Model Checking Bounded Prioritized Time Petri Nets

Bernard Berthomieu, Florent Peres, and François Vernadat

LAAS-CNRS, Université de Toulouse, Toulouse, France
fax: +33 (0)5.61.33.64.11 tel: +33 (0)5.61.33.63.63
e-mail:{Bernard.Berthomieu|Florent.Peres|Francois.Vernadat}@laas.fr

Abstract. In a companion paper [BPV06], we investigated the expressiveness of Time Petri Nets extended with Priorities and showed that it is very close to that Timed Automata, in terms of weak timed bisimilarity. As a continuation of this work we investigate here the applicability of the available state space abstractions for Bounded Time Petri Nets to Bounded Prioritized Time Petri Nets. We show in particular that a slight extension of the "strong state classes" construction of [BV03] provides a convenient state space abstraction for these nets, preserving markings, states, and LTL formulas. Interestingly, and conversely to Timed Automata, the construction proposed does not require to compute polyhedra differences.

1 Introduction

Since their introduction in [Mer74], Time Petri nets (TPN for short) have been widely used for the specification and verification of systems in which time plays an essential role like communication protocols, hardware, or realtime systems.

TPNs extend Petri nets with temporal intervals associated with transitions, specifying firing delay ranges for the transitions. Assuming transition t became last enabled at time θ, and the end-points of its time interval are α and β, then t cannot fire earlier than time $\theta + \alpha$ and must fire no later than $\theta + \beta$, unless disabled by firing some other transition. Firing a transition takes no time.

Finite state space abstractions for bounded TPN's, preserving various classes of properties, can be computed in terms of so-called state classes [BM83] [BD91] [YR98] [BV03] [Had06]. State classes abstract sets of states by a marking and a polyhedron capturing temporal information. The polyhedra can be represented by difference systems, built and compared in polynomial time.

Though priorities are pervasive in some families of realtime systems, they are not supported by the Time Petri Net models, and cannot be generally encoded within. In a companion paper [BPV06] we proposed an extension of TPNs with Priorities: in a Prioritized TPN ($PrTPN$ for short) a transition is not allowed to fire if some transition with higher priority is firable at the same instant. We then proved that priorities strictly extend the expressiveness of Time Petri nets, and in particular that Bounded $PrTPN$s can be considered equivalent to Timed Automata, in terms of weak timed bisimularity.

As a continuation of this work, we investigate in this paper state space abstractions for Bounded Prioritized Time Petri Nets. We first explain why the classical "state classes" construction of [BM83] is unable to cope with priorities. Then, we show that a minor extension of the "strong state classes" construction of [BV03] (also called "state zones" by some authors) is a suitable state space abstraction, preserving the markings, states, and the formulas of linear time temporal logics of the state space of $PrTPN$s. A refinement of this construction also preserves the CTL properties of the state space. Interestingly, and conversely to the constructions proposed for model checking Prioritized Timed Automata [LHHC05], [DHLP06], the constructions required for $PrTPN$s preserve convexity of state classes; they do not require to compute expensive polyhedra differences.

The paper is organized as follows. Section 2 recalls the definition, semantics and main properties of Prioritized Time Petri Nets. Section 3 discusses their state space abstractions; not all available abstractions for TPNs can be extended to cope with priorities. The modeling power of $PrTPN$s and the capabilities of the proposed abstractions is illustrated in Section 4 by a simple scheduling example. Finally, related work and some side-issues are discussed in Section 5.

2 Time Petri nets with priorities

2.1 Definition

Let $\mathbf{I}^+$ be the set of non empty real intervals with non negative rational endpoints. For $i \in \mathbf{I}^+$, $\downarrow i$ denotes its left end-point, and $\uparrow i$ its right end-point, or ∞ if i is unbounded. For any $\theta \in \mathbb{R}^+$, let $i \dot{-} \theta = \{x - \theta | x \in i \wedge x \geq \theta\}$.

