

Reachability Problems and Abstract State Spaces for Time Petri Nets with Stopwatches

Bernard Berthomieu¹, Didier Lime², Olivier H. Roux², and François Vernadat¹

¹ LAAS-CNRS, 7, Avenue du Colonel Roche, 31077 Toulouse Cedex, France
tel: +33.5.61.33.63.00, fax: +33.5.61.33.64.11

`Bernard.Berthomieu@laas.fr`, `Francois.Vernadat@laas.fr`

² IRCCyN, 1, rue de la Noë, BP 92101, 44321 Nantes Cedex 3, France
tel: +33.2.40.37.69.00, fax: +33.2.40.37.69.30

`Didier.Lime@irc cyn.ec-nantes.fr`, `Olivier-h.Roux@irc cyn.ec-nantes.fr`

Abstract. Several extensions of Time Petri nets (TPN's) have been proposed for modeling suspension and resumption of actions in timed systems. We first introduce a simple class of TPN's extended with stopwatches (SwTPN's), and present a semi-algorithm for building exact representations of the behavior of SwTPN's, based on the known state class method for Time Petri nets. Then, we prove that state reachability in SwTPN's and all similar models is undecidable, even when bounded, which solves an open problem. Finally, we discuss overapproximation methods yielding finite abstractions of their behavior for a subclass of bounded SwTPN's, and propose a new one based on a quantization of the polyhedra representing temporal information. By adjusting a parameter, the exact behavior can be approximated as closely as desired. The methods have been implemented, experiments are reported.

Keywords: Time Petri nets, stopwatches, state classes, reachability, decidability, approximation, real-time systems modeling and verification.

1 Introduction

Modeling of many embedded systems requires to express suspension and resumption of actions. Addressing this issue, several models have been proposed.

Extending Timed Automata (TA's), Stopwatch automata (SWA's) are a subclass of Linear Hybrid automata (LHA's) in which variable derivatives can only take two values expressing progress (1) or suspension (0). SWA's and LHA's have equivalent reachability problems (Cassez *et al.*, 2000), a problem known undecidable for LHA's (Alur *et al.*, 1995). As no decidable subclass of SWA's that preserve these modeling capabilities has been identified, finite state space abstractions are typically obtained by overapproximations, characterizing sets of states including the exact state space, but possibly larger. Such approximations yield sufficient conditions for safety properties.

Time Petri nets (TPN's) (Merlin, 1974) are another widely used model for real-time systems. Time Petri nets extend Petri nets with temporal intervals associated with transitions, specifying firing delay ranges for the transitions. State

space abstractions for TPN's preserving various classes of properties can be computed, in terms of so called state classes (Berthomieu *et al.*, 1983) (Berthomieu *et al.*, 1991) (Berthomieu *et al.*, 2003). State classes represent sets of states by a marking and a polyhedron capturing temporal information. State reachability is undecidable for TPN's, but decidable for bounded TPN's, sufficient for virtually all practical purposes.

Several extensions of TPN's have been proposed that address the modeling of suspension and resumption of actions, including Scheduling-TPN's (Roux *et al.*, 2002) (Lime *et al.*, 2003), Preemptive-TPN's (Bucci *et al.*, 2004), and Inhibitor Hyperarc TPN's (IHTPN's) (Roux *et al.*, 2004). The first two add resources and priorities primitives to the TPN model, IHTPN's introduce special inhibitor arcs that control the progress of transitions. Since all extend TPN's, their state reachability problem is undecidable, but state reachability for bounded nets, of high practical interest, remains an open problem.

For all these extensions, semi-algorithms are available that compute state space abstractions in terms of state classes. But, as for SWA's, no expressive enough decidable subclass has been identified. State space overapproximation methods are available too, approximating the polyhedron that characterizes temporal information in a state class by the smallest polyhedron denotable by a DBM that includes it. The method is efficient, but the overapproximations obtained are often too coarse.

In this article, we first introduce a simple model of Time Petri nets with stopwatch capabilities, called Stopwatch Time Petri nets (SwTPN's for short). SwTPN's extend TPN's by stopwatch arcs, that control the progress of transitions. They can be seen as a simplification of IHTPN's (Roux *et al.*, 2004).

We then prove that state reachability is undecidable for SwTPN's, even when bounded. It follows that many interesting properties of these nets are undecidable, and that these problems are also undecidable for all extensions of TPN's discussed above. Classically, the proof reduces state reachability in bounded SwTPN's to state reachability in Minsky's 2-counter machines.

The algorithms computing state class graphs for TPN's are easily adapted to operate on SwTPN's. They yield exact state space abstractions, but, as a consequence of the above undecidability result, boundedness of the SwTPN does not imply finiteness of the graphs of state classes. For ensuring termination on a class of bounded SwTPN's, we propose a new overapproximation method based on quantization of the polyhedra representing temporal information in state classes. By adjusting a parameter, the exact behavior of the SwTPN can be approximated as closely as desired. Both the exact and approximate computation methods have been implemented in an extension of the TINA tool (Berthomieu *et al.*, 2004).

The paper is organized as follows: Section 2 recalls the essentials about Time Petri nets and summarizes their decidability results. Section 3 introduces Stopwatch Time Petri nets and a semi-algorithm for computing their state classes. An example is presented in Section 4. Undecidability of state reachability for SwTPN's is established in Section 5. Section 6 explains the polyhedra quan-

tization method for computing overapproximations of state spaces of bounded SwTPN's, and discusses some experiments handled with the help of an experimental implementation of the methods.

2 Time Petri nets

2.1 Time Petri nets, states, state graphs

Let \mathbf{I}^+ be the set of nonempty real intervals with nonnegative rational end-points. For $i \in \mathbf{I}^+$, $\downarrow i$ denotes its left end-point, and $\uparrow i$ its right end-point (if i bounded) or ∞ . For any $\theta \in \mathbf{R}^+$, $i \dot{-} \theta$ denotes the interval $\{x - \theta \mid x \in i \wedge x \geq \theta\}$.

Definition 1. A Time Petri net (or TPN) is a tuple $\langle P, T, \mathbf{Pre}, \mathbf{Post}, m_0, I_s \rangle$, in which $\langle P, T, \mathbf{Pre}, \mathbf{Post}, m_0 \rangle$ is a Petri net, and $I_s : T \rightarrow \mathbf{I}^+$ is a function called the Static Interval function.

Function I_s associates a temporal interval $I_s(t) \in \mathbf{I}^+$ with every transition of the net. $Eft_s(t) = \downarrow I_s(t)$ and $Lft_s(t) = \uparrow I_s(t)$ are called the static earliest and latest firing times of t , respectively.

States, and the temporal state transition relation $\xrightarrow{t@\theta}$, are defined as follows:

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

We write $(m, I) \xrightarrow{t@\theta} (m', I')$ iff $\theta \in \mathbf{R}^+$ and:

1. $m \geq \mathbf{Pre}(t) \wedge \theta \geq \downarrow I(t) \wedge (\forall k \in T)(m \geq \mathbf{Pre}(k) \Rightarrow \theta \leq \uparrow I(k))$
2. $m' = m - \mathbf{Pre}(t) + \mathbf{Post}(t)$
3. $(\forall k \in T)(m' \geq \mathbf{Pre}(k) \Rightarrow$
 $I'(k) = \text{if } k \neq t \wedge m - \mathbf{Pre}(t) \geq \mathbf{Pre}(k) \text{ then } I(k) \dot{-} \theta \text{ else } I_s(k))$

We have $s \xrightarrow{t@\theta} s'$ if firing transition t from s at time θ after t became last enabled leads to state s' . (1) ensures that transitions fire in their temporal interval, unless disabled by firing some other transition. (2) is the standard marking transformation. From (3), transitions persistent wrt t have their current interval shifted by θ and truncated to nonnegative times, and newly enabled transitions are assigned their static intervals. Transitions that remained enabled during their own firing (that is such that $m - \mathbf{Pre}(t) \geq \mathbf{Pre}(t)$) are considered newly enabled, alternative treatments for such transitions are discussed in (Berthomieu, 2001). Firing a transition takes no time.

Note, in the definition above, that only state changes involving a discrete transition are considered. Transitions resulting from time progress could be added as well, but this does not bring any particular benefit for TPN's as no discrete transition can be disabled by just letting time progress. So, delay transitions are always agglomerated with some discrete transition.

The *state graph* SG of a TPN is the set of its states reachable from the initial state $s_0 = (m_0, I_0)$, where $I_0(t) = I_s(t)$ for any t enabled at m_0 , equipped with

the above timed state transition relation. As transitions may fire at any time in their temporal intervals, states typically admit an infinity of successors.

