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

Bernard Berthomieu[1], Didier Lime[2,3], Olivier H. Roux[3], and François Vernadat[1]

[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] Aalborg University, Fredrik Bajers Vej 7B, 9220 Aalborg East, Denmark
tel: +45.96.35.72.23, fax: +45.98.15.98.89
didier@cs.aau.dk

[3] 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
Olivier-h.Roux@irccyn.ec-nantes.fr

Abstract. Several extensions of Time Petri nets have been proposed for modeling suspension and resumption of actions in timed systems. Using a simple class of TPN's extended with stopwatches (SwTPN's), we first prove that state reachability in all these models is undecidable, even when bounded. A semi-algorithm is then proposed for building exact representations of the behavior of SwTPN's, based on the known state class method for Time Petri nets. Next, we discuss overapproximation methods ensuring termination of the construction on a subclass of bounded SwTPN's, and propose 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.

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 [CL00], a problem known undecidable for LHA's [ACH+95]. As no decidable subclass of SWA's has been identified that preserve these modeling capabilities, 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) [Mer74] are another widely used model for real-time systems. Time Petri nets extend Petri nets with temporal intervals associated

with transitions. State space abstractions for TPN's preserving various classes of properties can be computed, in terms of so called state classes [BM83] [BD91] [BV03]. 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 modeling of suspension and resumption of actions, including Scheduling-TPN's [RD02] [LR03], Preemptive-TPN's [BFSV04], and Inhibitor Hyperarc TPN's (IHTPN's) [RL04]. 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 [RL04].

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 [BRV04].

The paper is organized as follows: Section 2 introduces Stopwatch Time Petri nets and a semi-algorithm for computing their state classes. An example is presented in Section 3. Undecidability of state reachability for these nets is established in Section 4. Section 5 explains the polyhedra quantization 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 Stopwatch Time Petri nets

2.1 SwTPN's, 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 any) or ∞. For any $\theta \in \mathbf{R}^+$, let $i \dot{-} \theta$ denote the interval $\{x - \theta | x \in i \wedge x \geq \theta\}$.

Definition 1. *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* $\mathbf{Sw} : T \to P \to \mathbf{N}$ *is a function called the* stopwatch incidence function.

Time Petri nets extend Petri nets by $I_s : T \to \mathbf{I}^+$, called the *Static Interval* function. Function $\mathbf{Sw}$ 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, with diamond 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.

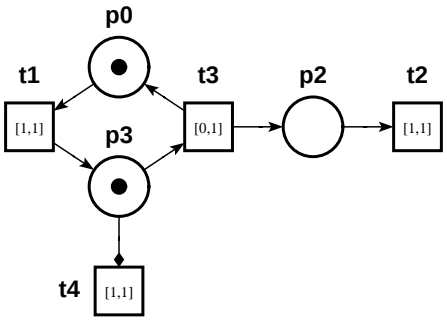

Fig. 1. A Stopwatch Time Petri net

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 2. *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. $(\forall k \in T)(m' \geq \mathbf{Pre}(k) \Rightarrow$
   $I'(k) =$ **if** $k \neq t \ \wedge \ m - \mathbf{Pre}(t) \geq \mathbf{Pre}(k)$
   then if $m \geq \mathbf{Sw}(k)$ **then** $I(k) \dot{-} \theta$ **else** $I(k)$
   else $I_s(k)$

We have $s \xrightarrow{t@\theta} s'$ if firing t from s at (relative) time θ leads to s'. (1) ensures that transitions fire in their temporal interval, unless disabled by firing another transition, and that t is active. (2) is the standard marking transformation. From (3), newly enabled transitions are assigned their static intervals, while transitions persistent wrt t either have their firing intervals unchanged (if suspended), or have it shifted by θ and truncated to nonnegative times (if active). Transitions that remain enabled during their own firing are considered newly enabled.

The *state graph* of a SwTPN is the set of states reachable from its initial state $s_0 = (m_0, I_0)$, where $I_0(t) = I_s(t)$, equipped with relation $\xrightarrow{t@\theta}$. A *firing schedule* is a sequence of successively firable transitions, its *support*, together with their relative firing times.