Definition 1. *[BPV06] A* Prioritized Time Petri Net *(PrTPN for short) is a tuple* $\langle \mathbf{P}, \mathbf{T}, \mathbf{Pre}, \mathbf{Post}, \succ, m_0, \mathbf{I}_s \rangle$ *in which:*

- $\langle \mathbf{P}, \mathbf{T}, \mathbf{Pre}, \mathbf{Post}, m_0 \rangle$ *is a Petri net.* $\mathbf{P}$ *is the set of places,* $\mathbf{T}$ *is the set of transitions,* m_0 *is the initial marking, and* $\mathbf{Pre}$, $\mathbf{Post} : \mathbf{T} \to \mathbf{P} \to \mathbb{N}^+$ *are the precondition and postcondition functions, respectively.*
- $\mathbf{I_s} : \mathbf{T} \to \mathbf{I}^+$ *is the static interval function.*
- $\succ$ *is the priority relation, assumed irreflexive, asymmetric and transitive.*

TPNs extend PNs with the static Interval function $\mathbf{I}_s$, $PrTPN$s extend TPNs with the priority relation $\succ$ on transitions. Priorities will be represented by directed arcs between transitions, the source transition having higher priority.

For $f, g : \mathbf{P} \to \mathbf{N}^+$, $f \geq g$ means $(\forall p \in P)(f(p) \geq g(p))$ and $f\{+|-\}g$ maps $f(p)\{+|-\}g(p)$ with every p. A marking is a function $m : \mathbf{P} \to \mathbf{N}^+$. A transition $t \in \mathbf{T}$ is *enabled* at m iff $m \geq \mathbf{Pre}(t)$.

Definition 2. *A* state *of a TPN is a pair* $s = (m, I)$ *in which* m *is a marking and* I *is a function called the* interval *function. Function* $I : \mathbf{T} \to \mathbf{I}^+$ *associates a temporal interval with every transition enabled at* m*.*

The temporal components in states can be conveniently seen as firing domains, rather than interval functions: The *firing domain* of state (m, I) is the set of real vectors $\{\underline{\phi} | (\forall k)(\underline{\phi}_k \in I(k))\}$.

2.2 Semantics

Definition 3. *The semantics of a $PrTPN$ $\langle \mathbf{P}, \mathbf{T}, \mathbf{Pre}, \mathbf{Post}, \succ, m_0, \mathbf{I}_s \rangle$ is the timed transition system $\langle S, s_0, \rightsquigarrow \rangle$ where:*

- *S is the set of states (m, I) of the $PrTPN$*
- *$s_0 = (m_0, I_0)$ is the initial state, where m_0 is the initial marking and I_0 is the static interval function $\mathbf{I}_s$ restricted to the transitions enabled at m_0.*
- *$\rightsquigarrow \subseteq S \times \mathbf{T} \cup \mathbb{R}^+ \times S$ is the state transition relation, defined as follows ($(s, a, s') \in \rightsquigarrow$ is written $s \overset{a}{\rightsquigarrow} s'$):*
  - *Discrete transitions: we have $(m, I) \overset{t}{\rightsquigarrow} (m', I')$ iff $t \in \mathbf{T}$ and:*
    1) $m \geq \mathbf{Pre}(t)$
    2) $0 \in I(t)$
    3) $(\forall t' \in \mathbf{T})(m \geq \mathbf{Pre}(t) \wedge (t' \succ t) \Rightarrow 0 \notin I(t'))$
    4) $(\forall k \in \mathbf{T})(m' \geq \mathbf{Pre}(k) \Rightarrow$
    $I'(k) =$ *if* $k \neq t \wedge m - \mathbf{Pre}(t) \geq \mathbf{Pre}(k)$ *then* $I(k)$ *else* $\mathbf{I}_s(k))$
  - *Continuous transitions: we have $(m, I) \overset{\theta}{\rightsquigarrow} (m, I')$ iff $\theta \in \mathbb{R}^+$ and:*
    5) $(\forall k \in \mathbf{T})(m \geq \mathbf{Pre}(k) \Rightarrow \theta \leq \uparrow I(k))$
    6) $(\forall k \in \mathbf{T})(m \geq \mathbf{Pre}(k) \Rightarrow I'(k) = I(k) \dot{-} \theta)$

Transition t may fire from (m, I) if t is enabled at m, firable instantly, and no transition with higher priority satisfies these conditions. In the target state, the transitions that remained enabled while t fired (t excluded) retain their intervals, the others are associated with their static intervals. A continuous transition by θ is possible iff θ is not larger than the right endpoint of any enabled transition.

