_i - x_i \leq y_i \leq \beta_i - x_i\} \quad (1)$$
Note that variables of Y represent delays.
2) Replace in D each y_i of Y by $-y'_i$. We can then rewrite it to obtain:
$$Z \cap \bigcap_{x_i \in X - \{x_0\}} \{y'_i \leq 0 \wedge y'_i - x_i \leq -\alpha_i \wedge x_i - y'_i \leq \beta_i\} \quad (2)$$
3) Put D in canonical form, rename x_0 in y_0 and eliminate all variables x_i.
4) Replace in D each y'_i by $-y_i$.

The resulting D is a set of atomic constraints. Therefore, D is convex and has a unique canonical form defined by: $\forall y_i, y_j \in Y, d_{ij} = Sup_D(y_i - y_j)$.
The rest of the proof is based on the *constraint graphs*. Let F be a set of atomic constraints. The constraint graph of F is a weighted directed graph, where nodes represent variables of F, with one extra node representing the value 0. An arc connecting a node u_i to a node u_j with weight $(\prec, c)$, where $\prec$ is either $<$ or $\leq$, represents any atomic constraint equivalent to $u_i - u_j \prec c$.
We have the two following known results:

[2]Any trace in the model state space is also possible in the ZBG.
[3]Any trace in the ZBG is also possible in the model state space.

- F has, at least, one solution (one tuple of values that satisfies, at once, all constraints in F) if and only if, the constraint graph of F has no negative cycle,
- If the constraint graph of F has no negative cycle, the weight of the shortest path, in the graph, going from a node u_i to a node u_j, is equal to the supremum of $(u_i - u_j)$ in F ($Sup_F(u_i - u_j)$).

Consider now the set of atomic constraints (2) above obtained at step 2. Its constraint graph is shown in figure 6, where solid arrows represent constraints of Z and dashed ones are constraints to be added to obtain the constraint graph of the firing domain D of Z. Recall that x_0 and y_0 are respectively "zero" nodes of constraint graphs of Z and D.

Since Z is not empty, (2) is consistent and then its constraint graph has no negative cycle. In addition, since for all $x_i \in X, y'_i = -y_i$ and $x_0 = y_0 = y'_0 = 0$, it follows that the weight of the shortest path, in the constraint graph of formula (2), connecting a node y'_i to a node y'_j is equal to d_{ji}. By assumption, Z is in canonical form. The weight of the shortest path connecting a node x_i to a node x_j, in the constraint graph of Z, is then z_{ij}. Knowing that for all $x_i \in X, 0 \leq x_i \leq \beta_i$, it follows that: for all $x_i, x_j \in X, 0 \leq z_{i0} \leq \beta_i$, $z_{0i} \leq 0$ and $z_{ij} \leq z_{i0} + z_{0j} \leq \beta_i + z_{0j}$. Then, the added nodes and edges, to obtain the constraint graph of formula (2), do not affect weights of the shortest paths connecting nodes of X. It follows that:

$$d_{ji} = min(0 + z_{0j} + \beta_j, -\alpha_i + z_{ij} + \beta_j)$$

■

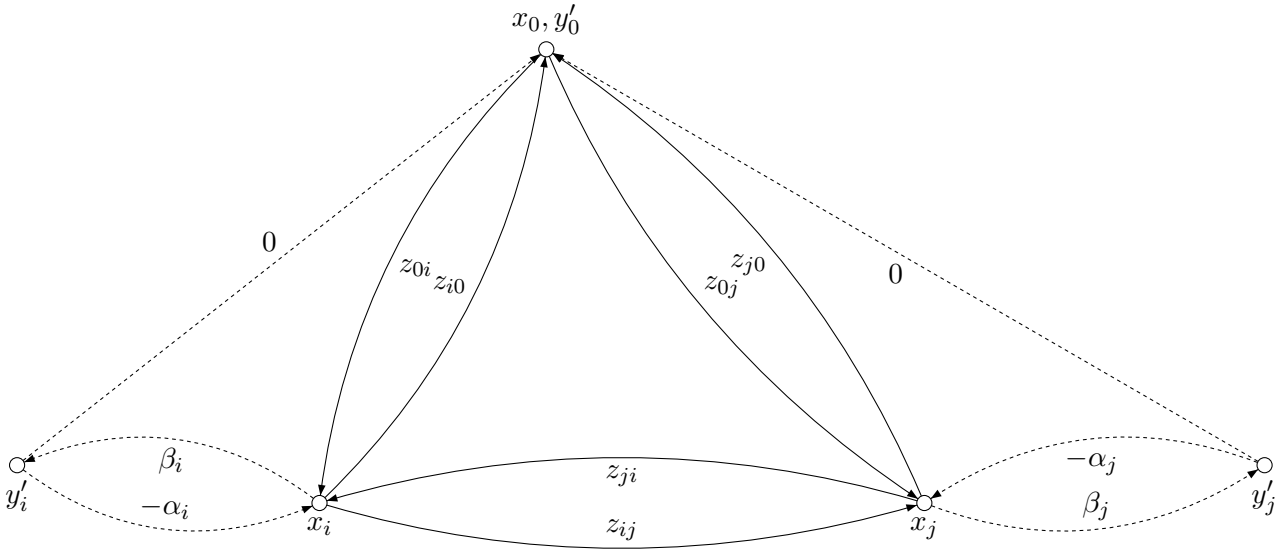

Fig. 6. Constraint graph of $Z \cap \bigcap_{x_i \in X - \{x_0\}} \{y'_i \leq 0 \wedge y'_i - x_i \leq -\alpha_i \wedge x_i - y'_i \leq \beta_i\}$

Lemma 2: Let (M, Z) be a symbolic state in canonical form, X its set of clocks, (M, Z') and (M, Z'') be respectively its k_x-approximation and k'_x-approximation.
Then: (M, Z'), (M, Z') and (M, Z'') have the same firing domain.