2.2 State classes for SwTPN

As for TPN's, states of a SwTPN typically have an infinity of successors. The state class graph constructions that provide state space abstractions for TPN's [BM83] [BV03] are easily adapted to SwTPN's. We adapt below the construction of [BM83] here, that preserves markings and LTL properties of the state space.

State classes are denoted by pairs (m, D), where m is a marking and D is a firing domain, described by an inequality system $A\underline{\phi} \leq \underline{b}$. Variable ϕ_i describes the times at which the i^{th} transition enabled at m may fire. Let us write $(m, D = \{A\underline{\phi} \leq \underline{b}\}) \cong (m', D' = \{A'\underline{\phi} \leq \underline{b'}\})$ 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 pair 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 $\cong$ to some pair in C. There is an arc labeled t between classes C_σ and c iff $c \cong C_{\sigma.t}$.

- *The initial pair is* $C_\epsilon = (m_0, \{Eft_s(t) \leq \underline{\phi}_t \leq Lft_s(t) \mid \mathbf{Pre}(t) \leq m_0\})$
- *If σ is firable and* $C_\sigma = (m, D = \{A\underline{\phi} \leq \underline{b}\})$, *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 $D \cup \{\underline{\phi}_t \leq \underline{\phi}_i \mid i \neq t \wedge m \geq \mathbf{Pre}(i) \wedge m \geq \mathbf{Sw}(i)\}$ *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 $\underline{\phi'_k}$ is introduced, obeying:*
     $\underline{\phi'_k} = \underline{\phi}_k - \underline{\phi}_t$ *if* $k \neq t$, $m - \mathbf{Pre}(t) \geq \mathbf{Pre}(k)$, *and* $m \geq \mathbf{Sw}(k)$
     $\underline{\phi'_k} = \underline{\phi}_k$ *if* $k \neq t$, $m - \mathbf{Pre}(t) \geq \mathbf{Pre}(k)$, *and* $\neg(m \geq \mathbf{Sw}(k))$
     $\underline{\phi}_k \in I_s(k)$ *otherwise*
  3. *Variables $\underline{\phi}$ are eliminated.*

For TPN's (without stopwatch arcs), the set of distinct systems D one can build by Algorithm 1 is finite [BM83], whether the TPN is bounded or not. So, bounded TPN's admit finite state class graphs. Further, systems D are difference systems, for which canonical forms can be computed efficiently.

Unfortunately, these properties do not hold if stopwatch arcs are present. Describing firing domains requires richer inequality systems, and boundedness of the SwTPN does not imply finiteness of its set of state classes.

As an example, consider the net in Figure 1. By simple temporal arguments, it can be shown that this net is bounded. In this net, schedules of support $t_3.t_1.(t_3.t_2.t_1)^n.t_2.t_4$, 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. Firing σ_n leads to the following class (the corresponding clock space is also given):

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

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

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 [BV03] 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 TA's in [DT98] or TPN's in [BH04]. These alternatives only preserve markings or states (not LTL) but typically produce smaller graphs.

3 A simple example

Though similar algorithms are available for all TPN extensions referred to, none have been implemented so far. We report in this Section some experiments with an implementation (available at `http://www.laas.fr/tina/next`) of Algorithm 1 for SwTPN's embedded in an extension of the *Tina* tool [BRV04]. For polyhedral operations, the implementation relies on the *NewPolka* library [Jea02].

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 [BFSV04]. 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).

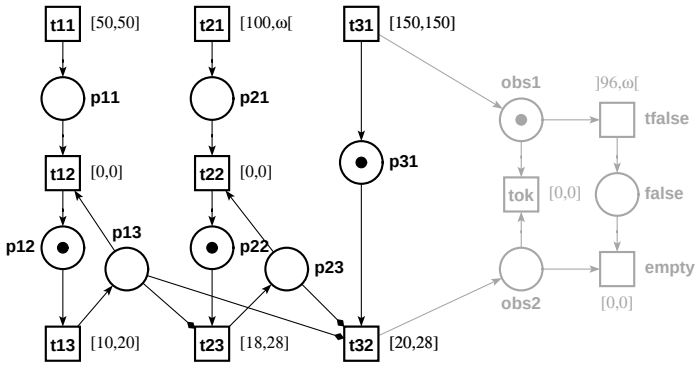

Fig. 2. SwTPN of two periodic and a sporadic processes and an observer

Schedulability is satisfied if the net is safe. The DBM overapproximated state space of [RD02,BFSV04] yields a graph of 608 classes, all with safe markings. To check if some run is indeed possible, [BFSV04] 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. Specific methods for checking quantitative timed properties are also proposed.