This definition differs from that used for Time Petri nets [BPV06] by the addition of condition (3), that prevents a transition to fire when some other transition also firable instantly has higher priority. Note that priorities do not modify the time-elapsing rules: all enabled transitions k are considered in condition (5), whether or not t has priority over k.

The *State Graph* of a $PrTPN$ is the timed transition system $SG = (S, s_0, \rightsquigarrow)$. From the properties of continuous transitions, any sequence of transitions of SG ending with a discrete transition is equivalent to a sequence alternating delay and discrete transitions, called a *firing schedule* or a *time transition sequence*.

Following [BM83], an alternative definition of the state space is often used for TPNs, capturing only the states reachable by some firing schedule. It amounts to interpret time-elapsing as nondeterminism. The *Discrete State Graph* of a $PrTPN$ is the triple $DSG = (S, s_0, \rightarrow)$, with $\rightarrow \subseteq S \times \mathbf{T} \times S$ and:

$$s \overset{t}{\rightarrow} s' \Leftrightarrow (\exists \theta)(\exists s'')(s \overset{\theta}{\rightsquigarrow} s'' \wedge s'' \overset{t}{\rightsquigarrow} s').$$

Any state of SG which is not in DSG is reachable from some state of DSG by a continuous transition. Both the SG and DSG are dense graphs: states may have uncountable numbers of successors.

A property of the state graphs of $PrTPN$s is worth to be mentioned: As in TPNs, but conversely to Timed Automata, at least one of the enabled transitions at a state is guaranteed to be firable after some delay. Time-elapsing may only increase the number of firable transitions and temporal deadlocks cannot happen.

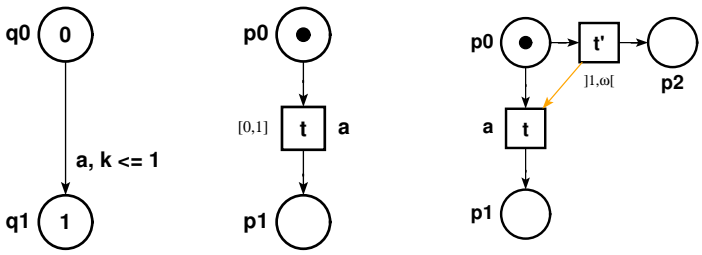
2.3 Properties

Boundedness: A PN is *bounded* if the marking of each place is bounded, boundedness implies finiteness of the set of reachable markings. Boundedness is undecidable for TPNs, and thus for $PrTPN$s, but there are a number of decidable and convenient sufficient conditions for this property [BM83], [BV07].

Composability: Adding to their definition a labeling of transitions by actions, a product operator can be defined for Prioritized Time Petri Nets. Under certain restrictions, this product is compositional in the sense that the semantics of a product of $PrTPN$s is equal to the product of their semantics [BPV06]. Compositional verification in presence of priorities has been investigated in length in a more general framework in [GS03] and other works by these authors.

Expressiveness: It was shown in [BCH$^+$05] that bounded TPNs are equivalent to *Timed automata* (TAs for short) in terms of language acceptance, but that TAs are strictly more expressive in terms of weak timed bisimilarity. Adding priorities to TPNs preserves their expressiveness in terms of language acceptance, but strictly increases their expressiveness in terms of weak timed bisimilarity: it is shown in [BPV06] that any TA with invariants of the form $\wedge_i(k_i \leq c_i)$ is weak time bisimilar to some bounded $PrTPN$, and conversely.

