An Efficient Method for Computing Exact State Space of Petri Nets With Stopwatches

Morgan Magnin
IRCCyN, Nantes, France

Didier Lime
Aalborg University - CISS, Denmark

Olivier (H.) Roux
IRCCyN, Nantes, France

Abstract

In this paper, we address the issue of the formal verification of real-time systems in the context of a preemptive scheduling policy. We propose an algorithm which computes the state-space of the system, modeled as a time Petri net with stopwatches, exactly and efficiently, by the use of Difference Bounds Matrices (DBM) whenever possible and automatically switching to more time and memory consuming general (convex) polyhedra only when required. We propose a necessary and sufficient condition for the need of general polyhedra. We give experimental results comparing our implementation of the method to a full DBM over-approximation and to an exact computation with only general polyhedra.

Keywords: real-time systems, time Petri nets, polyhedra

1 Introduction

As systems demanding correctness proofs increase in complexity, we may need to consider formal models involving actions that can be suspended with a

1 Email: Morgan.Magnin@irccyn.ec-nantes.fr
2 Email: didier@cs.aau.dk
3 Email: Olivier-h.Roux@irccyn.ec-nantes.fr
memory of their current status. An obvious application is the modeling of preemption in the context of multi-tasking. This notion of suspension requires the introduction of variables whose continuous evolution may be stopped for a while and later resumed at the same point. This leads to the extension of traditional clock variables by “stopwatches”, of timed automata by stopwatch automata [6] and of time Petri nets by several related models including Preemptive time Petri nets (Preemptive-TPNs [5]) and Scheduling time Petri nets (Scheduling-TPNs [18]).

Verification of properties on a formal model involves the investigation of part or the whole set of its reachable states: Its state-space. This state-space is generally infinite due to the dense-time semantics considered. As a consequence, verification algorithms compute finite abstractions of it, preserving the properties to be verified. In these abstractions, concrete states are encoded into symbolic sets of states described by a discrete part (location or marking) and a continuous part (value of clocks/stopwatches). The continuous part is represented by a convex polyhedron, that is to say, a set of linear inequations.

In the case of models with simple clocks (timed automata, time Petri nets), the considered polyhedra have a degenerated form, which can be encoded into an efficient data structure called Difference Bound Matrix (DBM [4,8]). Handling DBMs is much more efficient than handling general polyhedra.

When dealing with stopwatches, polyhedra retain their general form which cannot be encoded into DBMs [14,17]: The gain of expressivity of stowatches is balanced by undecidability results [11,3], an increased complexity of the verification algorithms, and a much higher memory consumption.

A natural counter-measure against the high complexity and undecidability results linked to the handling of general polyhedra consists of over-approximating the computed state-space by approximating the polyhedra by simpler englobing polyhedra, such as the tightest englobing DBM which yields a good speed-up of the computation [14,5,3]. The properties preserved by an over-approximation are however limited to safety: The system is checked for the non-satisfaction of a given “bad” property. The intuitive reason for this is that the actual behavior of the system is included in the over-approximated one. Also, with a DBM over-approximation, the number of DBMs to be considered in the computation is finite, which is not necessarily true for general polyhedra. This may thus make the over-approximated computation terminate, while the exact one does not.

In order to perform an exact analysis in an efficient way, recent works use an over-approximation as a pre-computing and then refine the results to exactness by restricting them with the timing constraints of the net: In [5], the authors over-approximate the computation of the state class graph of a Pre-
emptive-TPN by using DBMs. Then, given an untimed transition sequence from the over-approximated state class graph, they can obtain the feasible timings between the firing of the transitions of the sequence as the solution of a linear programming problem. In particular, if there is no solution, the transition sequence has been introduced by the over-approximation and can be cleaned up, otherwise the solution set allows to check timed properties on the firing times of transitions. In [15], the authors translate Scheduling-TPNs into stopwatch automata and use HyTech [10] for the subsequent verification. The translation uses a DBM over-approximation to obtain the discrete structure of the automaton. They then compute guards and invariants syntactically from the timing constraints of the net. Thus, the discrete locations that were possibly added by the over-approximation are made unreachable and the obtained stopwatch automaton is proved to be time-bisimilar to the initial Scheduling-TPN. This intuitively means that its behavior is the same as that of the Scheduling-TPN (and not an over-approximation).

