Model checking of time Petri nets

Hanifa Boucheneb and Rachid Hadjidj
Department of Computer Engineering
École Polytechnique de Montréal
P.O. Box 6079, Station Centre-ville
Montréal, Québec
{hanifa.boucheneb,rachid.hadjidj}@polymtl.ca

Abstract

This paper considers time Petri nets (TPN model) for model checking. The main challenge in model checking techniques is to construct, with lesser resources (time and space), a much coarser abstraction preserving properties of interest. These properties can be verified using standard model checking techniques. In this paper, we review some techniques, proposed in the literature, to model check untimed and timed properties of the TPN.

Keywords: Time Petri nets, state explosion problem, abstraction, model checking, TCTL properties.

1. INTRODUCTION

Temporal logic model checking is a very attractive and automatic verification technique of systems. It is applied by representing the behavior of a system as a finite *state transition system*, specifying properties of interest in a temporal logic (*LTL, CTL, CTL*, MITL, TCTL*) and finally exploring the state transition system to determine whether they hold or not. To use this technique with timed systems, an extra effort is required to abstract their generally infinite state spaces. The abstraction must generate a finite graph while preserving properties of interest. For best performances, the graph should also be the smallest possible and computed with minor resources too (time and space). Several abstractions of the TPN state space have been proposed in the literature which preserves different kinds of properties (reachability, LTL, MITL, CTL*, TCTL properties). The preserved properties are verified using standard model checking techniques [1, 11, 13, 15].
This paper reviews some techniques, proposed in the literature, to model check untimed and timed properties of the TPN. The rest of the paper is organized as follows. Section 2 introduces the TPN and its semantics. Section 3 is devoted to the state space abstraction. Section 4 is devoted to the model checking techniques.

2. TIME PETRI NETS

A TPN is a Petri net with time intervals attached to its transitions. Formally, a TPN is a tuple $\mathcal{N} = (P, T, Pre, Post, m_0, Is)$ where P is a finite set of places, T is a finite set of transitions, such that $P \cap T = \emptyset$, Pre and $Post$ are the backward and the forward incidence functions: $P \times T \to \mathbb{N}$[1], $m_0 : P \to \mathbb{N}$, is the initial marking, Is[2] $: T \to \mathbb{Q}^+ \times (\mathbb{Q}^+ \cup \{\infty\})$ associates with each transition t an interval called *static firing interval* of t.
Let M be the set of all markings of a TPN, $m \in M$ a marking, and $t, t_f \in T$ two transitions. t is said to be *enabled* in m, iff all tokens required for its firing are present in m, i.e.: $\forall p \in P, m(p) \geq Pre(p, t)$. We denote by $En(m)$ the set of all transitions enabled in m. If m results from firing transition t_f from another marking, $New(m, t_f)$ denotes the set of all transitions newly enabled in m, i.e.:
$New(m, t_f) = \{t \in En(m) | t = t_f \vee (\exists p \in P, m(p) - Post(p, t_f) < Pre(p, t))\}$.
There are two known characterizations of the TPN state. The first one, based on clocks, associates with each transition t of the model a *clock* to measure the time elapsed since t became enabled most recently [7, 8, 13, 18]. The TPN *clock state* is a couple (m, v), where m is a marking and v is a clock valuation function, $v : En(m) \to \mathbb{R}^+$. When a transition t becomes enabled, its clock is initialized to zero. The value of this clock increases synchronously with time until t is fired or disabled by the firing of another

[1] $\mathbb{N}$ is the set of nonnegative integers.
[2] $\mathbb{Q}^+$ is the set of nonnegative rational numbers.