The notation $s \xrightarrow{t} s'$ stands for $(\exists \theta)(s \xrightarrow{t @ \theta} s')$, $s \longrightarrow s'$ for $(\exists t)(s \xrightarrow{t} s')$, and $s \xrightarrow{\sigma} s'$, with $\sigma \in T^*$, for $s \xrightarrow{\sigma_1} \dots \xrightarrow{\sigma_n} s'$. A *firing schedule* is a sequence of successively fireable transitions, called its *support*, together with their relative firing times. The *firing domain* of state (m, I) is the set of vectors $\{\underline{\phi} | (\forall k)(\underline{\phi}_k \in I(k))\}$, with components indexed by the enabled transitions.

2.2 Decidable and undecidable properties

\mathcal{R} being the set of reachable states of some TPN, consider the following problems:

- (1) The *marking reachability* problem: given m , $(\exists(m', I) \in \mathcal{R})(m' = m)$
- (2) The *boundedness* problem: $(\exists b \in \mathbf{N})(\forall(m, I) \in \mathcal{R})(\forall p \in P)(m(p) \leq b)$
- (3) The *k-boundedness* problem: given $k \in \mathbf{N}$, $(\forall(m, I) \in \mathcal{R})(\forall p \in P)(m(p) \leq k)$
- (4) The *state reachability* problem: given s , $s \in \mathcal{R}$
- (5) The *liveness* problem: $(\forall s \in \mathcal{R})(\forall t \in T)(\exists \sigma \in T^*)(\exists s' \in \mathcal{R})(s \xrightarrow{\sigma, t} s')$

For arbitrary TPN's, problem (1) is known undecidable from (Jones *et al.*, 1977). It directly follows that (2), (4) and (5) are undecidable too. Problem (3) is decidable: The *state class graph* construction of (Berthomieu *et al.*, 1983) terminates whenever applied to a bounded TPN, producing a graph that preserves the markings and firing sequences of the state graph. As any k -bounded TPN is also bounded, this construction decides problem (3): A net is k -bounded iff no state class is found, in the course of the construction, with the marking of some place overflowing k .

Though boundedness of TPN's is undecidable, there exists decidable sufficient properties for it (Berthomieu *et al.*, 1983). One, for instance, is that the underlying Petri net is bounded, which is known decidable. Also, some useful classes of TPN's are bounded "by construction". For bounded TPN's (thus assuming (2)), we have in addition:

Theorem 1. *Marking reachability (1), state reachability (4) and liveness (5) are decidable for bounded TPN's.*

Proof. (1) is clearly decided by the construction of (Berthomieu *et al.*, 1983). For (4): for bounded TPN's, the Strong state classes construction of (Berthomieu *et al.*, 2003) yields a finite graph in which classes canonically denote state sets, represented by a marking and an inequality system in clock variables $Q\underline{\gamma} \leq \underline{g}$. Given a state $s = (m, I)$, one can easily compute a clock vector $\underline{\delta}$ such that, for all transitions k enabled at m : $I(k) = I_s(k) \dot{-} \underline{\delta}_k$. State (m, I) is reachable iff there is a strong class $(m', Q\underline{\gamma} \leq \underline{g})$ such that $m = m'$ and $\underline{\delta}$ computed above satisfies $Q\underline{\delta} \leq \underline{g}$. For (5): the Atomic state class graph construction also introduced in (Berthomieu *et al.*, 2003) produces, for bounded TPN's, a state class graph bisimilar with its state graph (omitting temporal annotations). Adapting the known decision method for liveness of bounded Petri nets, a bounded TPN is live iff all its transitions appear as edge labels in all pending strong connected components of its atomic state class graph. \square

3 Stopwatch Time Petri nets

3.1 SwTPN's, states, state graphs

Definition 3. A Stopwatch Time Petri net (SwTPN for short) is a tuple $\langle P, T, \mathbf{Pre}, \mathbf{Post}, \mathbf{Sw}, m_0, I_s \rangle$, in which $\langle P, T, \mathbf{Pre}, \mathbf{Post}, m_0, I_s \rangle$ is a Time Petri net and $\mathbf{Sw} : T \rightarrow P \rightarrow \mathbf{N}$ is a function called the stopwatch incidence function.

Stopwatch Time Petri Nets add function \mathbf{Sw} to Time Petri Nets. \mathbf{Sw} associates an integer with every $(p, t) \in P \times T$, values greater than 0 are represented by special arcs, called *stopwatch arcs*, possibly weighted, represented with square shaped arrows. Figure 1 shows a Stopwatch Time Petri net, the arc from place p_3 to transition t_4 is a stopwatch arc of weight 1.

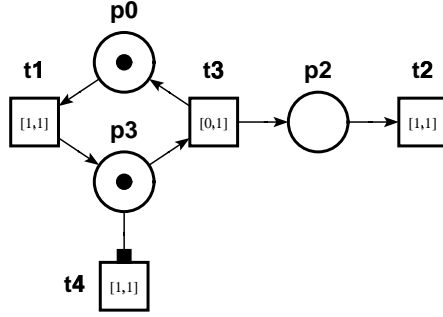


Fig. 1. A Stopwatch Time Petri net

As usual, a transition t is *enabled* at marking m iff $m \geq \mathbf{Pre}(t)$. In addition, a transition enabled at m is *active* iff $m \geq \mathbf{Sw}(t)$, otherwise it is said *suspended*. States, and the temporal state transition relation $\xrightarrow{t@\theta}$, are defined as follows:

Definition 4. A state of a SwTPN is a pair $s = (m, I)$ in which m is a marking and I , the interval function, associates a temporal interval in \mathbf{I}^+ with every transition enabled at m . We write $(m, I) \xrightarrow{t@\theta} (m', I')$ iff $\theta \in \mathbf{R}^+$ and:

1. $m \geq \mathbf{Pre}(t) \wedge m \geq \mathbf{Sw}(t) \wedge \theta \geq \downarrow I(t) \wedge (\forall k \in T)(m \geq \mathbf{Pre}(k) \wedge m \geq \mathbf{Sw}(k) \Rightarrow \theta \leq \uparrow I(k))$
2. $m' = m - \mathbf{Pre}(t) + \mathbf{Post}(t)$
3. $(\forall k \in T)(m' \geq \mathbf{Pre}(k) \Rightarrow I'(k) = \text{if } k \neq t \wedge m - \mathbf{Pre}(t) \geq \mathbf{Pre}(k) \text{ then if } m \geq \mathbf{Sw}(k) \text{ then } I(k) \dot{-} \theta \text{ else } I(k) \text{ else } I_s(k)$

Compared to TPN's (see Definition 2 in Section 2.1), only active transitions may fire, and, in item (3), persistent but frozen transitions have their temporal intervals unchanged. So, when a frozen transition becomes active again, due to

a change in marking, it resumes with the temporal interval captured in the state rather than its static interval. Stopwatch arcs allow to model suspension and resumption of transitions, note that they do not convey ant tokens.

As an illustration, let us fire some transitions in the net Figure 1. The interval functions in states can be represented by systems of inequalities with one variable for each enabled transition (their solution set is the firing domain of the state).

The initial state of the net is $s_0 = (m_0, I_0)$, where m_0 is its initial marking and interval I_0 can be represented by the inequality system D_0 (ϕ_t is the variable associated with transition t):

$$\begin{aligned} m_0 &: p_0, p_3 \\ D_0 &: 1 \leq \phi_{t_1} \leq 1 \\ & 0 \leq \phi_{t_3} \leq 1 \\ & 1 \leq \phi_{t_4} \leq 1 \end{aligned}$$

All enabled transitions are active at s_0 . Waiting $\theta_3 \in [0, 1]$ units of time then firing t_3 , or, this is equivalent, firing t_3 at relative time θ_3 , leads to a state $s_1 = (m_1, D_1)$, described by:

$$\begin{aligned} m_1 &: p_0 * 2, p_2 \\ D_1 &: 1 - \theta_3 \leq \phi_{t_1} \leq 1 - \theta_3 \\ & 1 \leq \phi_{t_2} \leq 1 \\ & 1 - \theta_3 \leq \phi_{t_4} \leq 1 - \theta_3 \end{aligned}$$

Transition t_4 is suspended at s_1 , t_1 and t_2 are active. From s_1 , only t_1 may fire, after a delay of $1 - \theta_3$. Firing it leads to a state $s_2 = (m_2, D_2)$ described by:

$$\begin{aligned} m_2 &: p_0, p_2, p_3 \\ D_2 &: 1 \leq \phi_{t_1} \leq 1 \\ & \theta_3 \leq \phi_{t_2} \leq \theta_3 \\ & 0 \leq \phi_{t_3} \leq 1 \\ & 1 - \theta_3 \leq \phi_{t_4} \leq 1 - \theta_3 \end{aligned}$$

Note that the interval of t_4 did not change while going from s_1 to s_2 , while that of t_2 was shifted by $1 - \theta_3$. All enabled transitions are active at s_2 .