Proof: Let D, D' and D'' be respectively the firing domains of (M, Z), (M, Z') and (M, Z''). We have to show that $D = D' = D''$.
Note that we give only here the proof of $D = D''$. The proof of $D = D'$ is very similar. Recall the canonical forms of D, D'' and Z'': $\forall x_i, x_j \in X$,

$$d_{ji} = \begin{cases} \beta_j + min(z_{0j}, z_{ij} - \alpha_i) & \text{if } i \neq j \\ 0 & \text{otherwise.} \end{cases}$$

$$d''_{ji} = \begin{cases} \beta_j + min(z''_{0j}, z''_{ij} - \alpha_i) & \text{if } i \neq j \\ 0 & \text{otherwise} \end{cases}$$

Z'' is the canonical form of the zone B defined by: $\forall x_i, x_j \in X$,

$$b_{ij} = \begin{cases} \infty & \text{if } \beta_j = \infty \\ \infty & \text{if } z_{ij} - \alpha_i \geq z_{0j} \\ z_{ij} & \text{otherwise} \end{cases}$$

Note that $Z \subseteq Z''$ and then $\forall x_i, x_j \in X,\ z_{ij} \leq z''_{ij} \leq b_{ij}$. Consequently, if $b_{ij} = z_{ij}$, for some i and j, then $z''_{ij} = z_{ij}$.

Let us now show that $\forall x_i, x_j \in X, d_{ji} = d''_{ji}$:

- In case $(i = j)$: we have $d_{ji} = d''_{ji} = 0$.
- In case $i \neq j$:
  1 If $\beta_j = \infty$ then:
  $d_{ji} = \beta_j + min(z_{0j}, z_{ij} - \alpha_i) = \infty = \beta_j + min(z''_{0j}, z''_{ij} - \alpha_i) = d''_{ji}$
  2 If $\beta_j < \infty$ and $z_{ij} - \alpha_i \geq z_{0j}$, we have: $z''_{ij} \geq z_{ij}$, $b_{0j} = z_{0j} = z''_{0j}$. It follows that:
  $d_{ji} = \beta_j + min(z_{0j}, z_{ij} - \alpha_i) = \beta_j + z_{0j} = \beta_j + z''_{0j} = \beta_j + min(z''_{0j}, z''_{ij} - \alpha_i) = d''_{ji}$
  3 If $\beta_j < \infty$ and $z_{ij} - \alpha_i < z_{0j}$, we have $b_{ij} = z_{ij} = z''_{ij}$ and $b_{0j} = z_{0j} = z''_{0j}$.
  Then: $d''_{ji} = \beta_j + min(z''_{0j}, z''_{ij} - \alpha_i) = \beta_j + min(z_{0j}, z_{ij} - \alpha_i) = d_{ji}$.

Consequently, $D = D''$

■

Theorem 4: Let $\mathcal{N}$ be a TPN and $\mathcal{S}_\mathcal{N}$ its semantics. The zone-based abstraction (computed with k_x-$approximation$ or k'_x-$approximation$) of the state space of $\mathcal{N}$ preserves reachable markings and traces of $\mathcal{S}_\mathcal{N}$.

Proof: On a TPN $\mathcal{N}$, the zone-based forward algorithm (without k-approximation)[4] computes exactly all states and traces of the semantics $\mathcal{S}_\mathcal{N}$ [25]. Furthermore the lemma 2 proves that a symbolic state and its k_x-$approx$ and k'_x-$approx$ approximations have the same firing domain. In [10] and [14], authors have shown that all state classes sharing the same marking and firing domain share also the same traces. Therefore, our abstraction (with k_x-$approximation$ or k'_x-$approximation$) is exact: These approximations add states but preserve reachable markings and traces. ■

G. Comparisons of the three approximations

In table I, we compare the number of symbolic states (*i.e.* the number of zones) needed to compute the state space of TPN preserving reachable markings (i.e. an abstraction by inclusion and reverse inclusion) when we use the k_x-$approximation$, the LU^*_x-$Approx$ improvement of [6] and our k'_x-$approximation$ improvements. These three approximations have been implemented in the tool ROMEO. We use the classical level crossing for n trains proposed in [13] and presented in section VII.

The gain of our approximation in both time and size is significant and grows with the size of the model. It has also to be noted that, for one example (train6.xml, *i.e.* the classical gate controller problem with 6 trains) it made the computation feasible while other abstraction ran out of memory.

H. Comparisons with other methods

We have implemented and tested approximations proposed here and other abstractions developed in the literature: $SCGs$ [9] (linear state class graphs), $SSCG$ [13](linear strong state class graphs) and $ASCG$ [13] (atomic state class graphs), ZBG_k [25]. Table II reports the results obtained for the classical level crossing model with number of trains equal to 2, 3 or 4 trains, and for some producer/consumer model with number of producer/consumer equal 3, 4, 5, 6 or 7. The figures represent the size of the graph computed in term of states and edges, and the time needed to compute it.