transition. t can fire, if the value of its clock is inside its static firing interval $Is(t) = [tmin(t), tmax(t)]$. It must be fired immediately, without any additional delay, when the clock reaches $tmax(t)$. The second characterization, based on intervals, defines the TPN state as a marking and a function which associates with each enabled transition a firing interval [2]. The TPN interval state is a couple (m, I), where m is a marking and I is an interval function, $I : En(m) \rightarrow \mathbb{Q}^+ \times (\mathbb{Q}^+ \cup \{\infty\})$. When a transition t becomes enabled, its firing interval is set to its static firing interval $Is(t)$. The bounds of this interval decrease synchronously with time, until t is fired or disabled by another firing. t can fire, if the lower bound of its firing interval reaches 0, but must be fired, without any additional delay, if the upper bound of its firing interval reaches 0.
Despite their relatedness, the two state characterizations still show some major differences. The origin of these differences stems from the relationship between the two characterizations of states which can be stated as follows: Let (m, v) be a clock state. Its corresponding interval state is (m, I) such that: $\forall t \in En(m), I(t) = [max(0, tmin(t) - v(t)), tmax(t) - v(t)]$. Note that for any real value $u \geq tmin(t)$, if $tmax(t) = \infty$, $tmax(t) - u = \infty$ and $max(0, tmin(t) - u) = 0$. This means that for TPN models with unbounded firing intervals, infinitely many clock states may map to the same interval state. In such a case, all these states will obviously exhibit the same future behavior. The same remark extends also to state classes, where several state classes based on clocks (sometimes an infinity) may map to a single state class based on intervals. Note also that states in a clock state class can be distinguished one by one, whereas it is impossible with an interval state class. The reason is that the firing domain of an interval state class is the union of firing domains of all its states, and the union is known to be an irreversible operation. Together, the mentioned remarks suggest that the interval characterization of states has a more abstracting power than the clock characterization, and allows to construct more compact abstractions of the TPN state space, preserving linear properties. In addition, abstractions based on clocks do not enjoy naturally the finiteness property for bounded TPNs with unbounded intervals as it is the case for abstractions based on intervals. The finiteness is enforced using an approximation operation on state classes [2, 7, 10], which may involve some overhead computation. However, abstractions based on intervals are not appropriate for constructing abstractions preserving branching properties (*ASCGs*). Indeed, this construction, based on splitting state classes (a set of states), is not possible with the characterization interval of state classes.
The semantics of the TPN can be defined using either the clock state or interval state characterization. Let $\theta \in \mathbb{R}^+$ be a nonnegative reel number, t a transition of T, s and s' two states of a TPN. We write $s \xrightarrow{\theta} s'$, iff state s' is reachable from state s after a time progression of θ time units. We write $s \xrightarrow{t} s'$ iff state s' is immediately reachable from state s by firing transition t.
The *state space* of a *TPN* is defined as a structure $(\mathcal{S}, \rightarrow, s_0)$, where s_0 is the initial state of the model, and $\mathcal{S} = \{s | s_0 \xrightarrow{*} s\}$ is the set of reachable states ($\xrightarrow{*}$ is the reflexive and transitive closure of $\rightarrow$). The state space of the TPN model is generally infinite and then not suitable for enumerative analysis. Hence, further abstractions are needed.

3. TPN STATE SPACE ABSTRACTIONS

Abstraction techniques aim to construct by removing some irrelevant details, a contraction of the state space of the model, which preserves properties of interest. We can find, in the literature, several state space abstractions for the *TPN* model. These abstractions may differ mainly in the agglomeration criteria of states, the characterization of states and state classes (interval states or clock states), the kind of properties (linear or branching) they preserve and their size.
In abstractions preserving linear properties, states reachable by the same firing sequence independently of their firing times are usually agglomerated in the same node. States reachable via time progression may be either included (ZBG [7]) or abstracted (SCG [2], GRG [18], SSCG [2], CSZG [10]). The agglomerated states are then considered modulo some relation of equivalence (firing domain of the SCG [2], k-approximation of the ZBG [7], approximation of the SSCG [2], and k-normalization of the CSZG [7]). These abstractions, except the GRG, are finite for all bounded time Petri nets. The SCG is, in general, more smaller than others [2] and then more appropriate to model checking linear properties. The reason is that the SCG is an interval state abstraction while the others are clock state abstractions. In addition, we obtain coarser abstractions when we add to each state class all states reachable from it by time progression (relaxing state classes). Indeed, two different state classes may have identical relaxed state class. As an example, consider the two interval state classes of a SCG $\alpha_1 = (m, 2 \leq t \leq 3)$

TPN	$RSCG$	SCG	$CSZG$	$SSCG$	θ^{CSZG}_{RSCG}	ZBG	θ^{ZBG}_{RSCG}
P(2)	593/1922	748/2460	1773/6131	7963/42566	2.99/3.19	12280/42280	20.70/22
cpu(s)	0.02	0.04	0.06	0.51	3	0.49	24.5
P(3)	3240/15200	4604/21891	16600/82834	122191/1111887	5.12/5.45	?	-
cpu(s)	0.14	0.40	1.73	20.04	12.36		-
P(4)	9504/56038	14086/83375	68451/429014	659377/7987583	7.20/7.66	?	-
cpu(s)	0.62	1.83	12.15	183.87	19.60		-
P(5)	20877/145037	31657/217423	196707/1447269	9.42/9.98	-	?	-
cpu(s)	2.01	5.67	51.04		25.40		-