However, the over-approximation used in these two methods may cause them to not terminate by adding an infinity of false behaviors to the model, while an exact computation would terminate. Put in another way, the over-approximation computes an infinity of unreachable markings while the net is indeed bounded.

The method developed in this paper also tackles the issue of exact and efficient state-space computation of stopwatch extensions of TPNs. It is in particular applicable to Preemptive-TPNs and Scheduling-TPNs. However, for the sake of simplicity, it is explained on the Scheduling-TPN model.

Our approach is based on the following two remarks:

(i) the initial symbolic state of a Scheduling-TPN can always be represented by a DBM,

(ii) it is “easy” to determine if a given polyhedron is a DBM

By extending the necessary condition given in [5], for detecting the need of general polyhedra, to a necessary and sufficient condition, we are able to propose a mixed exact computation of the state-space, which uses both the efficient DBM representation and, only when required, the general polyhedra representation. We have implemented the method for Scheduling-TPNs and we illustrate its efficiency through experimental results.

The paper is organized as follows: Section 2 gives the formal definition of the Scheduling-TPN model and illustrates the state class graph computation. Section 3 introduces theorems and proofs about polyhedral computation. Section 4 describes an algorithm that improves the efficiency of exact state space computation. Finally, in section 5, we give a brief description of the tool
implementing the algorithm and some experimental results.

2 Scheduling Time Petri nets

2.1 Definition and semantics of Scheduling-TPNs

In [16], Roux et al. introduce an extension of TPNs to scheduling. This extension consists of enriching the Time Petri net model with the scheduling policies (i.e. the way the different schedulers of the system activate or suspend the tasks).

**Definition 2.1** A Scheduling-Time Petri Net (Scheduling-TPN) is a n-tuple \( T = (P, T, \cdot^*, ()^*, \alpha, \beta, M_0, \text{Act}) \) where:

- \( P = \{p_1, p_2, \ldots, p_m\} \) is a non-empty finite set of places.
- \( T = \{t_1, t_2, \ldots, t_n\} \) is a non-empty finite set of transitions \( (T \cap P = \emptyset) \).
- \( \cdot^* : T \rightarrow (\mathbb{N}^P) \) is the backward incidence function.
- \( ()^* : T \rightarrow (\mathbb{N}^P) \) is the forward incidence function.
- \( M_0 \in \mathbb{N}^P \) is the initial marking of the net.
- \( \alpha \in (\mathbb{Q}^+)^T \) and \( \beta \in (\mathbb{Q}^+ \cup \{\infty\})^T \) are functions giving for each transition respectively its earliest and latest firing times \( (\alpha \leq \beta) \).
- \( \text{Act} \in (\mathbb{N}^P)^{\mathbb{N}^P} \) is the active marking function. \( \text{Act}(M) \) represents the interpretation of the marking \( M \) over the scheduling strategy.

\( \text{Act} \) is the specific element that extends TPNs to Scheduling-TPNs.

**Example 2.2** An example of a Scheduling-TPN is presented in figure 1. The initial marking of the net is \( \{P_1, P_3\} \). However, since those two places are allocated to the same processor and the priority of \( P_3 \) is the highest, the initial active marking is \( \{P_3\} \). So the first transition fired will be \( T_3 \).

A marking \( M \) of the net is an element of \( \mathbb{N}^P \) such that \( \forall p \in P, M(p) \) is the number of tokens in the place \( p \). An active marking \( \text{Act}(M) \) of the net is an element of \( \mathbb{N}^P \) such that \( \forall p \in P \):
• $\text{Act}(M(p)) = 0$ if the task associated to the place $p$ is currently suspended;
• $\text{Act}(M(p)) = M(p)$ otherwise.

In [16], $\text{Act}(M)$ is defined for a fixed priority scheduling policy; three new parameters are introduced to deal with this model:
• $\text{Proc}\{\phi, \text{proc}_1, \text{proc}_2, \ldots, \text{proc}_l\}$ is a finite set of processors (including $\phi$ that is introduced to specify that a place is not assigned to an effective processor of the hardware architecture),
• $\omega \in \mathbb{N}^P$ is the priority assignment function.
• $\gamma \in \text{Proc}^P$ is the allocation function.

Actually, when a place $p$ does not represent a true activity for a processor (for example a register or memory state), neither a processor ($\gamma$) nor a priority ($\omega$) have to be attached to it. That means we always have $\text{Act}(M(p)) = M(p)$ for such a place.