[4]with no guaranty of termination

TPN	Nb. of symbolic states		
	k_x-$approx$	LU_x^*-$Approx$	k'_x-$approx$
train2	76	70	70
train3	383	296	296
train4	2 202	1518	1286
train5	12 068	6573	5748
train6	$\times$	28530	26 280
traffic-lights	3 856	3 014	2872

TABLE I

COMPARISON BETWEEN THE DIFFERENT APPROXIMATIONS COMPUTED ON A PIV 3GHZ, 1GB RAM

TPN	$ASCG$	$SSCG$	SCG	Our	TPN	$ASCG$	$SSCG$	SCG	Our
Train2 [1]	192	141	123	114	PrCs3	79 253	24 784	8 961	6 392
Arcs [2]	844	254	218	200	Arcs	1 200 714	115 458	41 159	29 092
CPU	$<$ 1s	$<$ 1s	$<$ 1 s	$<$ 1s	CPU	8 min 10s	7s	1s	$<$ 1s
Train3	6 967	5 051	3 101	2 817	PrCs4	130 161	40 507	14 852	10 670
Arcs	49 826	13 019	7 754	6 944	Arcs	2 476 886	232 586	84 478	60 814
CPU	4.6s	$<$ 1s	$<$ 1s	$<$ 1s	CPU	21 min 23s	23s	3s	2s
Train4	356 952	351 271	134 501	122 290	PrCs5	191 297	60 785	22 577	16 378
Arcs	3 448 100	1 193 376	436 896	391 244	Arcs	4 294 211	408 116	150 923	110 748
CPU	6 min 51s	53s	16s	56 s	CPU	43 min 46s	56s	5s	3.6s
PrCs6	262 673	86 303	32 456	23 761	PrCs7	346 304	117 792	44 837	33 085
Arcs	6 734 676	658 918	247 142	184 482	Arcs	9 915 3425	1 002 194	380 725	288 421
CPU	1h 24min	2 min	9.4s	6.2s	CPU	3h 29min	3 min 51s	16s	9s

TABLE II

SIZE COMPARISONS OF GRAPHS COMPUTED BY ABSTRACTIONS PRESERVING LINEAR OR BRANCHING PROPERTIES

[a] Number of symbolic states of the graph computed
[b] Number of edges of the graph computed

All tests have shown that our approximation is very appropriate since it is exact and results in a much smaller graphs.

Note that tables I and II are based on two different criteria of comparison of zones. In table I, we report results obtained, for different approximations, when we use inclusion as criterion of comparison (abstractions by inclusion). Such abstractions are known to preserve markings but do not necessarily preserve firing sequences. Table II reports results obtained, when we use equality as criterion of comparison (exact abstractions). Such abstractions preserve markings and firing sequences.

VI. ON-THE-FLY *TPN-TCTL* MODEL-CHECKING

An on-the-fly model-checking algorithm generates the state space of the investigated system and verifies the property of interest simultaneously. In contrast to the approach of generating the whole state space before verification, this allows the verification to be stopped as soon as the truth-hood of the property

The efficiency of such on-the-fly methods were proven for TA with tools like UPPAAL [30] (on TA) for instance. These tools restrict the set of TCTL properties that can be verified in order to provide efficient on-the-fly algorithms and faster convergence criteria. Though the set of properties is reduced, practical case studies show they were sufficient for most of the models studied.

We can now use our abstraction to efficiently perform on-the-fly model-checking of Time Petri Nets. In this section, we propose a subset of *TPN-TCTL* for which we give algorithms to decide their truth value for a bounded TPN.

A. *TPN-TCTL$_\mathcal{S}$: a subset of TPN-TCTL for on-the-fly model-checking*

We consider the subset of *TPN-TCTL* where formulas are not imbricated. However, in practice the most interesting property for real-time systems is the bounded response (liveness) property which is an imbricated formula.

The response (liveness) property is defined by : $\varphi \rightsquigarrow_I \psi$ corresponding to $\forall\Box(\varphi \Rightarrow \forall\Diamond_I\psi)$. For example, the time-bounded leads-to operator $\varphi \rightsquigarrow_{[0,c]} \psi$ expresses that whenever the state property φ holds then the state property ψ must hold within at most c time-units thereafter.

Thus we add it to our subset of *TPN-TCTL*.

Definition 18 (TPN-TCTL$_\mathcal{S}$): *TPN-TCTL$_\mathcal{S}$* is the sub-class of the temporal logic *TPN-TCTL* defined by:

$$\textit{TPN-TCTL}_\mathcal{S} ::= \exists\varphi\,\mathcal{U}_I\psi \,|\, \forall\varphi\,\mathcal{U}_I\psi \,|\, \exists\Diamond_I\varphi \,|\, \forall\Diamond_I\varphi \,|\, \exists\Box_I\varphi \,|\, \forall\Box_I\varphi \,|\, \varphi \rightsquigarrow_{I_l} \psi$$

where $\varphi, \psi \in$ *GMEC*, $(\varphi \rightsquigarrow_{I_l} \psi) = \forall\Box(\varphi \Rightarrow \forall\Diamond_{I_l}\psi)$, $I \in \{[a,b],[a,b),(a,b],(a,b)\}$ with $a \in \mathbb{N}$ and $b \in \mathbb{N} \cup \{\infty\}$, and $I_l \in \{[0,c],[0,c)\}$ with $c \in \mathbb{N}$.

B. On-the-fly TPN-TCTL$_\mathcal{S}$ model-checking

TPN-TCTL$_\mathcal{S}$ is adapted to "on-the-fly" model-checking, that is computing the state space while testing the truth value of the *TPN-TCTL$_\mathcal{S}$* formula. Compared to other classical methods based on fix-point computation (TA), it allows to only explore the relevant part of the state space relatively to the property being checked. Besides, the practical use of fix-point algorithms on TPN is discussable since, at the contrary to TA, the discrete structure of the state space (*i.e.* the set of reachable markings) is not known a priori.