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 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 space computed for the extended net confirms that the property is true, but it becomes false if the firing interval of `tfalse` includes 96. The WCRT of this task is thus 96 time units.

Reasoning only on the DBM approximated graph, we would obtain a WCRT equal to 144 time units. Finally, if the execution time of task 3 (`t32`) is increased to [20,35], then the DBM approximated state class graph becomes unbounded, while the exact state class graph is still finite. In this case the methods of [BFSV04] cannot be applied.

4 Decidability issues for Stopwatch Time Petri nets

The marking reachability, state reachability, and boundedness problems are known undecidable for TPN's [JLL77]. It follows that these problems are also undecidable for SwTPN's, in general.

Though undecidable, there are a number of convenient sufficient conditions for the boundedness property, applying to TPN's and SwTPN's. One, for instance, is that the underlying Petri net is bounded, which is known decidable. Considering bounded nets only, marking reachability for TPN's is decidable using

the state class graph of [BM83], and state reachability and liveness are decidable using the alternative constructions of [BV03]. The question addressed in this Section is whether these problems are decidable for bounded SwTPN's.

The answers are negative, unfortunately. It is proven in the sequel that state reachability for SwTPN's can be reduced to the halting problem for 2-counter machines. After recalling the structure of such machines, an encoding into SwTPN's is developed, and the undecidability result derived. The ideas underlying this encoding are similar to those used in [HKPV95] for encoding 2-counter machines into subclasses of hybrid automata, but the encoding itself is obviously different as SwTPN's do not handle clocks explicitly.

4.1 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 [Min61].

4.2 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 an 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 $i.clock$, associated with counter C_i, and a reference event $tick$ (i.e. delays $\rho/2, \rho/4, \rho/8, \ldots$ encode values values $0, 1, 2, \ldots$). Similar encodings of unbounded discrete spaces into bounded dense spaces have been already used in [Čer92] [HKPV95].

Observed when some place p becomes marked, the phase delay between $i.clock$ and $tick$ will be denoted ϕ_i^p The nets encoding instructions are built from five elementary blocks, described in the sequel. Most proofs are omitted, but many simply paraphrase the behavior of blocks and are easily derived.

Shared registers and Initialization block: Periodic events of period ρ can be modeled by a loop with a transition tagged with interval $[\rho, \rho]$. As these loops store a phase for an event, they will be called "registers". As we need to synchronize external transitions with some periodic events, at particular instants