To illustrate this result, let us consider the TA and nets in the figure below. As shown in [BCH$^+$05], no TPN is weakly timed bisimilar with TA (a). In particular, the TPN (b) is not: when at location q_0 in the TA, time can elapse of an arbitrary amount, while time cannot progress beyond 1 in TPN (b). Consider now the $PrTPN$ (c), in which $t \prec t'$. Transition t' is silent and is firable at any time greater than 1, transition t bears label a and the default interval $[0, \infty[$. t may fire at any time not larger than 1, but not later, since t' is then firable and it has priority over t. Indeed, $PrTPN$ (c) is weakly timed bisimilar with TA (a).



(a) (b) (c)

3 State Space Abstractions for *PrTPN*s

The state graphs of $PrTPN$s are generally infinite, even when bounded. To model check $PrTPN$s, one needs finite abstractions of their state graphs. If G is

some state space, A some abstraction of it, and f a logical formula of the family one wish to preserve, then we must have $G \models f$ iff $A \models f$. Traditionally, we will focus on abstractions of the DSG rather than the SG.

For Time Petri nets, state space abstractions are available that preserve markings (including deadlocks) [BV07], markings and all traces [BM83], [BD91], states [BH06], states and traces [BV03], [Had06], and states, traces and branching properties [YR98], [BV03], [BH06]. We investigate in this section extensions of these abstractions to Prioritized Time Petri Nets, when applicable.

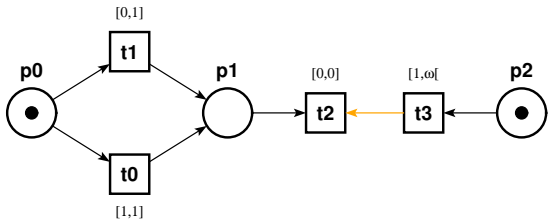
3.1 Abstractions preserving markings and traces

Definition 4. *For each firable sequence $\sigma \in \mathbf{T}^*$, let C_σ be the set of states inductively defined by:* $C_\epsilon = \{s_0\}$ *and* $C_{\sigma.t} = \{s' | (\exists s \in C_\sigma)(s \xrightarrow{t} s')\}$

C_σ the set of states reached in the discrete state graph by firing schedules of support sequence σ. Each state of the DSG is in some C_σ. For each such set C_σ, let $\mathcal{M}(C_\sigma)$ be the marking of any state in C_σ (all states in C_σ bear the same marking), and $\mathcal{F}(C_\sigma)$ be the union of the firing domains of all states in C_σ. Finally, let $C_\sigma \cong C_{\sigma'}$ iff $\mathcal{M}(C_\sigma) = \mathcal{M}(C_{\sigma'})$ and $\mathcal{F}(C_\sigma) = \mathcal{F}(C_{\sigma'})$.

The classical state class construction of [BM83], [BD91], termed SCG in the sequel, stems from the observation that, if $C_\sigma \cong C_{\sigma'}$, then any firing schedule firable from some state in C_σ is firable from state in $C_{\sigma'}$, and conversely. The state classes of [BM83] denote the above sets C_σ, for all firable sequences σ, considered modulo equivalence $\cong$. The set of classes is equipped with a transition relation: $C_\sigma \xrightarrow{t} X \Leftrightarrow C_{\sigma.t} \cong X$, it is finite iff the net is bounded.

The SCG is a convenient abstraction for LTL model checking, it preserves the markings and traces of the net. A weaker abstraction, only preserving markings, is obtained from the SCG by merging the classes related by inclusion of their firing domains. Unfortunately, both these abstractions are too coarse to preserve the effects of priorities. In fact, the founding observation that sets of states equivalent by $\cong$ have same futures in terms of firing schedules simply does not hold in presence of priorities, as illustrated by the following example.



Firing t_0 or t_1 in the above net leads to the same SCG class. Now, because t_3 has higher priority than t_2 and remains enabled while t_0 or t_1 fires, t_2 can never fire after t_0, and may only fire after t_1 if t_1 fired earlier than time 1.