Proposition 2: (Sketch of TPN-TCTL$_\mathcal{S}$ Model-checking Algorithm)

- k_x-$approximation$
  Since we choose to add a time interval to *TPN-TCTL$_\mathcal{S}$* operators, we need an additional clock to test if a run of the TPN verifies the timing constraint of the formula being model-checked. Classically, we add to the model an additional clock to be able to check the timing constraint. It counts the time length of firing sequences of the TPN. Adding this clock is strictly equivalent to extend the TPN model with a transition ($[0,\infty[$ as firing interval) that is never fired. Since it is an unbounded clock, we need to choose a constant k_c to perform the k_x-$approximation$. Its value is chosen as follows:
  - If the *TPN-TCTL$_\mathcal{S}$* formula has a bounded interval and c_{max} is its upper bound, we choose $k_c = c_{max}$. This ensures that we will compute the exact time of TPN firing sequences.
  - If the *TPN-TCTL$_\mathcal{S}$* formula has an unbounded interval and C_{min} is its lower bound, we choose $k_c = c_{min}$. This allows to detect runs whose time length are greater than c_{min}.
- k'_x-$approximation$
  In the case of the k'_x-$approximation$, the transition added has to be specially handled while approximating. Actually, the first step of the approximation, that is to replace z_{0j} with 0 if $\beta_j = \infty$, will erase the minimal absolute time at which a marking is reachable. To avoid the problem, this clock has to be approximated in the usual way with the k_x-$approximation$.

Theorem 5: The proposition 2 is exact for *TPN-TCTL$_\mathcal{S}$* model-checking.

Proof: Thanks to lemma 2, a zone and its approximations (k_x-approximation and k_x'-approximation) map to the same firing domain and then have same traces. The proof of this lemma 2 can be easily extended for any approximation values greater than the values selected for the k_x'-approximation. To verify *TPN-TCTL$_\mathcal{S}$* we add an additional clock for which we apply the particular k_x-$approximation$ of the proposition 2. Consequently, the ZBG of the extended TPN preserves markings and traces. Since the added clock is used to measure the time lengths of some traces, it remains to show that these times are exact. In [16], authors have shown that the time lengths of the TPN traces, including those of its states, can be computed using markings and firing domains. It follows that the approximations proposed here do not affect time lengths of the traces. ■

Consequently, we can build on-the-fly the state space of the extended TPN with the forward based algorithm presented in the previous section. Note that to deal with zeno TPN, we record the current explored path in order to detect zeno runs. In order to verify *TPN-TCTL$_\mathcal{S}$* properties, we introduced three new algorithms (see *Appendix*) that will handle the verification of: $\exists\varphi U_I\psi$, $\exists\Box_I\varphi$ and $\forall\varphi U_I\psi$. The other operators can be deduced from them.

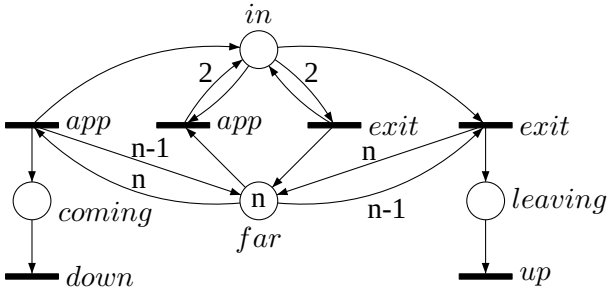

Fig. 7. Gate Controller model - Level crossing example

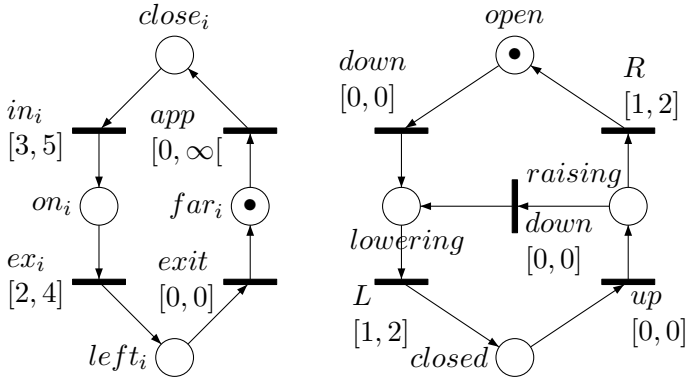

Fig. 8. Train and Gate models - Level crossing example

VII. Experimental results

A. Implementation

A prototype of the on-the-fly model-checker was implemented in the tool ROMEO [24], a tool for edition and analysis of TPN. It currently allows to model-check *TPN-TCTL*$_{\mathcal{S}}$ properties on a bounded TPN. According to the formula to check, a trace (counter-example or witness) may be generated for analysis purpose. The three approximations (k_x-$approx$, LU_x^*-$Approx$ and k'_x-$approx$) were also implemented to compare their efficiency.

B. Example

Let us consider the classical level crossing example. A TPN version of this example is proposed in [13]. The net is obtained by the parallel composition of n train models (figure 8) synchronized with the gate controller model (app, $exit$, n instantiated) (figure 7) and a gate model ($down$, up) (figure 8).

C. Properties

Nonquantitative timed properties like "when no train approches, the gate eventually opens" can be checked on the Atomic State Classes Graph [13] computed with tool TINA [12]. We first compared our computation with that of [13] in terms of number of state classes and edges of the computed graph. Figures for TINA are the size of the graph preserving CTL properties, the ones for ROMEO are the size of the graph computed on-the-fly to decide the truth-hood of the property:

$$far \rightsquigarrow \neg far \vee open$$

where far is the GMEC : $M(far) = n$. The results are reported in table III. They show the efficiency of the on-the-fly method compared to the method proposed in [13] that consists in computing the whole graph preserving CTL properties and then applying a model-checker.

Besides CTL properties, the main interest of the method is that we are now able to verify quantitative properties over TPN.

For instance, the following properties can be proved true:

TPN		Graph preserving CTL (computed with TINA)	On the fly CTL model-checking (computed with ROMEO)
3 trains	States[a]	6 967	519
	Edges or Effective States [b]	49 826	1 817
4 trains	States	356 952	8 887
	Edges or Effective States	3 448 100	36 619
5 trains	States	× [c]	185 304
	Edges or Effective States	×	957 119

TABLE III

GRAPH SIZES TO VERIFY A CTL FORMULA OVER THE GATE CONTROLLER MODEL

[a]For TINA, the figures correspond to the number of stong atomic classes computed. For ROMEO, this is the number of symbolic states of the graph computed to verify the property.