A transition $t$ is said to be active if it is enabled by the active marking $\text{Act}(M)$. We denote it by $t \in \text{enabled}(\text{Act}(M))$. Transitions that are enabled but not active are said to be suspended.

Let $M$ be a marking of the net and $t_i$ a firable transition. We will denote by $\uparrow \text{enabled}(M, t_i)$ the set of transitions newly enabled by the firing of $t_i$, i.e. transitions enabled by the new marking $M - \bullet t_i + t_i^*$ but not by $M - \bullet t_i$ (if $t_i$ remains enabled after its firing then it is considered as newly enabled). Formally,

$$\uparrow \text{enabled}(M, t_i) = \{t_k \in T \mid (t_k \leq M - \bullet t_i + t_i^*) \land ((t_k = t_i) \lor (t_k > M - \bullet t_i))\}$$

Similarly, we will denote by $\text{disabled}(M)$ the set of transitions disabled by the firing of $t_i$, i.e. transitions enabled by $M$ but not by $M - \bullet t_i$.

A valuation is a mapping $\nu \in (\mathbb{R}^+)^T$ such that $\forall t \in T, \nu(t)$ is the time elapsed since $t$ was last enabled. Note that $\nu(t)$ is meaningful only if $t$ is an enabled transition. $\overline{0}$ is the null valuation such that $\forall k, \overline{0}_k = 0$.

We can define the semantics of scheduling extended time Petri nets as Timed Transition Systems (TTS) [13]. In this model, two kinds of transitions may occur: Continuous transitions when time passes and discrete transitions when a transition of the net fires.

**Definition 2.3** The semantics of a Scheduling-TPN $T$ is defined as a TTS $\mathcal{S}_T = (Q, q_0, \rightarrow)$ such that
• $Q = \mathbb{N}^P \times (\mathbb{R}^+)^T$ represents the set of all states of the system
• $q_0 = (M_0, \overline{0})$ is the initial state
• $\rightarrow \in Q \times (T \cup \mathbb{R}) \times Q$ is the transition relation including a continuous
transition relation and a discrete transition relation.

· The continuous transition relation is defined \( \forall d \in \mathbb{R}^+ \) by:

\[
(M, \nu) \xrightarrow{d} (M, \nu') \quad \text{iff} \quad \begin{cases} 
\forall t_i \in \text{enabled}(M), \nu'(t_i) = \nu(t_i) \text{ if } \text{Act}(M) < t_i \land M \geq \bullet(t_i) \\
\nu(t_i) + d \text{ otherwise},
\end{cases}
\]

\[
\forall t_k \in T, M \geq \bullet t_k \Rightarrow \nu'(t_k) \leq \beta(t_k)
\]

· The discrete transition relation is defined \( \forall t_i \in T \) by:

\[
(M, \nu) \xrightarrow{t_i} (M', \nu') \quad \text{iff} \quad \begin{cases} 
\text{Act}(M) \geq \bullet t_i, \\
M' = M - t_i + t_i\bullet, \\
\alpha(t_i) \leq \nu(t_i) \leq \beta(t_i), \\
\forall t_k, \nu'(t_k) = \begin{cases} 0 & \text{if } t_k \in \uparrow \text{enabled}(M, t_i), \\
\nu(t_k) & \text{otherwise}
\end{cases}
\end{cases}
\]

2.2 State space abstraction for Scheduling-TPN

2.2.1 State class graph for Scheduling-TPN.

In order to analyze a time Petri net, the computation of its reachable state space is required. However, the reachable state space of a time Petri net is obviously infinite: is indeed so a method has been proposed by Berthomieu and Diaz [2] to partition it in a finite set of infinite state classes. In [14], LIME and ROUX extended this method and gave a semi-algorithm for computing the state space of a Scheduling-TPN (as proven in [16], reachability and boundedness problems for Scheduling-TPNs are undecidable).

The firing domain of a time Petri net class was proved to be of the form \( \theta_i - \theta_j \leq \gamma_{ij} \) and \( \alpha_k \leq \theta_k \leq \beta_k \) where \( i, j \in \{1, \ldots, n\}, k \in \{1, \ldots, m\} \) and \( \gamma_{ij}, \alpha_k \) (and \( \beta_k \)) are integer constants depending on \( i, j \) and \( k \). \( \theta_i \) represents the time valuation associated to transition \( t_i \).