The notations and concepts introduced for TPN's at the end of Section 2.1 naturally apply to SwTPN's too. In particular, *State graphs* for SwTPN's are defined similarly to TPN's, from the above $\xrightarrow{t@\theta}$ relation.

3.2 State classes for SwTPN

As for TPN's, the states of a SwTPN typically have an infinity of successors. The constructions that provide state space abstractions for TPN's: the state class graph of (Berthomieu *et al.*, 1983), and the strong and atomic state class graphs of (Berthomieu *et al.*, 2003), are easily adapted to SwTPN's. We adapt below the former, that preserves markings and *LTL* properties of the state space.

In that construction, state classes are denoted by pairs (m, D) , where m is a marking and D is a firing domain, described by an inequality system $A\phi \leq \underline{b}$. Variable ϕ_i ranges over the times at which the i^{th} transition enabled at m may fire. Let us write $(m, D = \{A\phi \leq \underline{b}\}) \equiv (m', D' = \{A'\phi \leq \underline{b}'\})$ when $m = m'$, and D and D' have equal solution sets. The graph of state classes of a SwTPN is built as follows:

Algorithm 1 (Computing state classes)

For each firable firing sequence σ , a class C_σ can be computed as explained below. Compute the smallest set C including C_ϵ and such that, whenever $C_\sigma \in C$ and $\sigma.t$ is firable, then either $C_{\sigma.t} \in C$, or $C_{\sigma.t}$ is equivalent by \equiv to some class in C . There is an arc labeled t between classes C_σ and c iff $c \equiv C_{\sigma.t}$.

- The initial class is $C_\epsilon = (m_0, \{Eft_s(t) \leq \phi_t \leq Lft_s(t) \mid \mathbf{Pre}(t) \leq m_0\})$
- If σ is firable and $C_\sigma = (m, D = \{A\phi \leq b\})$, then $\sigma.t$ is firable iff:
 - (i) $m \geq \mathbf{Pre}(t) \wedge m \geq \mathbf{Sw}(t)$ (t is enabled and active at m)
 - (ii) System $D \cup \{\phi_t \leq \phi_i \mid i \neq t \wedge m \geq \mathbf{Pre}(i) \wedge m \geq \mathbf{Sw}(i)\}$ is consistent
- If $\sigma.t$ is firable, then $C_{\sigma.t} = (m', D')$ is computed from $C_\sigma = (m, D)$ by:
 - $m' = m - \mathbf{Pre}(t) + \mathbf{Post}(t)$
 - D' obtained by:
 1. The firability constraints for t in (ii) above are added to D .
 2. For each k enabled at m' , a new variable ϕ'_k is introduced, obeying:

$$\begin{aligned} \phi'_k &= \phi_k - \phi_t && \text{if } k \neq t, m - \mathbf{Pre}(t) \geq \mathbf{Pre}(k), \text{ and } m \geq \mathbf{Sw}(k) \\ \phi'_k &= \phi_k && \text{if } k \neq t, m - \mathbf{Pre}(t) \geq \mathbf{Pre}(k), \text{ and } \neg(m \geq \mathbf{Sw}(k)) \\ \phi'_k &\in I_s(k) && \text{otherwise} \end{aligned}$$
 3. Variables ϕ are eliminated.

For TPN's (without stopwatch arcs), the set of distinct systems D one can build by Algorithm 1 is finite (Berthomieu *et al.*, 1983), whether the TPN is bounded or not. So, bounded TPN's admit finite state class graphs. Further, systems D are difference systems (or DBM's), for which canonical forms can be computed in polynomial time in the number of variables, so equivalence \equiv can be checked efficiently.

Unfortunately, these properties do not hold if stopwatch arcs are present. Describing firing domains requires richer inequality systems, and boundedness of a SwTPN does not imply finiteness of its set of state classes. Equivalence \equiv remains decidable, however.

As an example, consider the net represented Figure 1. By simple temporal arguments, it can be shown that this net is bounded. In this net, schedules of support $\sigma_n = t_3.t_1.(t_3.t_2.t_1)^n.t_2.t_4$, for any $n \in \mathbf{N}$, are firable from the initial state, yielding by Algorithm 1 an infinite series of state classes, all with the same marking. Sequence σ_n leads to the following class (the corresponding clock space are also given, as would be computed by the alternative “strong classes” construction of (Berthomieu *et al.*, 2003)):

$$\begin{aligned} \text{marking} &= p_0 p_3 \\ \text{firing domain} &= \{\phi_{t_4} = 1, 0 \leq \phi_{t_3}, \phi_{t_3} \leq \phi_{t_1} \leq (n+1)/(n+2)\} \\ \text{clock space} &= \{\underline{\gamma}_{t_4} = 0, \underline{\gamma}_{t_1} = \underline{\gamma}_{t_3}, 1/(n+2) \leq \underline{\gamma}_{t_1} \leq 1\} \end{aligned}$$

But setting the static interval of transition t_1 to $[2, 2]$, for instance, yields a finite graph of state classes with 25 classes and 38 transitions.

So, Algorithm 1 does not in general terminate. To enforce termination on bounded SwTPN's, the algorithm must be combined with an overapproximation of the abstract state space. Overapproximations for Preemptive-TPN's,

Scheduling-TPN's and IHTPN's have been proposed in (Bucci *et al.*, 2004), (Lime *et al.*, 2003), and (Roux *et al.*, 2004), which amount to replace the class obtained at step (3) in Algorithm 1 by the smallest class denotable by a difference system (DBM) that includes the class computed. Such approximations, available e.g. in the ROMEO tool (Gardey *et al.*, 2005) for Scheduling TPN's, yield sufficient conditions for safety properties, but are sometimes too coarse in practice. An alternative overapproximation technique will be proposed in Section 6.

When Algorithm 1 terminates, it produces a finite abstraction of the state graph that preserves its markings and *LTL* properties. The “strong” and “atomic” state class graphs for TPN's introduced in (Berthomieu *et al.*, 2003) could be easily adapted to SwTPN's too, the former preserves states and *LTL* properties, and the latter states and *CTL** properties. Other constructions of interest are the variants of the basic and ‘strong’ versions obtained by merging a class with any class including it, as done for Timed Automata zones in (Daws *et al.*, 1998) or TPN's strong classes in (Boucheneb *et al.*, 2004). These alternatives only preserve markings or states (not *LTL*), but typically produce smaller graphs.

4 A simple example

Though similar algorithms are available for all TPN extensions referred to in the Introduction, none have been implemented at the time this paper was written. We report in this Section some experiments with an implementation of Algorithm 1 for SwTPN's embedded in an extension of the TINA tool (Berthomieu *et al.*, 2004). For polyhedral operations, the implementation relies on the *NewPolka* library (Jeannot, 2002). The prototype implementation is available from the authors.

Experiments suggests that the exact characterizations of the behaviors of SwTPN's obtained by Algorithm 1 are finite in many practical cases. To illustrate this, let us consider a simple scheduling example proposed in (Bucci *et al.*, 2004). The example is a model of three independent tasks: Two periodic tasks of period 50 and 150 units, and a sporadic task with a minimum interarrival time of 100. Task 1 (period 50) has priority over both others, and task 2 (sporadic) has priority over task 3. This example is easily translated into Scheduling-TPN's or SwTPN's. The corresponding SwTPN's is shown in Fig. 2 (in black).

Typical properties of interest for such applications are schedulability (that the current instance of each task is complete before another starts) and quantitative properties like worst case response time (WCRT).

Schedulability is satisfied if the net is safe. The DBM overapproximated state space of (Roux *et al.*, 2002, Bucci *et al.*, 2004) yields a graph of 637 classes, all with safe markings. To check if some run is indeed possible, (Bucci *et al.*, 2004) proposes a method to compute feasible firing schedules for the transition sequences given by the overapproximated state class graph, by solving a linear programming problem. Ad-hoc methods for checking quantitative timed properties are also proposed.