TABLE 1: Comparison of different abstractions

and $\alpha_2 = (m, 1 \leq t \leq 3)$. We have $relax(\alpha_1) = relax(\alpha_2) = (m, 0 \leq t \leq 3)$ while $\alpha_1 \neq \alpha_2$. As an example, we report in table 1 sizes and computing times of the RSCG[3], SCG, SSCG, CSZG and ZBG for some producer and consumer model. A question mark in the table indicates a situation where the computation has not completed after an hour of time, or aborted due to a lack of memory.

Abstractions preserving branching properties (CTL* properties) are built using a partition refinement technique [14] in two steps [2, 3, 4, 8, 9, 10, 13, 18]. An abstraction, which does not necessarily preserve branching properties, is first built then refined in order to restore the condition AE, i.e.: $\forall(\alpha, t, \alpha') \in\Longrightarrow$[4], $(\alpha \subseteq Pred(\alpha', t))$, where $Pred(\alpha', t)$ is the set of all states which may lead after possibly some time progression and firing transition t to some states in α'. To verify the atomicity of class α for the transition (α, t, α'), it suffices to verify that α is equal or included in $Pred(\alpha', t)$. In case α is not atomic, it is partitioned into a set of convex subclasses so as to isolate the predecessors of α' by t in α, from those which are not. This *refinement operation* is repeated until all state classes are atomic. The refinement process generates a finite graph iff the intermediate abstraction is finite [2, 3]. The intermediate abstractions used in the literature are all based on clock states but may differ in agglomeration criteria of states and state classes (equality [2, 18] , inclusion [3, 4] or convex-combination [9, 10]). The choice of the abstraction to be refined has a significant impact on the refinement process. Indeed, in [6, 10], authors showed that the refinement process is significantly improved when we use coarser abstractions. Such abstractions may be obtained using relaxation, inclusion or convex-combination as an additional agglomeration criterium of state classes. Experimental results have shown an important reduction in refinement times (more than *ten* times for some tested models) and memory usage, resulting in graphs closer in size to the optimal. Despite the simplicity of the used models, they allowed to illustrate some interesting features related to the computation pattern followed by the refinement procedure, depending on which abstraction is refined. If an inclusion or a convex-combination abstraction is used, the refinement follows a linear pattern[5]. When an abstraction preserving linear properties is refined, the size of the computed graph starts first to grow up to a *peek size* then decreases until an atomic state class space is obtained. In certain cases, the peek size grows out of control, leading to a state explosion (see *figure 1.S(5)*).

4. MODEL CHECKING TIMED PROPERTIES

To verify some timed properties, in [14], authors used observers to express them in the form of TPNs and reduce them to reachability properties. However, properties on markings are quite difficult to express with observers [14]. Other techniques define translation procedures from the TPN model into timed automata [5, 12], in order to make use of available model checking tools [11, 13, 15]. Model checking is then performed on the resulting timed automata, with results interpreted back on the original TPN model. The translation into timed automata may be either structural (each transition is translated into a timed automata using the same pattern) [5] or semantic (the state class graph of the TPN is first constructed and then translated into a timed automaton)[9]. Such translations show that CTL^*, TCTL, LTL, MITL model checking are decidable for bounded TPNs and that developed algorithms on timed automata may be extended to TPNs. Though effective, these techniques face the difficulty to interpret back and forth properties between the two models.