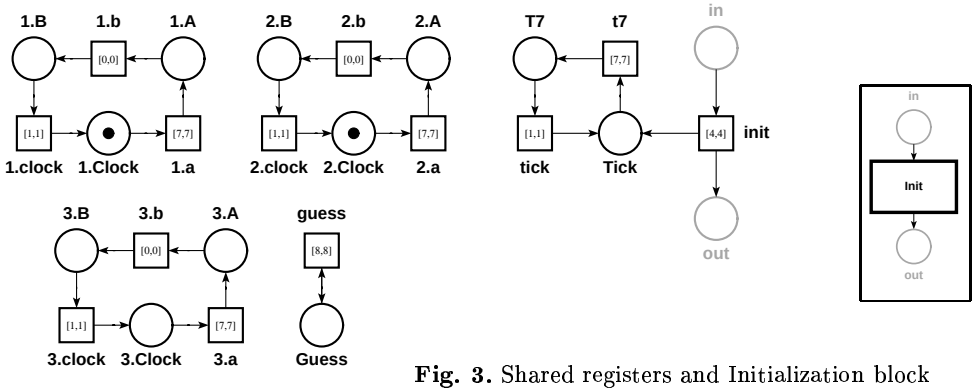

Fig. 3. Shared registers and Initialization block

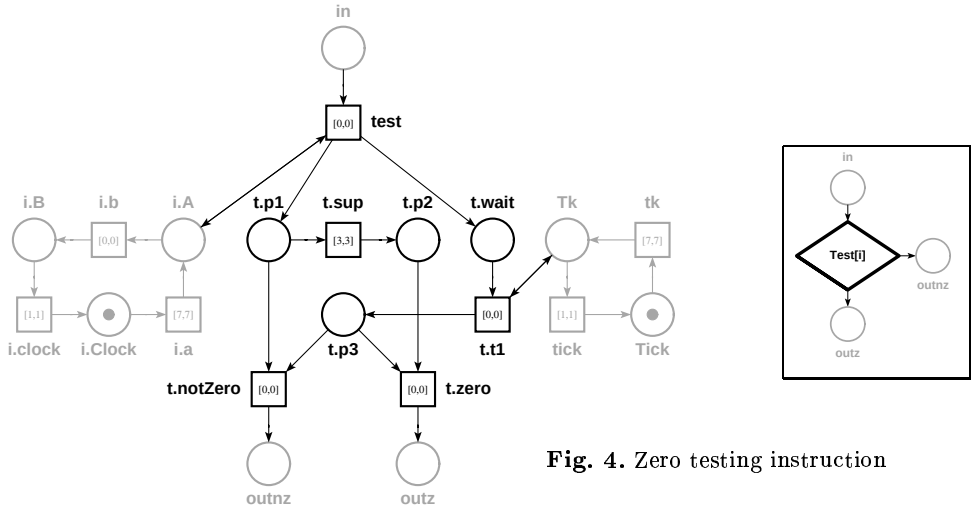

Fig. 4. Zero testing instruction

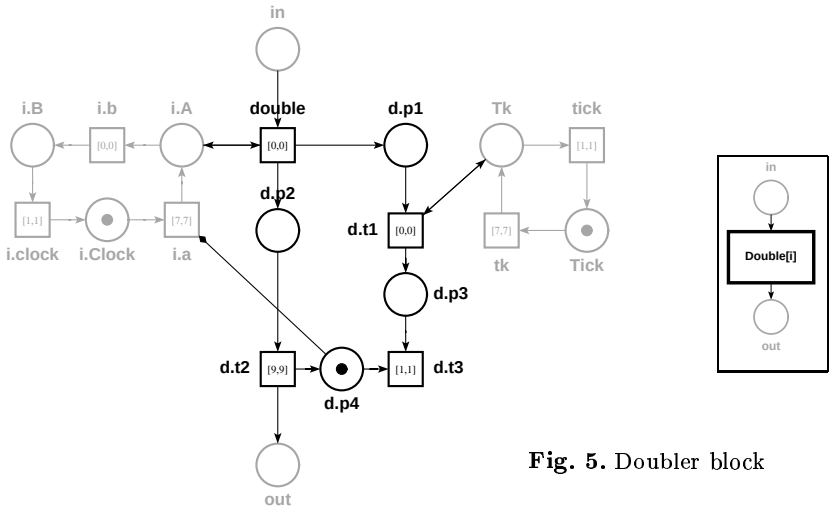

Fig. 5. Doubler block

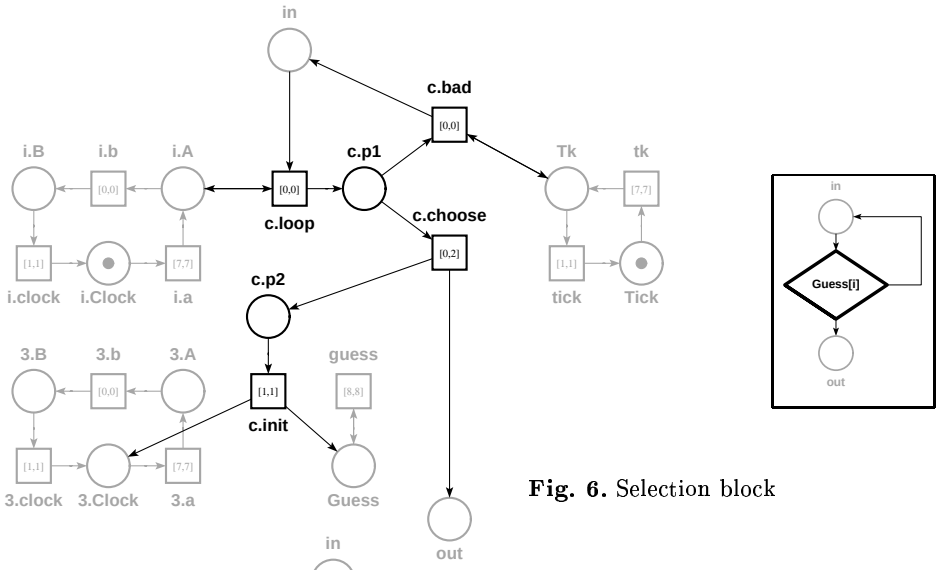

Fig. 6. Selection block

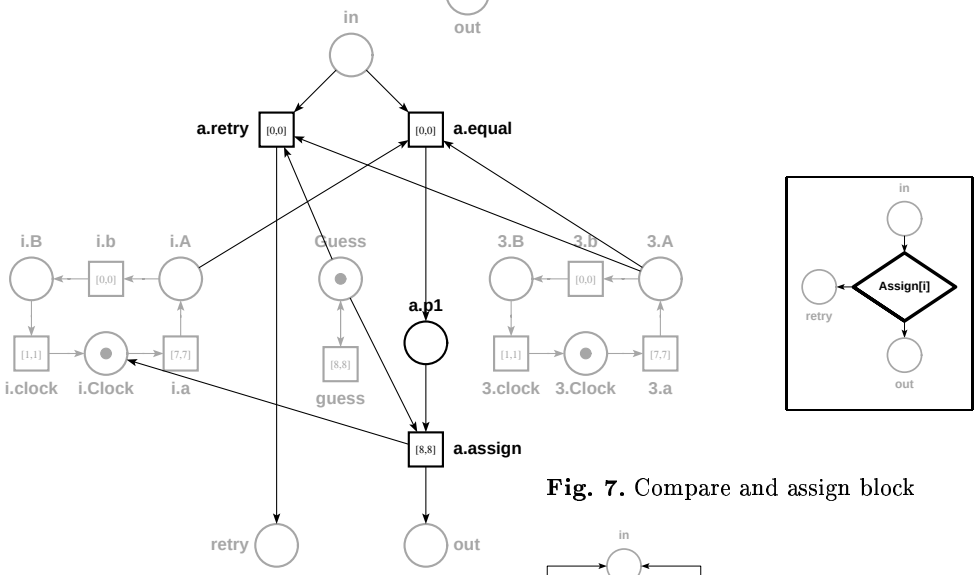

Fig. 7. Compare and assign block

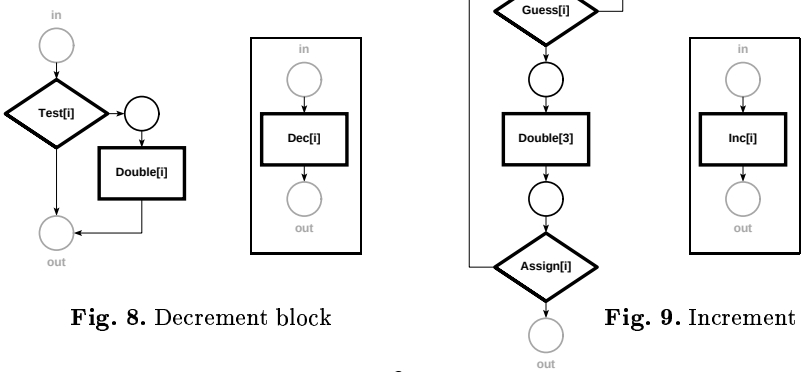

Fig. 8. Decrement block

Fig. 9. Increment block

in their period, some registers will hold several transitions in sequence (including the one materializing the event), with the sum of their delays equal to period ρ.

We need a register for each counter ($1.clock$ and $2.clock$), plus one for the reference event ($tick$). We also need registers for two temporary events ($3.clock$ and $guess$, uninitialized). Figure 3 shows the required registers, together with the *init* initialization block (transition *init* and connected edges). Places in and out (in grey) are not part of the block, they just materialize the context of use.