3.2 Abstractions preserving states and traces

The previous SCG construction considers the sets C_σ in Definition 4 modulo equivalence $\cong$. In contrast, the *strong state classes* construction ($SSCG$ for short) proposed in [BV03], also called *state zones* by some authors, exactly coincides with those sets C_σ. For building the $SSCG$, one first needs a canonical representation for the sets C_σ, clock domains serve this purpose.

Clock domains: With each reachable state, one may associate a *clock function* γ. Function γ associates with each transition enabled at the state the time elapsed since it was last enabled. Clock functions may also be seen as vectors $\underline{\gamma}$ indexed over the transitions enabled.

In the $SSCG$ construction, a class is represented by a marking and a clock system, but classes still denote sets of states as in Definition 2 (defined from interval functions). The set of states denoted by a marking m and a clock system $Q = \{G\underline{\gamma} \leq \underline{g}\}$ is the set $\{(m, \Phi(\underline{\gamma}))|\underline{\gamma} \in \langle Q \rangle\}$, where $\langle Q \rangle$ is the solution set of Q and firing domain $\Phi(\underline{\gamma})$ is the solution set in $\underline{\phi}$ of:

$$\underline{0} \leq \underline{\phi}\ ,\ \underline{e} \leq \underline{\phi} + \underline{\gamma} \leq \underline{l} \ \text{ where } \ \underline{e}_k = \downarrow \mathbf{I}_s(k) \text{ and } \underline{l}_k = \uparrow \mathbf{I}_s(k)$$

Each clock vector denotes a state, but, unless the static intervals of all transitions are bounded, different clock vectors may denote the same state, and clock systems with different solution sets (possibly an infinity) may denote the same set of states. For this reason we introduce equivalence $\equiv$:

Definition 5. *Given $c = (m, Q = \{G\underline{\gamma} \leq \underline{g}\})$ and $c' = (m', Q' = \{G'\underline{\gamma}' \leq \underline{g}'\})$, let $c \equiv c'$ iff $m = m'$ and clock systems Q and Q' denote the same sets of states.*

Equivalence $\equiv$ is clearly decidable, efficient methods for checking it are discussed e.g. in [BV03] and [Had06], relying upon some relaxations of clock systems. When the static intervals of all transitions are bounded, $\equiv$ reduces to equality of the solution sets of the clock systems.

Construction of the $SSCG$: Strong state classes are represented by a marking m and a system $Q = \{G\underline{\gamma} \leq \underline{g}\}$ describing a clock domain for the enabled transitions. Clock variable $\underline{\gamma}_i$ is associated with the i^{th} transition enabled at m.

The first step in the computation of a successor class, when building the $SSCG$, is to determine which transitions are firable from the current class. In absence of priorities, transition t is firable from some state in class (m, Q) iff there is some delay $\theta \in \mathbb{R}^+$ such that, augmented by θ, the clocks of all enabled transitions are not larger than the right endpoint of their respective static intervals, and within that interval for transition t:

(a1) $\theta \geq 0$
(a2) $\theta + \underline{\gamma}_t \in \mathbf{I}_s(t)$
(a3) $(\forall i \neq t)(m \geq \mathbf{Pre}(i) \Rightarrow \theta + \underline{\gamma}_i \leq \uparrow \mathbf{I}_s(i))$

In presence of priorities, a fourth condition must be added, asserting that no enabled transition with higher priority than t is firable at θ:

(a4) $(\forall i)(m \geq \mathbf{Pre}(i) \wedge i \succ t \Rightarrow \theta + \underline{\gamma}_i \notin \mathbf{I}_s(i))$

$\theta + \underline{\gamma}_i \notin \mathbf{I}_s(i)$ holds iff $\theta + \underline{\gamma}_i > \uparrow \mathbf{I}_s(t)$ or $\theta + \underline{\gamma}_i < \downarrow \mathbf{I}_s(t)$, but the former case may not happen since it contradicts condition (a3). The $SSCG$ for a $PrTPN$ is built as for a TPN, just augmenting the enabling conditions as explained:

Algorithm 1 Computing the Strong State Class Graph with Priorities

- $R_\epsilon = (m_0, \{0 \leq \underline{\gamma}_t \leq 0 \mid m_0 \geq \mathbf{Pre}(t)\})$
- If σ is firable and $R_\sigma = (m, Q)$ then $\sigma.t$ is firable iff
  - (i) $m \geq \mathbf{Pre}(t)$
  - (ii) Q augmented with
    $\theta \geq 0$, $\theta \geq \downarrow \mathbf{I}_s(t) - \underline{\gamma}_t$
    $\{\theta \leq \uparrow \mathbf{I}_s(i) - \underline{\gamma}_i \mid m \geq \mathbf{Pre}(i)\}$
    $\{\theta < \downarrow \mathbf{I}_s(j) - \underline{\gamma}_j \mid m \geq \mathbf{Pre}(j) \wedge j \succ t\}$
    is consistent
- If $\sigma.t$ is firable then $R_{\sigma.t} = (m', Q')$ is computed from $R_\sigma = (m, Q)$:
  m' = m - $\mathbf{Pre}$(t) + $\mathbf{Post}$(t)
  Q' obtained by :
  - (a) A new variable is introduced, θ, constrained by (ii) above;
  - (b) $\forall k \in \mathbf{T} : m' \geq \mathbf{Pre}(k)$, introduce a new variable $\underline{\gamma'}_k$, such that:
    $\underline{\gamma'}_k = \underline{\gamma}_k + \theta$ if $k \neq t \wedge m - \mathbf{Pre}(t) \geq \mathbf{Pre}(k)$
    $0 \leq \underline{\gamma'}_k \leq 0$ otherwise
  - (c) The variables $\underline{\gamma}$ and θ are eliminated.

The temporary variable θ stands for the possible delays after which t can fire. There is an arc labeled t between R_σ and c iff $c \equiv R_{\sigma.t}$. As for Time Petri Nets, the $SSCG$ of a $PrTPN$ is finite iff the net is bounded. The clock systems are difference systems, for which canonical forms can be computed here in time complexity $\mathcal{O}(n^2)$, following the observation of [Rok93] that they can be computed incrementally. The $SSCG$ preserves all traces of the net and also permits to decide state reachability [BV07].

Compared to the methods proposed for model checking Prioritized Timed Automata [LHHC05] [DHLP06], Algorithm 1 does not require the expensive ($\mathcal{O}(n^4)$) DBM subtractions mandatory there. As was explained, this follows from the fact that time-elapsing in $PrTPN$s is bounded by the smallest of the deadlines of the enabled transitions, whatever the priority relation.

3.3 Abstractions preserving branching properties

The branching properties are those one can express in branching time temporal logics like CTL, or modal logics like HML or the μ-calculus. Neither the SCG (for TPNs) nor the $SSCG$ (for TPNs or $PrTPN$s) preserve these properties.

An abstraction preserving branching properties of the DSG was first proposed in [YR98], called the *Atomic state class graph* ($ASCG$ for short), for the subclass of TPNs in which all transitions have bounded static intervals. An alternative construction was proposed in [BV03], in which the $ASCG$ is obtained from the $SSCG$ of the net by a partition refinement process. This construction remains applicable to $PrTPN$s, it can be summarized as follows.

Bisimilarity is known to preserve branching properties. Let $\rightarrow$ be a binary relation over a finite set U, and for any $S \subseteq U$, let $S^{-1} = \{x | (\exists y \in S)(x \rightarrow y)\}$. A partition P of U is *stable* if, for any pair (A, B) of blocks of P, either $A \subseteq B^{-1}$ or $A \cap B^{-1} = \emptyset$. Computing a bisimulation, starting from an initial partition P of states, is computing a stable refinement of P [ACH+92]. In our case, a suitable initial partition of the state space is the set of classes of the $SSCG$ (it is a cover rather than a partition, but the method remains applicable). Computing the $ASCG$ is computing a stable refinement of the $SSCG$, the technical details can be found in [BV03], [BV07].