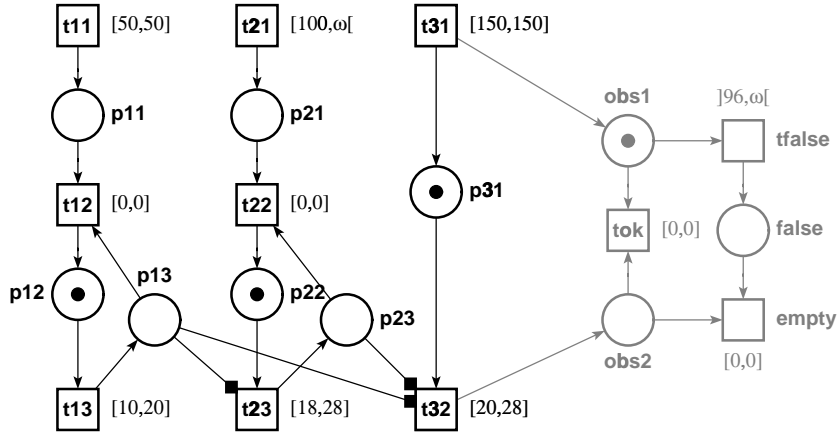


Fig. 2. SwTPN of two periodic and a sporadic processes and an observer

For this example, Algorithm 1 builds a graph of 323 classes and 477 transitions. All markings are safe, which implies schedulability. Quantitative properties can be checked by the use of observers. Fig. 2 shows for instance a nonintrusive observer (pictured in grey) for the property: "task 3 is always executed in less than 96 units of time". The property is satisfied if place `false` is never marked. The exact state class graph computed for the extended net confirms that the property holds, and, updating the observer, that it becomes false if the firing interval of `tfalse` includes 96. The WCRT of this task is thus 96 time units.

From the DBM approximated graph of (Roux *et al.*, 2002), we would obtain at best an estimated WCRT equal to 144 time units, significantly larger than the exact value of 96 we found. Finally, if the execution time of task 3 (`t32`) is increased to $[20,38]$, then the DBM approximated state class graph becomes infinite, while the exact state class graph is still finite. In this case the methods of (Roux *et al.*, 2002) or (Bucci *et al.*, 2004) cannot be applied.

5 Decidability issues for Stopwatch Time Petri nets

SwTPN's clearly include TPN's. So, the undecidability results for TPN's mentioned in Section 2.2 apply to SwTPN's too: boundedness, marking reachability, state reachability, and liveness are undecidable for arbitrary SwTPN's. Considering bounded TPN's, all these problems were found decidable, the question addressed in this Section is whether these problems remain decidable for bounded SwTPN's.

Unfortunately, the answers are negative, as will be seen. It is proven in this Section that state reachability for SwTPN's can be reduced to the halting problem for 2-counter machines, which in turns implies undecidability of the other problems. After recalling the structure of such machines, an encoding

into SwTPN's is developed, and the undecidability results derived. The ideas underlying this encoding are similar to those used in (Henzinger *et al.*, 1995) for encoding 2-counter machines into a subclass of Hybrid Automata, but the encoding itself is obviously different as TPN's and Hybrid Automata interpret time constraints differently.

5.1 2-counter machines

An *input-free 2-counter machine* is a tuple $\mathcal{M} = \langle Q, q_0, q_F, \mathcal{I}, C_1, C_2 \rangle$ where

- Q is a finite set of states,
- $q_0 \in Q$ is the initial state,
- $q_F \in Q$ is the final, or halting, state,
- C_1 and C_2 are counters, each storing a nonnegative integer, initially 0,
- \mathcal{I} is a finite set of instructions, with following forms and meanings ($i \in \{1, 2\}$):
 - (p, dec_i, q) : in state p , decrement C_i by 1 and go to state q ,
 - (p, inc_i, q) : in state p , increment C_i by 1 and go to state q ,
 - $(p, test_i, q, r)$: in state p , test C_i ; if $C_i = 0$ go to q , go to r otherwise.

A *configuration* of \mathcal{M} is a triple (q, x_1, x_2) , where $q \in Q$ and $x_1, x_2 \in \mathbf{N}$ are the values of counters C_1 and C_2 . The initial configuration is $c_0 = (q_0, 0, 0)$.

The halting problem for 2-counter machine is undecidable (Minsky, 1961).

5.2 Encoding 2-counter machines into SwTPN's

Encoding principles and notations: Given two periodic events e_1 and e_2 , of same period ρ , the *phase delay* of e_1 wrt e_2 is the time elapsed between the next occurrence of e_1 and the following occurrence of e_2 . A value of k for counter C_i ($i \in \{1, 2\}$) will be encoded by a phase delay of $\rho/2^{k+1}$ between an event *i.e.*, associated with counter C_i , and a reference event *r.e.* I.e. phase delays $\rho/2, \rho/4, \rho/8, \dots$ will encode values values $0, 1, 2, \dots$. Similar encodings of unbounded discrete spaces into bounded dense spaces have been already used in (Čerāns, 1992) (Henzinger *et al.*, 1995), for similar purposes.

A periodic event of period ρ can be modeled by a TPN constituted of a simple loop as shown in Figure 3 for $\rho = 8$. As such loops store a phase for an event, they will be called *registers*. Given two such registers, say defining events x and y , the phase delay observed at state $s = (m, I)$ of event x wrt y is the difference $I(y) - I(x)$ between the firing intervals (reduced to single points here) associated in s with transitions x and y .



Fig. 3. TPN representing a periodic event

The nets encoding instructions are built from five elementary blocks, described in the sequel. For each block, we prove a local lemma concerning its behavior. In these lemmas, φ_i^s denotes the phase delay between event $i.e$ ($i \in 1..4$) and the reference periodic event $r.e$, observed at state s , and \longrightarrow is the state reachability relation. For any state s , $s \oplus p$ denotes the state obtained from state s by adding a token to place p and the necessary temporal constraints to the interval component of s .

Shared registers and Initialization block: For modeling a 2-counter machine, we need a register for each counter (numbered 1 and 2, defining events $1.e$ and $2.e$), plus one for the reference event (referred to as register r , defining event $r.e$). We also need registers for two temporary events (numbered 3 and 4, defining events $3.e$ and $4.e$). The chosen period ρ is 8, in any unit of time.

As we need to synchronize external transitions with the periodic events, at particular instants in their periods, most registers will hold several transitions in sequence (including the one materializing the event of interest) rather than a single one, with the sum of their delays equal to period 8.

Figure 4 shows the required registers, together with the *init* initialization block (transition *init* and connected edges). Block *init* initializes the phase delays of events $1.e$ and $2.e$ to 4 (making both encode counter value 0); events $3.e$ and $4.e$ are not initialized. Places *in* and *out* (in grey) materialize the context of use of the block.

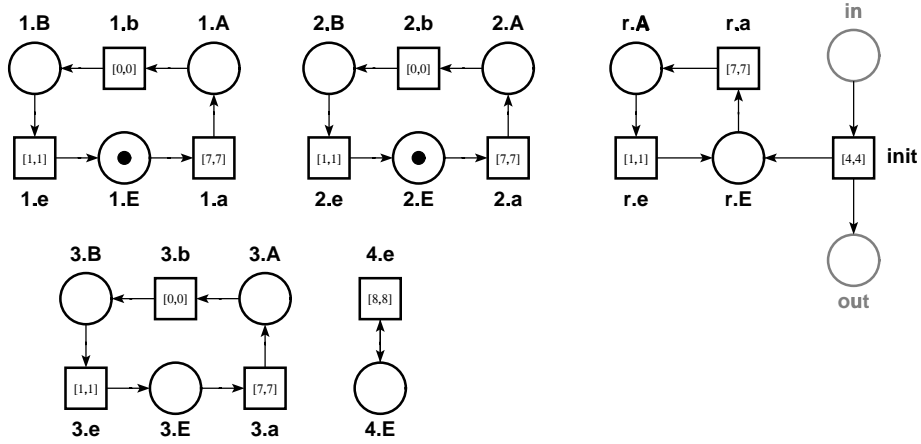


Fig. 4. Shared registers and Initialization block

Lemma 1 (Init lemma).

Let s_0 be the initial state of the registers. Then:

$$(i) (\forall s)(s_0 \oplus in \longrightarrow s \oplus out \Rightarrow \varphi_1^s = 4 \wedge \varphi_2^s = 4)$$

(ii) $(\exists s)(s_0 \oplus in \longrightarrow s \oplus out)$

Proof. (Omitted).