The value chosen for period ρ is 8, in any unit of time. Block init initializes the phase delays of both counter to 4 (encoding counter values of 0).

Zero testing instruction: Shown in Figure 4, this block tests whether the phase delay of register i equals its initial phase delay (4). As before, grey nodes and edges materialize the context. A token put in place in is propagated either to place $outz$ (if $\phi_i^{in} = 4$) or to $outnz$ (if $\phi_i^{in} \leq 2$). No stopwatch arc is required.

Doubler block: Shown in Figure 5, it doubles the phase delay of $i.clock$, assuming it smaller than 4. The arc from place $d.p4$ to $i.a$ is a stopwatch arc. Assume a token is put in place in, and $\phi_i^{in} = k$, with $0 < k \leq 2$. Then the token is propagated to place out, and $\phi_i^{out} = 2 * k$. As a proof sketch, assume $double$ fired at time t, then transition $i.a$ will become enabled at time $t + 1$, suspended at time $t + k + 1$, and then resumed at time $t + 9$. So, $i.clock$ will fire again at time $t + 17 - k$, and the following $tick$ at $t + k + 17$, establishing $\phi_i^{out} = 2 * k$.

Selection block: This block initializes registers $guess$ and 3, giving them a phase delay in interval $[0, \phi_i^{in}]$. It is represented in Figure 6.

Compare and assign block: Shown in Figure 7, this block compares the phases of $3.clock$ and $i.clock$. A token in place in propagates either to out (if $3.clock$ and $i.clock$ have equal phases) or, whatever the phases of $3.clock$ and $i.clock$ (thus non deterministically if $3.clock$ and $i.clock$ have equal phases), to place $retry$.

Decrement instruction: The decrement block, Figure 8, is implemented by a copy of the doubler block preceded by a copy of the test block. The test block prevents doubling the phase delay of register i if larger than 2 (that is to decrement it if it encodes the value 0).

Increment instruction: The increment block makes use of three blocks organized as shown Figure 9. Using a "selection" block, a phase value is first chosen between the phase of $i.clock$ and that of $tick$, and stored in registers $guess$ and 3. Next, a "doubler" block is used to double the phase delay of $3.clock$ wrt $tick$ and, finally, the phases of $i.clock$ and $3.clock$ are compared. If comparison is successful, the phase of $i.clock$ is updated to that of $guess$. Otherwise the whole instruction is restarted. A token put in place in may always propagate to place out, though it may take an arbitrary time to do so, and ϕ_i^{out} is half of ϕ_i^{in} then.

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

- If $\mathcal{M}$ has $n + 1$ states $q_0, q_1, \ldots, q_n$, then create $\mathcal{N}$ with $n + 2$ places named $start, p_0, p_1, \ldots, p_n$. 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 block Dec_i (resp. Inc_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 block $Test_i$, connected to places p_a (as input), p_b (as $outz$ output), and p_c (as $outnz$ output).

4.3 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 $s = (m, I)$ of $\mathcal{N}$, let us write $c \cong s$ iff:

- m marks p_i, $1.Clock$, $2.Clock$, and $Tick$,
- For $t \in \{1.clock, 2.clock, tick\}$, $I(t)$ holds a single point, with $I(tick) - I(1.clock) = 1/2^{x_1+1}$ and $I(tick) - I(2.clock) = 1/2^{x_2+1}$

Theorem 1. *A configuration c is reachable from c_0 in machine $\mathcal{M}$ (written $c_0 \to c$) iff some state s is reachable from s_0 in net $\mathcal{N}$ (written $s_0 \to s$) such that $s \cong c$. That is, iff:*

(i) $(\forall c)(c_0 \to c \Rightarrow (\exists s)(s_0 \to s \wedge s \cong c))$

(ii) $(\forall c)(\forall s)(s_0 \to s \wedge s \cong c \Rightarrow c_0 \to c)$

Proof. By induction on machine transitions, using the properties of blocks. □

Theorem 2. *State reachability is undecidable for bounded Stopwatch TPN's.*

Proof. Net $\mathcal{N}$ is clearly bounded, it is even safe (1-bounded). By Theorem 1 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 $\mathcal{N}$. □

As corollaries, we have that marking reachability, k-boundedness, and liveness are undecidable for bounded SwTPN's. Concerning other extensions of TPN's modeling preemption, bounded IHTPN's can be transformed into SwTPN's (replacing inhibitor arcs by stopwatch arcs from complementary places), and the doubler block is easily encoded by a Scheduling-TPN or Preemptive-TPN. So, Theorem 2 holds for these extensions of TPN's too, solving an open problem.