[b]For TINA, the number of transitions is the number of transitions of the graph preserving CTL properties. For ROMEO, it represents the number of symbolic states effectively computed by the algorithm to decide the truth-hood of the *TPN-TCTL*$_{\mathcal{S}}$ formula.

[c]Computation aborted due to lack of memory.

- Whenever a train crosses the gate, the gate is closed:

$$\phi_1 = \forall\Box_{[0,\infty[}\left(\left(\vee_{i\in[1,n]} on_i\right) \Rightarrow closed\right)$$

- A train can approach whereas the gate is raising.

$$\phi_2 = \exists\Diamond_{[0,\infty[}(\left(\vee_{i\in[1,n]} close_i\right) \wedge raising)$$

- In the first 1000 time units, the gate is not closed until a train approaches

$$\phi_3 = \forall\neg closed\, \mathcal{U}_{[0,1000]}\ coming$$

- If a train approaches, the gate is closed in less than 2 time units.

$$\phi_4 = coming \rightsquigarrow_{[0,2]} closed$$

We give in the table IV some computation times and the number of symbolic states computed for the model-checking of $\phi_{1,2,3,4}$. Experiments were led on a Pentium IV 3GHz with 1Gb of memory. It has to be noted that for these properties the whole state-space has to be generated to decide the truth value of the formula. It shows how efficient is the k'_x-$approx$ for the practical verification of *TPN-TCTL*$_{\mathcal{S}}$ formulas in comparison to the classical k_x-$approximation$.

We also compared our tool with UPPAAL. The gate model provided in UPPAAL distribution corresponds to n trains and 1 track with stop of train and a queue to restart the train. Though the models are not the same, we first use a modified version of the TA model for 7 trains and 7 tracks. We obtain the following results: on gate model (7 trains and 7 tracks), a computation time of 85s for ϕ_2 with UPPAAL against 17s with ROMEO. On the contrary, if we use the UPPAAL gate model (n trains and 1 track with stop of train and a queue to restart the train), we obtain the following results (for 7 trains) a computation time of 12.42s for the property $Train(i).Appr \rightsquigarrow Train(i).Cross$ with UPPAAL against 127.25s with ROMEO for the comparable property $close_i \rightsquigarrow left_i$ on the TPN model. It comes from the fact that the queue is done by an interpreted algorithm with UPPAAL whereas we model it by a TPN (with all the possible combinations) leading to a state space explosion.

These comparisons are not very significative as models are different but it shows the efficiency of our method and tool. Moreover, some implementation improvements (such as the use of Clock Difference Diagrams) are not yet integrated in ROMEO. Finally, the use of subscript TCTL in ROMEO eases the expression of quantitative properties: for instance the TA model must be modified (with an observer) to model-check $Train(i).Appr \rightsquigarrow_{[0,2]} Train(i).Cross$ with UPPAAL while it is not necessary for ROMEO.

		ϕ_1		ϕ_2	
		k_x	k'_x	k	k'_x
3 trains	CPU time	< 1 s	< 1 s	< 1 s	< 1 s
	Nb. Zones	3803	1 310	43	36
4 trains	CPU time	2.44 s	< 1 s	< 1 s	< 1 s
	Nb. Zones	60550	29 020	117	82
5 trains	CPU time	$\times$[a]	2 min 24s	< 1 s	< 1 s
	Nb. Zones	$\times$	753 119	1242	645

		ϕ_3		ϕ_4	
		k_x	k'_x	k_x	k'_x
3 trains	CPU time	< 1 s	< 1 s	< 1 s	< 1 s
	Nb. Zones	58	58	2 407	1 828
4 trains	CPU time	< 1 s	<1 s	3.46 s	< 1 s
	Nb. Zones	226	226	84 659	38 761
5 trains	CPU time	< 1s	< 1s	$\times$	2 min 37 s
	Nb. Zones	1046	1046	$\times$	958 537

TABLE IV

COMPUTATION TIME TO MODEL-CHECK *TPN-TCTL*$_{\mathcal{S}}$ FORMULAS.

[a]Computation aborted - Ran out of memory

VIII. CONCLUSION

We have considered TCTL model-checking of Time Petri Nets. From a theoretical point of view, we have proved using region graphs the decidability of the model-checking of *TPN-TCTL* on bounded TPN as well as its PSPACE complexity. However, in practice, region graphs are not useful as they run up against the state explosion problem. To overcome this limitation, we have considered a subset of *TPN-TCTL* properties (*TPN-TCTL*$_{\mathcal{S}}$) for which we have proposed efficient on-the-fly forward model-checking algorithms using zone based abstractions. As for Timed Automata, an over-approximation operator on zones is used to enforce the termination of the algorithms. In order to speed up the convergence of these algorithms, we have defined a finer approximation operator over zones which leads to a much compact abstraction. We have shown that our abstraction is exact and allows the effective verification of *TPN-TCTL*$_{\mathcal{S}}$ properties. The method was implemented and integrated in the tool ROMEO [24]. It is then possible to check efficiently real-time properties with this tool, and to benefit from all the advantages of it: counter example generation when a property is false, simulation environment.

From a theoretical point of view, the complexity of model-checking should be explored for general TPN or subclasses of TPN as well as combining reduction methods (structural or partial order methods) to decrease the practical cost of the model-checking of TPN.

ACKNOWLEDGMENTS :

The authors are grateful to François Laroussinie, Didier Lime and Agata Pólrola for their comments on the original draft paper.

REFERENCES

[1] R. Alur, C. Courcoubetis, and D. Dill. Model-checking in dense real-time. *Information and Computation*, 104(1):2–34, 1993.