The fact that most registers in Figure 4 hold several transitions, rather than a single one as in Figure 3, makes observation of phase delays φ_i^s slightly more complex than previously explained, transitions $i.e$ and $r.e$ not being enabled at all markings of the registers. Assuming $0 < k \leq 4$, $\varphi_i^s = k$ can be observed as follows:

- $\varphi_i^s = k$ holds iff for some state $s' = (m', I')$:
- (i) $s \xrightarrow{\sigma} s'$, with schedule σ firing only transitions in registers i and r ;
 - (ii) m' marks $i.E$ and $r.E$;
 - (iii) $I'(r.a) - I'(i.a) = k$ (if $i \neq 4$), or $I'(r.a) - I'(4.e) = k$ (otherwise)

The encoding of blocks will ensure that, at the states the phase delays are observed, the registers are free to progress. So, if $\varphi_i^s = k$ holds, then such an s' above will always be reachable from s .

Lemma 1, like all block lemmas to come, expresses both a safety property, relating the phase delay of some event at block exit to that phase delay at block entry, and a liveness property asserting that some firing schedule allows to exit the block. All lemmas could be alternatively proved by model checking the behaviors of the blocks obtained by our experimental implementation of Algorithm 1.

Zero testing instruction: Block test, shown in Figure 5, behaves as a nonintrusive observer for the phase delay between events $i.e$ and $r.e$. As before, grey nodes and edges materialize the context. A token put in place in may propagate to place $outz$ if that phase delay is 4, or to place $outnz$ if that phase delay is not larger than 2. Note that no stopwatch arc is used.

Lemma 2 (Test lemma).

Let s be the current state of registers and assume $0 < \varphi_i^s \leq 4$. Then:

- (i) $\varphi_i^s \leq 2 \Rightarrow (\forall s')(s \oplus in \longrightarrow s' \oplus outnz \Rightarrow \varphi_i^{s'} = \varphi_i^s)$
- (ii) $\varphi_i^s \leq 2 \Leftrightarrow (\exists s')(s \oplus in \longrightarrow s' \oplus outnz)$
- (iii) $\varphi_i^s = 4 \Rightarrow (\forall s')(s \oplus in \longrightarrow s' \oplus outz \Rightarrow \varphi_i^{s'} = \varphi_i^s)$
- (iv) $\varphi_i^s = 4 \Leftrightarrow (\exists s')(s \oplus in \longrightarrow s' \oplus outz)$

Proof. Assume a token is put into place in and $\varphi_i^s = k$, with $0 < k \leq 4$. Then:

- When transition $test$ fires, it does so exactly 1 unit of time before $i.e$.
- Since $k \leq 4$, place $r.A$ in register r is always empty when $test$ fires. So, after $test$ fired, ref fires exactly 1 unit of time before the next $r.e$ event.
- From the above, we have that ref fires exactly k units of time after $test$ fired. So, when ref fires, either $k \leq 2$, sup did not fire yet, and only $nonZero$ can fire, or $k = 4$, sup fired, and only $zero$ can fire.
- Finally, firing $test$ and ref do not change the phase of any event. \square

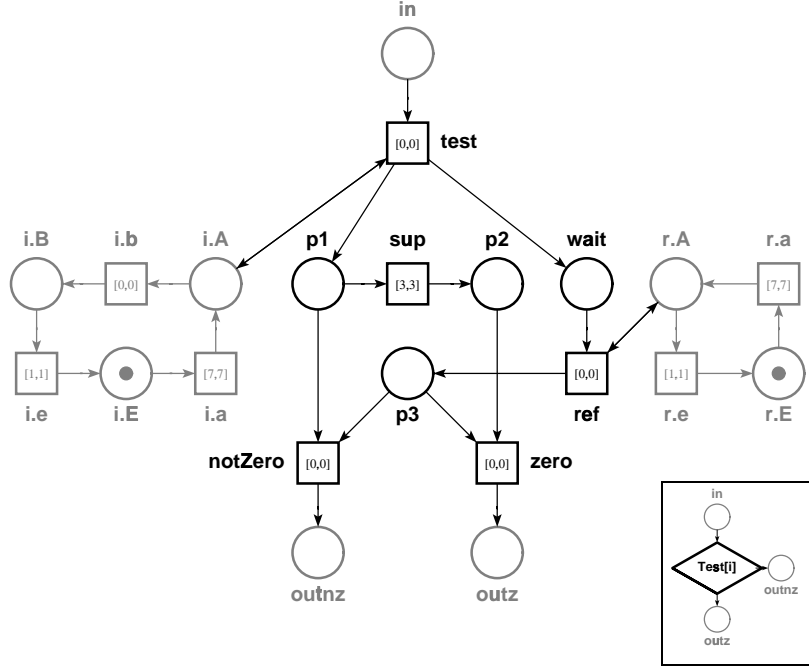


Fig. 5. Zero testing instruction for register i , and pictogram

Note that firing transition $test$ can be delayed an arbitrary amount of time, nondeterministically, but that, once $test$ fired, transition ref will necessarily fire k units of time later. Most of the forthcoming blocks use the same mechanism for entering blocks and observing phase delay ϕ_i^s .

Doubler block: Shown in Figure 6, this doubles the phase delay of $i.e$, assuming it not larger than 2. The arc from place $p4$ to $i.a$ is a stopwatch arc. When executed, the block suspends transition $i.a$ a full period less the phase delay between $i.e$ and $r.e$. Thus, that phase delay has doubled.

Lemma 3 (Doubler lemma).

Let s be the current state of registers and doubler block (assuming place $p4$ marked from the start), and assume $0 < \varphi_i^s \leq 2$. Then:

- (i) $(\forall s')(s \oplus in \longrightarrow s' \oplus out \Rightarrow \varphi_i^{s'} = 2 * \varphi_i^s)$
- (ii) $(\exists s')(s \oplus in \longrightarrow s' \oplus out)$

Proof. Assume place in is marked and $\varphi_i^s = k$, with $0 < k \leq 2$.

– Using the same arguments than in the proof of Lemma 2, transition ref , when it fires, does so exactly k unit of times after $double$. Also, firing these transitions do not change the phase of events $i.e$ and $r.e$.

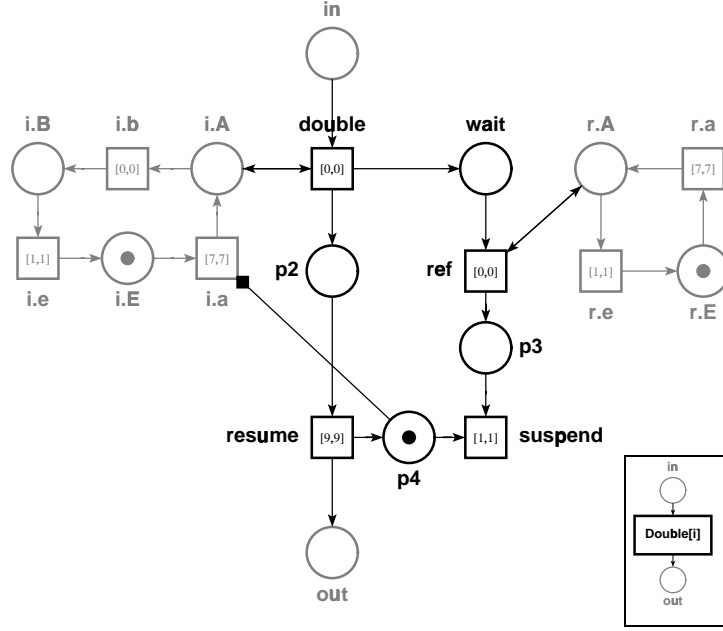


Fig. 6. Doubler block for register i , and pictogram

– From the above, assuming $double$ fired at time t , transition $i.a$ will become enabled at time $t + 1$, suspended at time $t + k + 1$ (when $suspend$ fires), and then resumed at time $t + 9$ (when $resume$ fires). So, event $i.e$ will occur next at time $t + 17 - k$, and the following event $r.e$ at $t + k + 17$, establishing $\varphi_i^{s'} = 2 * k$. \square

Selection block: This block initializes registers 3 and 4, giving events 3.e and 4.e the same phase delay wrt $r.e$, in interval $]0, 2]$. It is represented Figure 7.

Lemma 4 (Selection lemma).

Let s be the current state of registers. Then:

- (i) $(\forall s')(s \oplus in \longrightarrow s' \oplus out \Rightarrow \varphi_4^{s'} = \varphi_3^{s'} \wedge 0 < \varphi_3^{s'} \leq 2 \wedge \varphi_i^{s'} = \varphi_i^s)$
- (ii) $(\exists s')(s \oplus in \longrightarrow s' \oplus out)$

Proof.