5 Approximate state spaces

Theorem 2 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 all reachable states, but possibly more. Such overapproximations provide sufficient conditions for safety properties.

Overapproximations for Preemptive-TPN's, Scheduling-TPN's and IHTPN's has been proposed in [BFSV04], [LR03], and [RL04], approximating the temporal domains of state classes by the smallest DBM including them. The method is efficient as the DBM shape is preserved for polyhedra, but approximations are often too coarse. On the other hand, approximations proposed for linear hybrid automata [ACH+95] are unnecessarily rich for our purposes, as will be seen.

We propose here another technique, 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 technique yields more accurate approximations than previous techniques, and its precision is adjustable.

Let us first remind Motzkin's decomposition theorem for polyhedra (see e.g. [Sch86]): 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 other properties of the polyhedra computed by Algorithm 1:

Theorem 3. *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 these may only move towards 0. For (ii), the only step in Algorithm 1 that may introduce oblique rays is (1), but these disappear after the projection step (3). □

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

Definition 3 (Polyhedra approximations). *Assume given some $k > 0$ ($k \in \mathbf{Q}$). Suppose $\mathbf{R}^{\mathbf{n}}_{+}$ discretized into hypercubes of side k, and let $\mathcal{H}_k$ denote the set of these hypercubes. Let $P \subseteq \mathbf{R}^{\mathbf{n}}_{+}$ be some polyhedron. Then, relative to k:*

- *A point $x \in \mathbf{R}^{\mathbf{n}}_{+}$ is approximated by the intersection of all hypercubes of $\mathcal{H}_k$ containing x.*
- *A polytope $Q \subseteq \mathbf{R}^{\mathbf{n}}_{+}$ 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 some k such that $h_k(P) = P$.

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

Algorithm 2 (Approximated state class graph for grid size k)
As Algorithm 1, except D' is approximated by $h_k(D')$ after step 3.

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

Theorem 4. *For any k and any inherently bounded SwTPN, Alg. 2 terminates.*

Proof. Approximated polyhedra obey Theorem 3 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 the polyhedra is finite too, and only finitely many polyhedra can be build from these and a finite set of rays. The reason for the restriction to inherently bounded SwTPN's is that relaxing time constraints may make unbounded the approximated behavior of a (simply) bounded SwTPN. □

Since any polyhedron can be approximated as closely as desired by adjusting 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 2).

Algorithm 2 has been implemented. Approximations only require polyhedra operations typically provided by available polyhedral libraries. As an example, the table below shows sizes of the approximated state class graphs for the net Figure 1, for various grid sizes and for two methods for grouping classes (Algorithm 2, using equivalence $\cong$, and its variant using $\subseteq$ instead, as suggested in Section 2.2). The approximated behavior becomes unbounded when the grid size is taken larger than 1 (this net is not inherently bounded). For the example in Figure 2, the exact behavior is found with grid size 1.

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

As defined in [LR03] or [BFSV04], the "smallest enclosing DBM" approximation often yields a too coarse approximation. This could be improved by making the method parameterizable, like ours, by defining a grid on the spaces of polyhedra and approximating polyhedra P by the smallest DBM including P with its extreme vertices on the grid instead of the smallest DBM with integer extreme vertices. With this easily implementable refinement, the user could indeed adapt approximations to the actual endpoints of static intervals in the net (for the same precision of approximation, smaller endpoints require a smaller grid).

Now, even with this improvement, the "smallest enclosing DBM" approximation cannot approximate some polyhedra as closely as the "quantization" approximation, for the simple reason that it gives a necessarily coarser approximation when the polyhedron is defined from non-DBM constraints. Such constraints frequently occur in practice. As an illustration, Figure 10 shows the results of approximations with grid size $k = 1$ of two simple example polyhedra in 2 dimensions taken (projected) among those computed by Algorithm 1 for the net Figure 1. For the left polyhedron, defined by $\{0 \leq y,\ y \leq x \leq 3/4\}$ and pictured

in black, both approximations yield the same result (the outer triangle). But for the right polyhedron, defined by $\{0 \leq x \leq 1,\ x + y = 1\}$ (the black diagonal) the "smallest enclosing DBM" method yields the full square of side 1 (whatever the grid size $k \leq 1$ chosen), while "quantization" yields the exact polyhedron.