[2] R. Alur, C. Courcoubetis, and D. L. Dill. Model-checking for real-time systems. In *5th IEEE Symposium on Logic in Computer Science*, pages 414–425, Philadelphia, Pennsylvania, June 1990. IEEE Computer Society Press.

[3] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[4] G. Behrmann, P. Bouyer, E. Fleury, and K. G. Larsen. Static guard analysis in timed automata verification. In *Proceedings of the 9th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS'03)*, volume 2619 of *Lecture Notes in Computer Science*, pages 254–277. Springer, 2003.

[5] G. Behrmann, P. Bouyer, K. G. Larsen, and R. Pelánek. Lower and upper bounds in zone based abstractions of timed automata. In *Proceedings of the 10th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS'04)*, volume 2988 of *Lecture Notes in Computer Science*, pages 312–326. Springer, 2004.

[6] G. Behrmann, P. Bouyer, K. G. Larsen, and R. Pelánek. Lower and upper bounds in zone-based abstractions of timed automata. *International Journal on Software Tools for Technology Transfer*, 8(3):204–215, June 2006.

[7] Johan Bengtsson and Wang Yi. Timed automata: Semantics, algorithms and tools. In W. Reisig and G. Rozenberg, editors, *In Lecture Notes on Concurrency and Petri Nets*, Lecture Notes in Computer Science vol 3098. Springer–Verlag, 2004.

[8] B. Bérard, F. Cassez, S. Haddad, D. Lime, and O. (H.) Roux. Comparison of the expressiveness of timed automata and time Petri nets. In *3rd International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS 05)*, volume 3829 of *Lecture Notes in Computer Science*. Springer, September 2005.

[9] B. Berthomieu. La méthode des classes d'états pour l'analyse des réseaux temporels. In *3e congrès Modélisation des Systèmes Réactifs (MSR'2001)*, pages 275–290, Toulouse, France, October 2001. Hermes.

[10] B. Berthomieu and M. Diaz. Modeling and verification of time dependent systems using time Petri nets. *IEEE trans. on soft. eng.*, 17(3):259–273, 1991.

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

[12] B. Berthomieu and F. Vernadat. TINA. http://www.laas.fr/tina.

[13] B. Berthomieu and F. Vernadat. State class constructions for branching analysis of time Petri nets. In *TACAS 2003*, volume 2619 of *Lecture Notes in Computer Science*, pages 442–457, 2003.

[14] H. Boucheneb and G. Berthelot. Toward a simplified building of time Petri nets reachability graph. In *PNPM'93*, pages 46–55, Toulouse, France, 1993.

[15] H. Boucheneb and R. Hadjidj. Towards optimal CTL* model checking of time Petri nets. In *WODES'04*, REIMS (France), June 2004.

[16] H. Boucheneb and J. Mullins. Analyse des réseaux de Petri temporels: Calcul des classes en $o(n)$ et des temps de chemin en $o(m \times n)$. *Technique et Science Informatiques*, 22/4:435–459, 2003.

[17] P. Bouyer. Timed automata may cause some troubles. Technical report, LSV, July 2002.

[18] P. Bouyer. Unteamable timed automata! In *Proc. 20th Annual Symposium on Theoretical Aspects of Computer Science (STACS'2003)*, volume 2607 of *lncs*, pages 620–631, Berlin, Germany, February 2003. Springer Verlag.

[19] F. Cassez and O. (H.) Roux. Structural translation from time Petri nets to timed automata. In *AVoCS'04*, London (UK), September 2004.

[20] F. Cassez and O. (H.) Roux. Structural translation from Time Petri Nets to Timed Automata – Model-Checking Time Petri Nets via Timed Automata. *The journal of Systems and Software*, 79(10):1456–1468, 2006.

[21] A. Cheng, J. Esparza, and J. Palsberg. Complexity results for 1-safe nets. *Theoretical Computer Science*, 147:117–136, 1995.

[22] C. Daws and S. Tripakis. Model checking of real-time reachability properties using abstractions. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'98)*, volume 1384, 1998.

[23] D. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Proc. of Workshop on Computer Aided Verification Methods for Finite State Systems*, volume 407, pages 197–212, 1989.

[24] G. Gardey, D. Lime, M. Magnin, and O. (H.) Roux. Romeo: a tool for analyzing time petri nets. In *Proc. 17th International Conference on Computer Aided Verfication (CAV'05)*, volume 3576 of *Lecture Notes in Computer Science*, pages 418–423, Edinburgh, Scotland, UK, July 2005. Springer–Verlag. http://romeo.rts-software.org.

[25] G. Gardey, O. (H.) Roux, and O. (F.) Roux. Using zone graph method for computing the state space of a time Petri net. In *FORMATS'03*, volume 2791 of *lncs*, 2003.

[26] G. Gardey, O. (H.) Roux, and O. (F.) Roux. State space computation and analysis of time Petri nets. *Theory and Practice of Logic Programming (TPLP). Special Issue on Specification Analysis and Verification of Reactive Systems*, 6(3):301–320, 2006.

[27] A Giua, F. DiCesare, and M Silva. Generalized mutual exclusion constraints on nets with uncontrollable transitions. In *IEEE Int. Conf. on SMC*, 1992.

[28] R. Hadjidj and H. Boucheneb. On-the-fly tctl model checking of time Petri net using state class graphs. In *Proc. of the sixth international conference on Application of Concurrency to System Design, ACSD 2006*. IEEE Comp. Soc. Press., 2006.

[29] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. A user guide to HYTECH. In *Proc. 10th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'95)*, volume 1019 of *Lecture Notes in Computer Science*, pages 41–71, 1995.

[30] K. G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1–2):134–152, October 1997. http://www.uppaal.com/.

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

[32] D. Lime and O. (H.) Roux. Model checking of time Petri nets using the state class timed automaton. *Journal of Discrete Events Dynamic Systems - Theory and Applications (DEDS)*, 16(2):179–205, 2006.