Virbitskaite and Pokozy in [17] proposed a method to model check TCTL properties on TPN. The method is based on the region graph method and is similar to the one proposed by Alur [1] for timed

[3]RSCG is computed like SCG but each computed state class is relaxed before comparing it with other computed state classes.
[4]Relation $\Longrightarrow$ is the transition relation of the abstraction.
[5]The size of the graph grows linearly in time during its construction

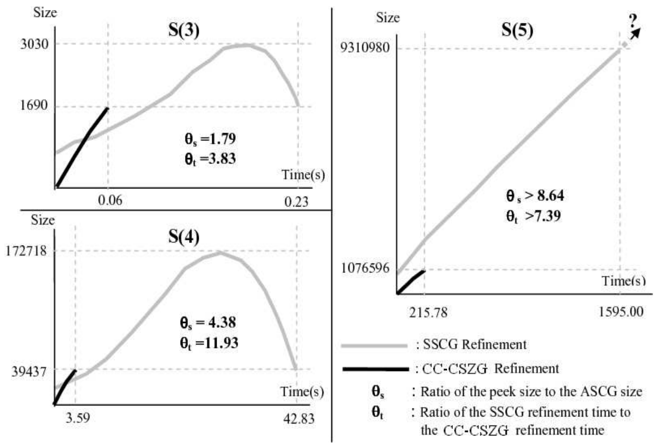
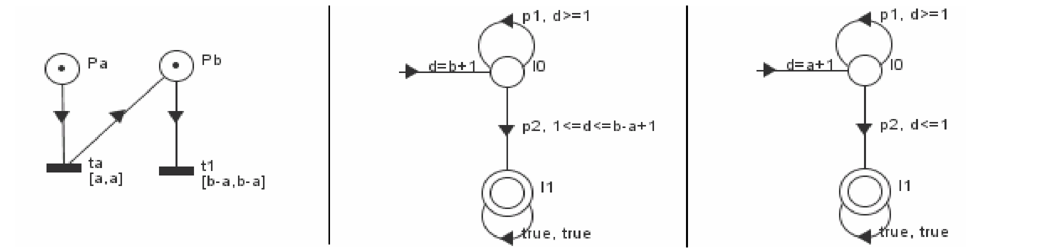


FIGURE 1: Refinement patterns of SSCGs and CC-CSZGs

automata. However, the region graph is known to be a theoretical method which is not applicable in practice because of its lack of efficiency. To achieve the same goal, it is possible to adapt to the TPN, the method proposed in [13, 15] for timed automata. The verification of a TCTL formula proceeds by adding a special transition ts[6] to the TPN, translating the TCTL formula into some CTL formula, constructing an abstraction which preserves CTL properties of the completed TPN and then applying a CTL model checking technique. The transformation of TCTL formulae into CTL ones needs to extend CTL with atomic propositions of the form $t_s \in I$, and a particular next operator X_{t_s} defined by: for each formula ψ and each state s' of the TPN, s' satisfies $X_{t_s}\psi$ iff the state resulting by firing ts satisfies ψ. For example, the formula $\phi = \forall(\phi_1 U_I \phi_2)$ is translated into the formula $\phi' = X_{t_s}(\forall(\phi'_1 U(\phi'_2 \wedge t_s \in I)))$. The verification of ϕ' is performed using the classical CTL model checking technique by constructing the quotient graph of TA' according with some strong time Bisimulation relation [13, 15]. However, this method needs to compute the whole abstraction of the model before it is analyzed. In [8], authors proposed, using the state class method (SCG), a forward on-the-fly model checking technique for a subclass of TCTL properties. The on-the-fly methods allow to stop the construction of the graph as soon as the truth value of the property is obtained. The verification proceeds by augmenting the TPN model under analysis with a special TPN, called *Alarm-clock* shown in figure 2.a, to allow the capture of relevant time events (reaching, over passing a time interval). A forward on-the-fly exploration combined with an abstraction by inclusion is then applied on the resulting TPN. In order to capture all firings which will occur inside interval $[a, b]$, transition t_a is fired before any other transition which can be fired at exactly the same time, whereas, transition t_1 is fired after any other transition which can be fired at exactly the same time. To model check timed linear properties, one can apply the technique proposed



a) The Alarm-clock TPN

b) A delay based Buchi automaton *of (p1 $U_{[a,b]}$ p2),* for case b≠∞

c) A delay based Buchi automaton *of (p1 $U_{[a,b]}$ p2),* for case b=∞

FIGURE 2: The Alarm-clock TPN

in [15, 16] for timed automata. This technique is based on timed Buchi automata and consists in two steps. The first step constructs some timed *Buchi* automaton for the negation of the property to be checked. The second step computes the synchronous product of a clock based abstraction preserving

[6]This transition is used to deal with time constraints of the property to be verified. Its firing interval is $[0, \infty]$.

linear properties (ZBG, SSCG, CSZG) of the TPN with the timed *Buchi* automaton. This construction can be performed using an on-the-fly method combined with an abstraction by inclusion. The property is satisfied iff the synchronous product is empty. The timed Buchi automaton is a Buchi automaton extended with clocks and time constraints attached to locations and edges. Knowing that the interval based abstractions (SCG and RSCG) are in general much more smaller than the clock based abstractions, it will be interesting to adapt this technique to abstractions based on delays. In this perspective, we can extend the Buchi automata with delays and time constraints over delays to express timed linear properties and verify them on the SCG or the RSCG. As an example, figures 2.b and 2.c show the delay based timed Buchi automata of the MITL formula $p_1 U_{[a,b]} p_2$[7].