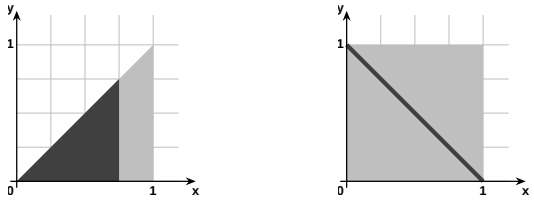

Fig. 10. Exact polyhedra and overapproximations.

Finally, it is worth to note that the "quantization" method is applicable to TPN's too (without stopwatches). It would be useful in cases where state space abstractions are huge. In this context, it preserves the DBM shape of polyhedra.

6 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 [RL04]. 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 undecidability of all these problems in all known similar 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, similar to those available for the above TPN's extensions, but implemented. Experiments with the implementation 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 SwTPN's, 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 TPN's.

Though conceptually close, the proposed techniques are more demanding than the similar techniques used for TPN's, computationally. Prospective work addresses algorithmic aspects for their efficient implementation.

References

[ACH+95] R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995.

[BFSV04] G. Bucci, A. Fedeli, L. Sassoli, and E. Vicario. Time state space analysis of real-time preemptive systems. *IEEE transactions on software engineering*, 30(2):97–111, February 2004.

[BH04] H. Boucheneb and R. Hadjidj. Towards optimal CTL^* model checking of time Petri nets. In *Proceedings of 7th Workshop on Discrete Events Systems*, Reims, France, September 2004.

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

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

[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 *Proc. Tools and Algorithms for the Construction and Analysis of Systems (TACAS'2003), Springer LNCS 2619*, 2003.

[Čer92] K. Čerāns. *Algorithmic problems in analysis of real time system specifications.* University of Latvia, Dr.sc.comp. Thesis, 1992.

[CL00] F. Cassez and K. G. Larsen. The impressive power of stopwatches. In *11th Int. Conf. on Concurrency Theory, (CONCUR'2000), University Park, P.A., USA*, Springer LNCS 1877, pages 138–152, 2000.

[DT98] C. Daws and S. Tripakis. Model checking of real-time reachability properties using abstractions. In *Proc. Tools and Algorithms for the Construction and Analysis of Systems (TACAS'1998), Springer LNCS 1384*, 1998.

[HKPV95] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? In *Proceedings of the 27th Annual Symposium on Theory of Computing (STOC), ACM Press*, pages 373–382, 1995.

[Jea02] B. Jeannet. The polka convex polyhedra library, edition 2.0.1. http://www.irisa.fr/prive/bjeannet/newpolka.html, IRISA, Rennes, 2002.

[JLL77] N. D. Jones, L. H. Landweber, and Y. E. Lien. Complexity of some problems in Petri nets. *Theoretical Computer Science 4*, pages 277–299, 1977.

[LR03] D. Lime and O. H. Roux. Expressiveness and analysis of scheduling extended time Petri nets. In *5th IFAC International Conference on Fieldbus Systems and their Applications, (FET'03).* Elsevier Science, July 2003.

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

[Min61] M. Minsky. Recursive unsolvability of Post's problem. *Ann. of Math*, 74:437–454, 1961.

[RD02] O. H. Roux and A.-M. Déplanche. A t-time Petri net extension for real time task scheduling modeling. *Eur. Journal of Automation (JESA)*, 36(7), 2002.

[RL04] O. H. Roux and D. Lime. Time Petri nets with inhibitor hyperarcs. formal semantics and state space computation. In *Proc. Int. Conf. on Applications and Theory of Petri Nets (ICATPN'04), Bologna, Italy*, 2004. (to appear).

[Sch86] A. Schrijver. *Theory of Linear and Integer Programming.* John Wiley and Sons, NY, 1986.