– Using the same arguments than in the proof of Lemma 2, transition ref , when it fires, does so exactly 1 unit of times before event $r.e$, and firing $choose$ and ref do not change the phase of any register.

– Next, transition set fires between 7 (included) and 9 (excluded) units of time after ref , establishing a phase delay for 3.e and 4.e wrt $r.e$ in interval $]0, 2]$. \square

Compare and assign block: Shown in Figure 8, this block compares the phases of events 3.e and $i.e$ for equality, and, if comparison is successful, makes the phase of event $i.e$ equal to that 4.e had when entering the block.

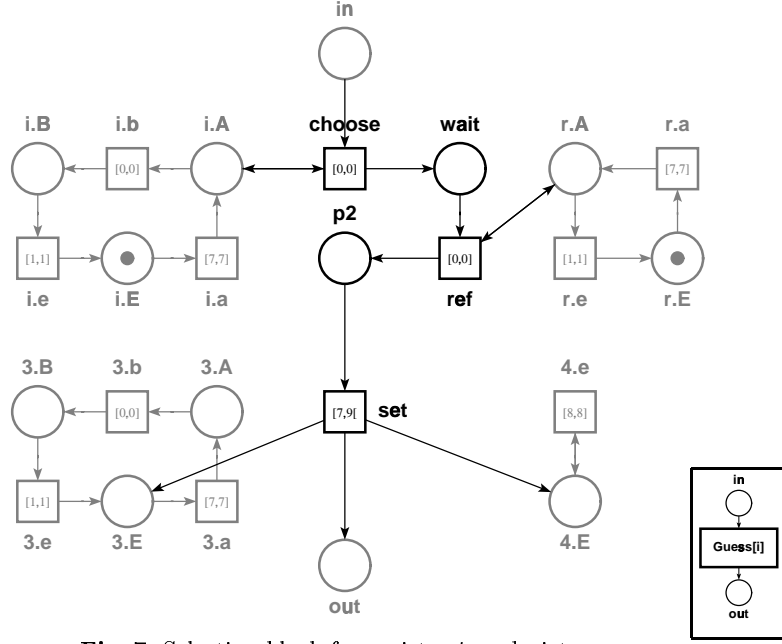


Fig. 7. Selection block for register i , and pictogram

Lemma 5 (Compare and assign lemma).

Let s be the current state of registers and assume $0 < \varphi_{i,3}^s \leq 4$ and $0 < \varphi_4^s < \varphi_3^s$. Then:

- (i) $\varphi_3^s = \varphi_i^s \Rightarrow (\forall s')(s \oplus in \longrightarrow s' \oplus out \Rightarrow \varphi_i^{s'} = \varphi_4^s)$
- (ii) $\varphi_3^s = \varphi_i^s \Leftrightarrow (\exists s')(s \oplus in \longrightarrow s' \oplus out)$

Proof.

- Transition *equal* may only fire if places $i.A$ and $3.A$ became marked at the same time, that is the block is entered only if $i.e$ and $3.e$ have equal phases.
- Next, assuming $\varphi_4^s < \varphi_3^s$, $4.e$ fires, simultaneously restarting transition *assign*. Then, after a full period (8), *assign* fires, giving $i.e$ the phase $4.e$ had when entering the block.
- If $3.e$ and $i.e$ had different phases on entry, then no state reachable from $s \oplus in$ marks *out*. \square

Decrement instruction: The decrement instruction, Figure 9 (left), is implemented by a copy of the doubler block preceded by a copy of the test instruction. The test instruction prevents doubling the phase delay of register i if larger than 2 (that is to decrement the counter i represents if it has value 0).

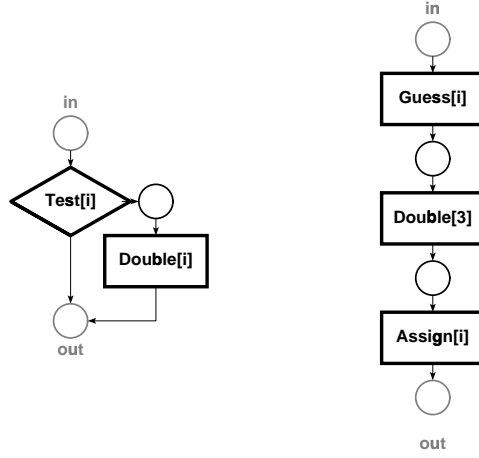


Fig. 9. Decrement (left) and Increment (right) blocks for register i

Encoding a machine: The SwTPN \mathcal{N} encoding machine \mathcal{M} is built as follows:

- If \mathcal{M} has $n + 1$ states q_0, q_1, \dots, q_n , then create \mathcal{N} with $n + 2$ places named $start, p_0, p_1, \dots, p_n$. Place $start$ is the sole marked.
- q_0 being the initial state of the machine, add to \mathcal{N} the shared registers, and block $Init$ connected to places $start$ (as input) and p_0 (as output).
- For each instruction (q_a, dec_i, q_b) (resp. (q_a, inc_i, q_b)) add to \mathcal{N} a copy of the *Decrement* (resp. *Increment*) block for register i , connected to places p_a (as input) and p_b (as output).
- For each instruction $(q_a, test_i, q_b, q_c)$, add to \mathcal{N} a copy of the *Test* block for register i , connected to places p_a (as input), p_b (as *outz* output), and p_c (as *outnz* output).

5.3 Undecidability results

Let \mathcal{N} , with initial state s_0 , be the SwTPN encoding some 2-counter machine \mathcal{M} , obtained as explained above.

For any configuration $c = (q_i, x_1, x_2)$ of \mathcal{M} , and state s of \mathcal{N} , let us write $c \cong s$ iff s marks place p_i , and the phase delays of events $1.e$ and $2.e$ wrt $r.e$ at s encode the counter values x_1 and x_2 , respectively; that is $\varphi_1^s = 1/2^{x_1+1}$ and $\varphi_2^s = 1/2^{x_2+1}$.

Theorem 2. *A configuration c is reachable from c_0 in machine \mathcal{M} (written $c_0 \rightarrow_{\mathcal{M}} c$) iff some state s is reachable from s_0 in net \mathcal{N} (written $s_0 \rightarrow_{\mathcal{N}} s$) such that $s \cong c$. That is:*

$$(\forall c)(c_0 \rightarrow_{\mathcal{M}} c \Leftrightarrow (\exists s)(s_0 \rightarrow_{\mathcal{N}} s \wedge s \cong c))$$

Proof. By induction on the length of sequences of machine instructions, using Lemmas 1, 2, 6, and 7. \square

Theorem 3. *State reachability, marking reachability, k -boundedness, and liveness (cf. Section 2.2) are undecidable for bounded Stopwatch TPN's.*

Proof.

Net \mathcal{N} is bounded, it is even safe (1-bounded): It is easily checked that all blocks are safe when a single token is put in their input place (for the doubler block, notice that *suspend* always fires before *resume*).

Then, by Theorem 2, we have that some configuration c holding the final state q_F is reachable in \mathcal{M} , i.e. \mathcal{M} halts, iff some state $s \cong c$ is reachable in \mathcal{N} . Hence state reachability is undecidable for safe SwTPN's. Undecidability of the other properties follows. \square

Since instructions of the machine are not executed concurrently, and phase values can be copied from a register into another (as done in the “compare and assign” block), any 2-counter machine can be encoded into a SwTPN that uses only a single copy of the doubler block and a single copy of the selection block (details omitted), that is into a SwTPN with a single stopwatch arc and a single transition with static interval not reduced to a point. Indeed, if no stopwatch arc was present, or if all intervals were punctual, then state reachability is decidable, since it is decidable for bounded TPN's (Section 2.2), and bounded TPN's or SwTPN's in which all static intervals are punctual have finite state spaces (this directly follows from the definition of states).

Concerning other extensions of TPN's modeling preemption, all the blocks described in this Section, including the doubler block in Figure 6, are easily encoded into Scheduling-TPN's (Roux *et al.*, 2002), Preemptive-TPN's (Bucci *et al.*, 2004) or Inhibitor Hyperarc TPN's (Roux *et al.*, 2004). It follows that Theorem 3 applies to these models as well, solving an open problem.