4 An example: Rate Monotonic Scheduling

This example illustrates the use of priorities for expressing scheduling policies on systems of tasks. The system is made of three realtime tasks, to be scheduled by a rate monotonic policy, the corresponding $PrTPN$ is shown in Figure 1.

Task i has the following transitions:

- $\mathbf{P}_i$ Periodically generates a token in place newP_i, representing a new period event from which a task will be released,
- $\mathbf{R}_i$ Marks the task release, i.e. the instant at which the task enter its idle state and waits to be started. Updates its state from done_i to notdone_i,
- $\mathbf{S}_i$ Starts the task, changes its state from idle_i to exec_i,
- $\mathbf{E}_i$ Completes the task: frees the resource and restores its state to done_i,
- $\mathbf{DL}_i$ Signals a deadline miss, occuring when the task is still executing and a new period event is generated.

A scheduler in a realtime system is often coded by a *policy*, which tells, when the unpoliticized system is nondeterministic, how to resolve this nondeterminism. The rate monotonic policy gives priorities to tasks according to their period value, the task with smaller period having the highest priority. The policy is expressed here using priorities. In this example, the upper task has the smallest period (3), followed by the middle task (5), and the lower task (11).

Thus, the RM policy requires here that every transition of T1 that can lead to a state enabling S_i in zero time has to possess a higher priority than all the corresponding transitions in T2, and similarly for T2 and T3. That is $P_1, R_1, S_1 \succ P_2, R_2, S_2 \succ P_3, R_3, S_3$ (to preserve readability, priorities are omitted in Figure 1). This priority relation represents exactly the RM policy.

Algorithm 1 has been implemented and integrated in an experimental version of the *Tina* toolbox [BRV04]. For this example, the unprioritized $SSCG$,

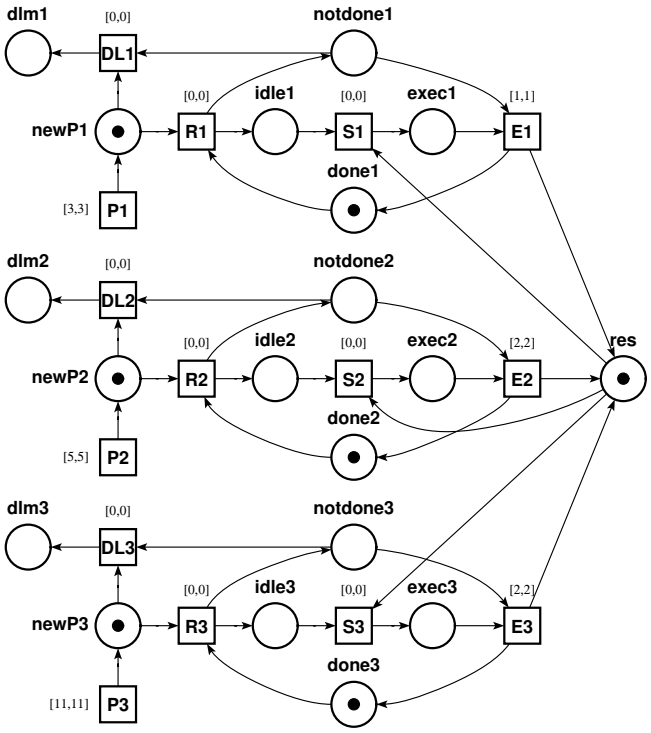


Fig. 1. A task system $(P_1, R_1, S_1 \succ P_2, R_2, S_2 \succ P_3, R_3, S_3)$

computed by *Tina* for the net with priorities removed, is unbounded. Taking priorities into account, the tool builds a finite $SSCG$ with 557 state classes and 671 transitions. All markings are safe (the places hold at most one token) and it can be checked on the graph produced that no deadline miss can occur (transitions DLi are dead).