[33] P. M. Merlin. *A study of the recoverability of computing systems*. PhD thesis, Dep. of Information and Computer Science, Univ. of California, Irvine, CA, 1974.

[34] Y. Okawa and T. Yoneda. Symbolic CTL model checking of Time Petri nets. In Scripta Technica, editor, *Electronics and Communications in Japan*, volume 80, pages 11–20, 1997.

[35] W. Penczek and A. Pólrola. Specification and model checking of temporal properties in time petri nets and timed automata. In *Proceedings of Applications and Theory of Petri Nets 2004: 25th International Conference, ICATPN 2004*, volume 3099 of *Lecture Notes in Computer Scienc*, pages 37–76. Springer–Verlag, sep 2004.

[36] L. Popova. On time petri nets. *Journal Information Processing and Cybernetics, EIK*, 27(4):227–244, 1991.

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

[38] S. Tripakis and S. Yovine. Analysis of timed systems using time-abstracting bisimulations. *Formal Methods in System Design*, 18(1):25–68, jan 2001.

[39] I. Virbitskaite and E. Pokozy. A partial order method for the verification of time petri nets. In *Fundamentals of Computation Theory: 12th International Symposium, FCT'99*, volume 1684 of *Lecture Notes in Computer Science*, pages 547–558. Springer-Verlag, 1999.

[40] T. Yoneda and H. Ryuba. CTL model checking of time Petri nets using geometric regions. *IEICE Transactions on Information and Systems*, E99-D(3):297–396, March 1998.

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

APPENDIX

This appendix details the model-checking algorithms for *TPN-TCTL*$_{\mathcal{S}}$. Versions presented are depth-first-search algorithms, however non-recursive implementations are also available in ROMEO.

A. Model-checking algorithms for TPN-TCTL$_{\mathcal{S}}$

1) Model Checking $\exists\varphi U_I\psi$*:* The aim of the algorithm is to look for a path $(s_0, v_0) \to \cdots \to (s_n, v_n)$ such that for all states $(s_i, v_i) \models \varphi$ and $(s_n, v_n) \models \psi$ in the time constraint window I. Finding such a path is possible by enumerating the reachable states of the TPN and looking for a state verifying ψ in the time interval I.

Algorithm 2 $CheckEU(s, \varphi, \psi, I)$

$Visited \leftarrow \emptyset$
$s_0 = \overrightarrow{Post}(M_0, \bar{0})$
return $checkEU_{aux}(s, \varphi, \psi, I)$

Algorithm 3 $CheckEU_{aux}(s, \varphi, \psi, I)$

$(M, Z) \leftarrow s$
if $M \models \psi \wedge Z \cap I \neq \emptyset$ **then**
  // First stopping criterion:
  // The property ψ is true for some valuations in I.
  return True
else if $M \not\models \varphi$ **then**
  // Second stopping criterion:
  // The property φ doesn't hold.
  // Don't investigate further for this state.
  return False
else
  $Z' \leftarrow Z \cap [0, I_{max})$
  if $Z' = \emptyset$ **then**
    // Third stopping criterion:
    // There is no time left for the property to hold.
    return False
  else
    // φ is true and there is time left for $E\varphi U_I\psi$ to hold.
    // Look in successors to find an accepting run
    $Visited \leftarrow Visited \cup \{s\}$
    for all $t \in firable(M, Z')$ **do**
      $s' = \overrightarrow{Post}_t(M, Z')$
      if $s' \not\subseteq Visited$ **then**
        if $checkEU_{aux}(s', \varphi, \psi, I)$ **then**
          return True
        end if
      end if
    end for
    // No successors verify the property
    return False
  end if
end if

The algorithm converges by inclusion: if we reach a state $s = (M, Z) \subseteq Visisted$,*i.e.* there exists $(M, Z') \in Visited$ such that $Z \subset Z'$, the successors of s are not computed since they will generate runs that are already

covered by Z'. Consequently, if the property is false for (M, Z') then it is also false for (M, Z); reciprocally, if the property is true for (M, Z) then it is true for (M, Z').

2) Model Checking $\exists\Diamond_I\varphi$: By definition $\exists\Diamond_I\varphi == \exists TrueU_I\varphi$.

3) Model Checking $\forall\Box_I\varphi$: By definition $\forall\Box_I\varphi = \neg\exists\Diamond_I\neg\varphi$.

4) Model Checking $\exists\Box_I\varphi$: By definition $\exists\Box_I\varphi = \neg\forall\Diamond_I\neg\varphi$. Since the model-checking of $\forall\Diamond_I$ needs to converge by equality, and that φ is a state formula in our subset of *TPN-TCTL$_S$*, we use a specific algorithm to check $\exists\Box_I$. It uses a convergence by inclusion, which ensures a faster computation time if the property holds.

Algorithm 4 $CheckEG(s, \varphi, \psi, I)$

```
Visited ← ∅
s_0 = Post(M_0, 0̄)
return checkEG_aux(s, φ, ψ, I)
```

Algorithm 5 $CheckEG_{aux}(s, \varphi, I)$

```
(M, Z) ← s
// First stopping criterion:
//     φ doesn't hold.
if M ⊭ φ then
   return False
else
   Visited ← Visisted ∪ {s}
   // Second stopping criterion:
   //     φ is true for all valuations
   if I ⊆ Z ∩ I then
      return True
   else
      // Iterate through successors
      for all t ∈ firable(s') do
         s' = Post_t(s)
         if s' ⊈ Visited then
            if checkEG_aux(s', φ, I) then
               return True
            end if
         end if
      end for
      // no successors verify the property
      return False
   end if
end if
```

5) Model Checking $\forall\Diamond_I\varphi$: By definition $\forall\Diamond_I\varphi = \forall TrueU_I\varphi = \neg\exists\Box_I\neg\varphi$. So we can use the algorithm $checkEG(s, \neg\varphi, I)$ to decide the truth value of the property.