6 Approximate state spaces

Theorem 3 definitely ends any hope to compute exact finite abstract state spaces for all bounded SwTPN's. In general, one will be bound to compute overapproximations of those spaces, capturing a behavior including the exact behavior of the net, but possibly larger. Such overapproximations provide sufficient conditions for safety properties.

We discuss in this Section two overapproximation techniques: the “tightest enclosing DBM” technique of (Bucci *et al.*, 2004), (Lime *et al.*, 2003) and a technique we introduce called “polyhedra quantization”. A number of other approximations techniques have been proposed for linear hybrid automata (Alur *et al.*, 1995), but these are unnecessarily rich for our purposes.

6.1 Tightest enclosing DBM

This overapproximation method has been proposed for Preemptive-TPN's (Bucci *et al.*, 2004), Scheduling-TPN's (Lime *et al.*, 2003) and IHTPN's (Roux *et al.*, 2004). Adapted to SwTPN's, the approximated state class graph by the tightest enclosing DBM method is built as follows, technical details and proofs can be found in the references mentioned:

Algorithm 2 (Approximation by tightest enclosing DBM)

As Algorithm 1, except that, after step 3, domain D' is approximated by the smallest domain including D' that can be described by a DBM.

Because the approximated polyhedra relax time constraints, and that relaxing time constraints in some bounded TPN's may make them unbounded, not all bounded TPN's admit finite approximated state class graphs by the tightest enclosing DBM method. However, the property holds for an important subclass of bounded TPN's, introduced in the sequel.

Let us say that a SwTPN is *inherently bounded* when its underlying Petri net (the net obtained by removing the stopwatch arcs and time constraints) is bounded. We have:

Theorem 4. *For any inherently bounded SwTPN, Algorithm 2 terminates.*

Preserving the DBM shape for polyhedra makes computation and comparison of classes in Algorithm 2 undoubtedly faster than in Algorithm 1. Now, it may happen that the computed approximations are too coarse for preserving some properties of interest, or much larger than the exact state class graph, when finite.

For instance, we have seen in the example discussed in Section 4 that the best estimation one can infer using Algorithm 2 for the WCET of task 3 using observers is 144, while its exact value of 96 can be checked on the (exact) state class graph computed by Algorithm 1.

It could also be observed that, widening the interval of transition t_{32} to $[20, 38]$, instead of $[20, 28]$ produces with Algorithm 2 an approximation of the behavior of the net with more than 12000 classes, much larger and more expensive to compute than the exact graph obtained with Algorithm 1, that has only 3641 classes in this case. Further, widening this interval to $[20, 39]$, the tightest enclosing DBM approximated graph becomes unbounded (the net is not inherently bounded) while Algorithm 1 still produces a finite graph. Using Algorithm 1, the behavior computed becomes unbounded from the larger interval value $[20, 49]$.

6.2 Polyhedra quantization

As shown above, computing the exact behavior of a SwTPN may be cheaper in some cases than approximating it by the tightest enclosing DBM, in general. And, in any case, there is a need for more accurate approximations. Now, the

undecidability results of Section 5.3 imply that termination of Algorithm 1 is not guaranteed, even on inherently bounded SwTPN's. We propose in the sequel an overapproximation technique that operates on the general polyhedra computed by Algorithm 1 and ensures its termination whenever applied to an inherently bounded SwTPN.

The technique is based on quantization of the polyhedra captured in state classes, according to a discretization of space. Polyhedra are still computed assuming a dense time though, as discretizing firing times would not in general produce overapproximations. The precision of approximations may be adjusted, so that approximated state class graphs can be computed as close as desired to the exact graph. When the precision is chosen “high enough” and Algorithm 1 produces a finite graph, then the result coincides with that computed by Algorithm 1, otherwise we obtain an overapproximation.

Let us first recall Motzkin's decomposition theorem for polyhedra (see e.g. (Schrijver, 1986)): any polyhedron P can be uniquely decomposed into a polytope (a bounded polyhedron, generated by convex combination of the extreme vertices of P , in finite number) and a polyhedral cone. In our case, since all polyhedra lie in the nonnegative quadrant, the cones are pointed cones (generated by positive linear combinations of the extreme rays of P , in finite number).

Our approximation method will take advantage of two properties of the polyhedra computed by Algorithm 1:

Theorem 5. *For any SwTPN \mathcal{N} :*

- (i) *There is an integer b such that all coordinates of all extreme vertices of the polyhedra computed by Algorithm 1 for \mathcal{N} are smaller than b ;*
- (ii) *All extreme rays, if any, of all polyhedra computed by Algorithm 1, belong to the canonical base of R^n .*

Proof. By induction. (i) and (ii) hold for the initial state class and are preserved by class derivations in Algorithm 1. For (i), b is any integer larger than the largest finite endpoint among those of the static intervals of transitions. Any extreme vertex is a finite endpoint of the firing interval of some state, and, from the definition of states, these may only move towards 0. (ii) means that, above some finite threshold, unbounded variables are always unrelated. The only step in Algorithm 1 that could introduce oblique rays is (1), but these disappear after the projection step (3). \square

The polyhedra characterizing temporal information in classes (as computed by Algorithm 1) will be approximated as follows:

Definition 5 (Polyhedra approximations). *Assume given some $k > 0$ ($k \in \mathbf{Q}$). Suppose \mathbf{R}_+^n discretized into hypercubes of side k , and let \mathcal{H}_k denote the set of these hypercubes. Let $P \subseteq \mathbf{R}_+^n$ be some polyhedron. Then, relative to k :*

- *A point $x \in \mathbf{R}_+^n$ is approximated by the intersection of all hypercubes of \mathcal{H}_k containing x .*

- A polytope $Q \subseteq \mathbf{R}_+^n$ is approximated by the convex hull of the approximations of its extreme vertices.
- Polyhedron P is approximated by the polyhedron $h_k(P)$ built from the cone component of P and the approximation of the polytope component of P .

Clearly, for any P and k , $h_k(P)$ contains P . Also, by adjusting the grid size k , $h_k(P)$ can be chosen as close to P as desired. Indeed, when P includes its boundary, then there is always some k such that $h_k(P) = P$.

The approximated state class graph for grid size k is built as follows:

Algorithm 3 (Approximation by quantization with grid size k)

As Algorithm 1, except D' is approximated by $h_k(D')$ after step 3.

Since each approximated state class contains (possibly strictly) some exact state class, and any schedule firable from some class is also firable from any larger class, the support of any schedule firable in the net appears as a path in the approximated state class graph built by Algorithm 3. But, since approximations relax time constraints, the converse is not necessarily true: Some paths in the approximated state class graph may not be the support of some firable schedule. Applied to a SwTPN or TPN, Algorithm 3 yields an overapproximation of its behavior.

For the same reasons than Algorithm 2, Algorithm 3 does not necessary terminate on all bounded TPN's or SwTPN's, but we have similarly:

Theorem 6. *For any grid size k and any inherently bounded SwTPN, Algorithm 3 terminates.*

Proof. Approximated polyhedra obey the conditions in Theorem 5 too. Next, the number of hypercubes of side k in any bounded subspace of \mathbf{R}^n is finite, so the number of candidate extreme vertices for approximated polyhedra is finite too, and only finitely many polyhedra can be built from these and a finite set of rays. Finally, since the net is inherently bounded, relaxing time constraints preserves its boundedness property. \square

Since any polyhedron can be approximated as closely as desired by adjusting the grid size k , the exact graph of state classes may be approximated as closely as desired, but there is not in general a bound on k such that the approximated graph coincides with the exact one (this would contradict Theorem 3).

Algorithm 3 has been implemented. Approximations only require polyhedra operations typically provided by available polyhedral libraries.

6.3 Discussion

Given some grid size k , it cannot be said in general that approximations by quantization with grid size k is more accurate than approximations by the tightest enclosing DBM method. However, because the “tightest enclosing DBM” approximation cannot approximate some polyhedra as closely as the “quantization” approximation, there is always some grid size k such that this is true. To

illustrate their differences, consider the polyhedron represented Figure 10(a), defined by $\{0 \leq x \leq 2, 0 \leq y \leq 2, y - x \leq 1, x + y \leq 2\}$. Polyhedron (b) is the polyhedron obtained from (a) by the tightest enclosing DBM method. Polyhedra (c) and (d) are those obtained by the quantization method, with grid sizes $k = 1$ and $k = 1/2$, respectively. Approximations (c) and (b) are uncomparable, but (d) is more accurate than (b); approximation (d) coincides in fact with the exact polyhedron (a).