5. CONCLUSION

In this paper, we presented and discussed model checking techniques of time Petri nets. We pointed out some strategies which allow to make model checking techniques more efficient. Indeed, for CTL* and TCTL properties, clock based abstractions contracted by inclusion or convex-combination allow to improve significantly the refinement process. The interval based abstractions RSCG and SCG are more coarser abstractions preserving linear properties. These abstractions can be used to verify both timed linear properties and a subclass of TCTL properties.

REFERENCES

[1] R. Alur, D. Dill, *Automata for modeling real-time systems*, 17me ICALP, LNCS 443, 1990.

[2] B. Berthomieu, F. Vernadat, *State class constructions for branching analysis of time Petri nets*, LNCS 2619, 2003.

[3] H. Boucheneb, R. Hadjidj, *Towards optimal CTL* model checking of Time Petri Nets*, In Proc. of the International Workshop on Discrete Event Systems (WODES). Reims-France, 2004.

[4] H. Boucheneb, R. Hadjidj, *CTL* model checking for time Petri nets*, Theoretical Computer Science, 353(1-3)(1-3), 2006.

[5] F. Cassez and O. H. Roux, *Structural translation from time Petri nets to timed automata*, Journal of Systems and Software, 2006.

[6] E. M. Clarke, O. Grumberg, and D. Peled, *Model Checking*, MIT Press, Cambridge, MA, 1999.

[7] G. Gardey, O. H. Roux, *Using zone graph methode for computing the state space of a time Petri net*, In Formal Modeling and Analysis of Timed Systems (FORMATS), LNCS, 2003. Springer-Verlag.

[8] R. Hadjidj, H. Boucheneb, *On-the-fly TCTL model checking for time Petri nets using the state class method*, In Proc of the Sixth International Conference on Application of Concurrency to System Design (ACSD), IEEE Computer Society Press, 2006.

[9] R. Hadjidj, H. Boucheneb, *Much compact Time Petri Net state class spaces useful to restore CTL* properties*, In Proc of the Sixth International Conference on Application of Concurrency to System Design (ACSD), IEEE Computer Society Press, 2005

[10] R. Hadjidj, H. Boucheneb, *Improving state class constructions for CTL* model checking of time Petri nets*, International Journal on Software Tools Technology Transfer, accepted on July 2006.

[11] T. A. Henzinger, P-H. Ho, H. Wong-Toi, *HyTech : A Model Checker for Hybrid Systems* Software Tools for Technology Transfer 1, 1997.

[12] D. Lime and O. H. Roux, *State class timed automaton of a time Petri net*, In Proc. of the 10th Int. Workshop on Petri Nets and Performance Models (PNPM). IEEE Comp. Soc. Press, 2003.

[13] W. Penczek, A. Polrola, *Specification and Model Checking of Temporal Properties in Time Petri Nets and Timed Automata*, In Proc. of ICATPN, 2004.

[14] J. Toussaint, F. Simonot-Lion, J.P. Thomesse, *Time constraint verifications methods based time Petri nets*. In Proc. of the 6th Workshop on Future Trends in Distributed Computing Systems, 1997.

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

[16] S. Tripakis, S. Yovine and A. Bouajjani. *Checking Timed Buchi Automata Emptiness Efficiently*, In Formal Methods in System Design, 26(3), 2005.

[17] I. Virbitskaite, E. Pokozy, *A partial order method for the verification of time Petri nets*, In Fundamentals of Computation Theory, LNCS 1684, Springer-Verlag, 1999.

[18] T.Yoneda, H. Ryuba, *CTL Model Checking of Time Petri Nets Using Geometric Regions*, IEICE Trans. Inf. & Syst., Vol. E99-D, no. 3, 1998.

[7] p_1 and p_2 are atomic propositions over markings of a TPN.