6) Model Checking $\varphi \rightsquigarrow_{\leq c} \psi$: We consider here the bounded response. It consists in verifying that every time φ becomes true, ψ always holds in less than c time units. Using an additional boolean property b that stores the requirement for ψ to hold within c time units, the property can be transformed into $\forall\Box_{[0,\infty[}(b \Rightarrow z \leq t)$ (see PhD thesis of P. Petterson). This reduced to check $\exists\Diamond_{[0,\infty[}(\neg b \wedge z > t)$ or also $\exists\Diamond_{[0,\infty[}(b \wedge z > t)$. This result in a slightly modified $\exists\Diamond_I$ algorithm, that is: (1) we add an additional clock z to store the time requirement for ψ; (2) each time a state for which φ starts to hold is encountered, b is set to true and z is reset; (3) each time ψ is true the value of b is set to false. The algorithm stops if it encounters a state such that b is true and $z > t$.

Algorithm 6 $CheckBoundedResponse(s_0, \varphi, \psi, c)$

$Visited \leftarrow \emptyset$
$s_0 = (M_0, \bar{0})$
return $\neg checkBoundedResponse\_aux(s_0, \varphi, \psi, c)$

Algorithm 7 $CheckBoundedResponse\_aux(s, \varphi, \psi, c)$

$(M, Z, b) \leftarrow s$
if $Z \cap (b, \infty) \neq \emptyset$ **then**
  // Stopping criterion:
  // ψ doesn't hold within the timing constraint $z \leq t$
  return True
end if
if $s \models \psi$ **then**
  // ψ is true, stop the need to verify ψ for successors
  $b \leftarrow false$
else if $\neg b \wedge s \models \varphi$ **then**
  // φ starts to hold
  $b \leftarrow true$
  $Z \leftarrow Z[z \leftarrow 0]$
end if
// Compute time successors
$Z \leftarrow \overrightarrow{Post}(Z)$
$Visited \leftarrow Visited \cup \{(M, Z, b)\}$
for all $t \in$ *firable*(s) **do**
  $s' \leftarrow Post_t(M, Z)$
  if $s \not\subseteq Visited$ **then**
    if $checkBoundedResponse\_aux(s', \varphi, \psi, c)$ **then**
      return True
    end if
  end if
end for
return False

7) Model Checking $\forall \varphi U_I \psi$: Let (M, Z) a symbolic state for which we want to check $\forall \varphi U_I \psi$. Informally, we can decide its truth value for this state using the following steps:

- If there are some valuations of z greater than I_{max} then the property is false.
- If $M \models \psi$ and all the valuations for the global clock z of Z are in I then the property is true for this state.
- If $M \not\models \varphi$ (ie $\varphi \vee \psi$) then the property is false.
- We compute the time successors of (M, Z).
- We remove all the states such that ψ is true and $z \in I$ and note (M, Z') this new symbolic state. Actually, these states removed belong to paths that verifies the property. We only need to verify states for which ψ is true and $z < I_{min}$, that is every runs starting from them will eventually lead to a state such that $\psi \wedge z \in I$ is true.
- For each firable transition we compute the discrete successors.

Concerning the convergence criterion, let us first suppose that the TPN is non-zeno *i.e.* there are no loops in null time. If we consider convergence by inclusion, since the TPN is non-zeno, it is guaranted that the global time progresses. Consequently, if we reach a state (M, Z) such that Z is included in some Z' of a state (M, Z') already visited, we are sure this can't be a loop in null time. Then, if the property is true for (M, Z') it is also true for (M, Z) and reciprocally if the property is false for (M, Z').

The problem is then to handle zeno runs for zeno TPN. Actually if the algorithm converges by inclusion, it isn't able to detect null time loops which may invalidate the property. Thus, to detect loops, we record the prefix (*i.e.* the trace of symbolic states) of the symbolic state being checked.

Algorithm 8 $CheckAU(s, \varphi, \psi, I)$

$Visited \leftarrow \emptyset$
$Prefix \leftarrow \emptyset$
$s_0 = (M_0, \bar{0})$
return $checkAU_{aux}(s, \varphi, \psi, I)$

Algorithm 9 $CheckAU_{aux}(s, \varphi, \psi, I)$

$Z' \leftarrow Z$
$(M, Z) \leftarrow s$
$Z = Z_{(0,I_{min})} \cup Z_{(I_{min},I_{max})} \cup Z_{(I_{max},inf)}$
if $M \models \psi$ **then**
  // First stopping criterion: $\psi \wedge I$ doesn't hold for all states
  if $Z_{(I_{max},inf)} \neq \emptyset$ **then**
    return False
  end if
  // We select only states for which $\psi \wedge I$ doesn't hold.
  $Z' \leftarrow Z \cap (0, I_{min})$
  // Sopping criterion: φ doesn't hold
else if $M \not\models \varphi$ **then**
  return False
else
  // We compute time successors
  $Z' \leftarrow timeNext(Z)$
  // Iterate through successors to test if all path are valid
  $Visited \leftarrow Visited \cup \{s\}$
  for all $t \in firable(M, Z')$ **do**
    $Prefix \leftarrow Prefix \cup s$
    $s' \leftarrow discreteNext(M, Z', t)$
    // Test if it is a loop state
    if $s' \in Prefix$ **then**
      return False
    end if
    // If it is already visited and not in prefix the property still holds.
    if $s' \notin Visited$ **then**
      if not $checkAU_{aux}(s', \varphi, \psi, I)$ **then**
        return False
      end if
    end if
    $Prefix \leftarrow Prefix \setminus s$
  end for
  // All successors verify the property
  return True
end if