5 Conclusion

The results presented in this paper increase the range of systems one can represent and analyze using Time Petri nets and similar models. Beside their modeling convenience, priorities do extend the expressiveness of Time Petri nets. Further, it has been shown that Prioritized Petri nets can be analyzed with methods similar to the well known state class based methods for Time Petri nets. These methods are easy to implement and have a tractable complexity.

As mentioned in the text, analyzing Prioritized Time Petri nets does not require to use polyhedra differences, adding priorities to TPNs does not augment the complexity of computation of classes.

The version of Prioritized TPNs introduced in this paper makes use of the simplest possible notion of priority: static priorities. Technically, nothing prevents to replace it by more flexible notions of priorities like dynamic priorities depending on markings.

References

[ACH+92] R. Alur, C. Courcoubetis, N. Halbwachs, D.L. Dill, and H. Wong-Toi. Minimization of timed transition systems. In *CONCUR'92, Springer LNCS 630*, pages 340–354, 1996.

[BCH+05] B. Bérard, F. Cassez, S. Haddad, O. H. Roux, and D. Lime. Comparison of the Expressiveness of Timed Automata and Time Petri Nets. In *FORMATS'05, Springer LNCS 3829*, pages 211–225, 2005.

[BD91] B. Berthomieu and M. Diaz. Modeling and verification of time dependent systems using time Petri nets. *IEEE Tr. on Soft. Eng.*, 17(3):259–273, 1991.

[BH06] H. Boucheneb and R. Hadjidj. Using inclusion abstraction to construct atomic state class graphs for time petri nets. In *International Journal of Embedded Systems*, Vol. 2, No 1/2, June 2006.

[BM83] B. Berthomieu and M. Menasche. An enumerative approach for analyzing time Petri nets. *IFIP Congress Series*, 9:41–46, 1983.

[BPV06] B. Berthomieu, F. Peres, and F. Vernadat. Bridging the gap between timed automata and bounded time petri nets. In *FORMATS'06, Springer LNCS 4202*, pages 82–97, 2006.

[BRV04] B. Berthomieu, P.-O. Ribet, and F. Vernadat. The tool TINA – construction of abstract state spaces for Petri nets and time Petri nets. *International Journal of Production Research*, 42(14):2741–2756, 15 July 2004.

[BV03] B. Berthomieu and F. Vernadat. State class constructions for branching analysis of time Petri nets. In *TACAS'2003, Warsaw, Poland, Springer LNCS 2619*, pages 442–457, 2003.

[BV07] B. Berthomieu and F. Vernadat. *State Space Abstractions for Time Petri Nets.* Handbook of Real-Time and Embedded Systems, Insup Lee, Joseph Y-T. Leung and Sang Son, CRC Press, Boca Raton, FL., U.S.A., 2007.

[DHLP06] A. David, J. Håkansson, K. G. Larsen, and P. Pettersson. Model checking timed automata with priorities using DBM subtraction. In *FORMATS'06, Springer LNCS 4202*, pages 128–142, 2006.

[GS03] G. Goessler and J. Sifakis. Priority systems. In *Proceedings of FMCO'2003, Leiden, the Netherlands, Springer LNCS 3188*, pages 314–329, 2003.

[Had06] R. Hadjidj. *Analyse et validation formelle des systèmes temps réel.* PhD Thesis, Ecole Polytechnique de Montréal, Univ. de Montréal, February 2006.

[LHHC05] Shang-Wei Lin, Pao-Ann Hsiung, Chun-Hsian Huang, and Yean-Ru Chen. Model checking prioritized timed automata. In *ATVA'2005, Springer LNCS 3707*, pages 370–384, 2005.

[Mer74] P. M. Merlin. *A Study of the Recoverability of Computing Systems.* PhD Thesis, Irvine, 1974.

[Rok93] T. G. Rokicki. *Representing and Modeling Circuits.* PhD Thesis, Stanford Univ., Stanford, CA, 1993.

[YR98] T. Yoneda and H. Ryuba. CTL model checking of Time Petri nets using geometric regions. *IEEE Trans. on Inf. and Systems*, E99-D(3):1–10, 1998.