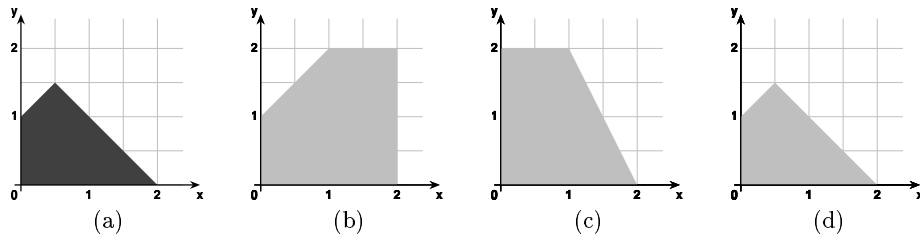


Fig. 10. Exact polyhedra and overapproximations

The first purpose of the quantization approximation method is to allow one to compute more accurate approximations of the state class graph of a SwTPN than computed by the tightest enclosing DBM method, when needed.

As shown above, one can always compute for any polyhedron P the largest grid size k such that the quantized polyhedron is equal or smaller than the tightest DBM polyhedron including P . Consequently, one can always compute the largest grid size k such that Algorithm 3 with grid size k yields an equally or more accurate approximation of the state class graph of a net than that computed by Algorithm 2 (g' is an equally or more accurate approximation than g'' if, for any firing sequence σ of g' , the firing domain obtained by σ in g' is included in that obtained by σ in g'').

In particular, when the state class graph of a SwTPN is finite, Algorithm 2 does not necessarily compute this exact graph, while Algorithm 3 does it for a “sufficiently small” grid size k .

For the example in Figure 2, for instance, the exact behavior is found with grid size 1. For the example in Figure 1, that admits an infinite state class graph, the table below shows the sizes of the approximated state class graphs computed by Algorithm 3 for various grid sizes and for two methods for comparing classes: using equality of solution sets (\equiv , as in Algorithm 1), and its variant using inclusion of solution sets (\sqsubseteq) instead, briefly discussed in Section 3.2. Because this net is not inherently bounded, the approximated behavior becomes unbounded when the grid size is taken larger than 1.

grid size	classes/transitions (\equiv rule)	classes/transitions (\sqsubseteq rule)
≥ 2	unbounded	unbounded
1	57/137	12/32
1/4	920/2060	18/47
1/16	29704/64436	18/47

Finally, it is worth to note that the “quantization” method is applicable to TPN’s too (without stopwatch arcs), taking grid sizes larger than 1 if static interval endpoints are integers.

7 Conclusion

This paper first introduces a simple extension of Time Petri nets with “stopwatch” arcs, able to express suspension and resumption of transitions upon conditions only depending on markings. The model can be seen as a simplification of the IHTPN’s of (Roux *et al.*, 2004). The features of IHTPN’s (inhibitor hyperarcs) could be safely merged with those of SwTPN’s to constitute an expressively rich modeling tool for a wide range of problems involving preemption.

The main result of the paper is the proof that state reachability is undecidable for SwTPN’s, even when bounded. This result implies in turn undecidability of marking reachability, k -boundedness, and liveness for SwTPN’s and all above mentioned extensions of TPN’s, even when nets are bounded. These problems remained open so far.

A semi-algorithm was presented for computing exact representations for the state spaces of SwTPN’s, and an implementation was reported, embedded into an extension of the tool TINA (Berthomieu *et al.*, 2004). Since this paper was written, Algorithm 1 has been implemented for the Scheduling TPN’s model too, integrated into the ROMEO tool (Gardey *et al.*, 2005). Experiments with the implementations suggest that exact state space computation terminates on many practical applications.

Finally, we proposed an original method for computing finite overapproximations of the state class graphs for a class of bounded Stopwatch Time Petri nets, based on quantization of the polyhedra capturing the temporal information. It yields more accurate approximations than available methods, and its precision can be parameterized. It is also applicable to Time Petri nets.

Though conceptually close, the proposed techniques are computationally more demanding than the similar techniques used for TPN’s. Prospective work addresses algorithmic aspects for their efficient implementation. A source of inefficiency in implementations of Algorithm 3 is that classes are computed from polyhedra in “constraints” forms, while approximations are computed on their dual “extreme vertices and rays” forms. The change from a representation form to the other, or maintaining the double representation, is expensive in the general case. We hope to take advantage of the specific properties of our polyhedra (cf. Theorem 5) to speed up representation changes and comparison of classes.

References

- Alur R., Courcoubetis C., Halbwachs N., Henzinger T., Ho P.-H., Nicollin X., Olivero A., Sifakis J., Yovine S., « The Algorithmic Analysis of Hybrid Systems », *Theoretical Computer Science*, vol. 138, p. 3-34, 1995.
- Berthomieu B., « La méthode des classes d'états pour l'analyse des réseaux Temporels – Mise en œuvre, Extension à la multi-sensibilisation », *Proc. Modélisation des Systèmes Réactifs, Toulouse, France*, October, 2001.
- Berthomieu B., Diaz M., « Modeling and Verification of Time Dependent Systems Using Time Petri Nets. », *IEEE Trans. on Software Engineering*, vol. 17, n° 3, p. 259-273, March, 1991.
- Berthomieu B., Menasche M., « An Enumerative Approach for Analyzing Time Petri Nets. », *IFIP Congress Series*, vol. 9, p. 41-46, 1983.
- Berthomieu B., Ribet P.-O., Vernadat F., « The tool TINA – Construction of Abstract State Spaces for Petri Nets and Time Petri Nets », *International Journal of Production Research*, vol. 42, n° 14, p. 2741-2756, 15 July, 2004.
- Berthomieu B., Vernadat F., « State Class Constructions for Branching Analysis of Time Petri Nets », *Proc. Tools and Algorithms for the Construction and Analysis of Systems, Springer LNCS 2619*, 2003.
- Boucheneb H., Hadjidj R., « Towards optimal CTL^* model checking of Time Petri nets », *Proceedings of 7th Workshop on Discrete Events Systems*, Reims, France, September, 2004.
- Bucci G., Fedeli A., Sassoli L., Vicario E., « Time state space analysis of real-time preemptive systems », *IEEE Trans. on Software Engineering*, vol. 30, n° 2, p. 97-111, February, 2004.
- Cassez F., Larsen K. G., « The Impressive Power of Stopwatches », *11th Int. Conf. on Concurrency Theory, University Park, P.A., USA*, Springer LNCS 1877, p. 138-152, 2000.
- Čerāns K., *Algorithmic problems in analysis of real time system specifications*, University of Latvia, Dr.sc.comp. Thesis, 1992.
- Daws C., Tripakis S., « Model checking of real-time reachability properties using abstractions », *Proc. Tools and Algorithms for the Construction and Analysis of Systems (TACAS'1998), Springer LNCS 1384*, 1998.
- Gardey G., Lime D., Magnin M., Roux O. H., « Roméo: A Tool for Analyzing time Petri nets », *17th International Conference on Computer Aided Verification, CAV'05, Springer LNCS 3576*, jul, 2005.
- Henzinger T. A., Kopke P. W., Puri A., Varaiya P., « What's decidable about hybrid automata? », *Proc. of the 27th Annual Symp. on Theory of Computing, ACM Press*, p. 373-382, 1995.
- Jeannet B., The Polka Convex Polyhedra library, Edition 2.0.1, <http://www.irisa.fr/prive/bjeannet/newpolka.html>, IRISA, Rennes, 2002.
- Jones N. D., Landweber L. H., Lien Y. E., « Complexity of Some Problems in Petri Nets. », *Theoretical Computer Science* 4p. 277-299, 1977.
- Lime D., Roux O. H., « Expressiveness and analysis of scheduling extended time Petri nets », *5th IFAC International Conference on Fieldbus Systems and their Applications, (FET'03)*, Elsevier Science, July, 2003.
- Merlin P. M., *A Study of the Recoverability of Computing Systems.*, PhD Thesis, Irvive, 1974.
- Minsky M., « Recursive Unsolvability of Post's problem », *Ann. of Math.*, vol. 74, p. 437-454, 1961.

- Roux O. H., Déplanche A.-M., « A t-time Petri net extension for real time task scheduling modeling », *Eur. Journal of Automation (JESA)*, 2002.
- Roux O. H., Lime D., « Time Petri Nets with Inhibitor Hyperarcs. Formal Semantics and State Space Computation », *Proc. Int. Conf. on Applications and Theory of Petri Nets (ICATPN'04)*, Bologna, Italy, 2004.
- Schrijver A., *Theory of Linear and Integer Programming*, John Wiley and Sons, NY, 1986.