State classes of Scheduling-TPNs are still defined as a pair with a marking and a firing domain. However, with the presence of stopwatches (here the valuation of the clocks), the firing domain of state classes cannot be encoded into a DBM anymore; a general polyhedron form is required.

**Theorem 2.4** A state class \( C \) of a Scheduling-TPN is a pair \((M, D)\) where \( M \) is a marking of the net and \( D \) a set of inequations. For Scheduling-TPNs, the general form of a domain \( D \) is that of a (convex) polyhedron with \( m \) constraints
(m ∈ N) involving up to n variables, with n being the number of transitions enabled by the marking of the class:

\[ A\vec{\theta} \leq B \]

with A and B being rational matrices of respective dimensions (m, n) and (m, 1) and \( \vec{\theta} \) being a vector of dimension n.

In the case of TPNs, the firing domain is simpler than a general polyhedron and therefore can be encoded into the efficient DBM datastructure [4,8]).

We need a new definition for the firability of a transition from a class, adapted to time Petri nets with stopwatches:

**Definition 2.5** [Firability] Let \( C = (M, D) \) be a state class of a Scheduling-TPN. A transition \( t_i \) is said to be firable from \( C \) iff there exists a solution \((\theta^*_0, \ldots, \theta^*_{n-1})\) of \( D \), such that \( \forall j \in [0..n-1] - \{i\}, \) s.t. \( t_j \) is active, \( \theta^*_i \leq \theta^*_j \).

Now, given a class \( C = (M, D) \) and a firable transition \( t_f \), the class \( C' = (M', D') \) obtained from \( C \) by the firing of \( t_f \) is given by

- \( M' = M - \cdot t_f + t_f \cdot \)
- \( D' \) is computed along the following steps, and noted next\((D, t_f)\): (i) variable substitutions for all enabled transitions that are active \( t_j \): \( \theta_j = \theta_f + \theta'_j \), (ii) intersection with the set of positive or null reals \( \mathbb{R}^+ \): \( \forall i, \theta'_i \geq 0 \), (iii) elimination (using for instance the Fourier-Motzkin method [7]) of all variables relative to transitions disabled by the firing of \( t_f \), (iv) addition of inequations relative to newly enabled transitions

\[ \forall t_k \in \uparrow enabled(M, t_f), \alpha(t_k) \leq \theta'_k \leq \beta(t_k). \]

Every polyhedron has a minimal representation [1]. Nevertheless, the complexity of the minimization of a general polyhedron is exponential at the worst case while, for DBM, complexity of inclusion testing, equality testing and inclusion is polynomial \( (O(n^2) \) where \( n \) is the number of variables) and complexity of empty testing and minimization is \( O(n^3) \).

Variable substitutions correspond to a translation of time origin for enabled transitions: The new origin corresponds to the firing time of \( t_f \). Constraints \( \theta'_i \geq 0 \) can also be written \( \theta_i \geq \theta_f \), which express the fact that we chose to fire \( t_f \), that means all the other enabled transitions are fired later.

The fact that the firing domain cannot always be expressed with a DBM practically means that, for example, a class may have the following domain:

\[ \{0 \leq \theta_1 \leq 3, 0 \leq \theta_2 \leq 4, 0 \leq \theta_3 \leq 4, 1 \leq \theta_2 + \theta_3 \leq 7\} \]  (1)
What we can see here is that the two last inequations cannot be expressed with a DBM. Furthermore, we can easily see that those new inequations may give even more complex inequations (i.e. involving more variables) when firing another transition for the domain.

2.2.2 DBM-over-approximation

Handling general polyhedra is much more time and memory consuming than for DBMs [1]. In order to be able to keep these algorithms efficient for Scheduling-TPNs, Lime et al. proposed an over-approximation of the state class graph of such a model by using DBM [14]. This method consists in wrapping a polyhedron in a DBM that contains it. This can be illustrated on the previous example: The DBM over-approximation consists in writing that the firing domain can be approximated by the following:

\[
\begin{align*}
0 & \leq \theta_1 \leq 3, \\
0 & \leq \theta_2 \leq 4, \\
0 & \leq \theta_3 \leq 4
\end{align*}
\] (2)

There is an obvious drawback to this over-approximation: It may add, in the state class graph, states that should not be reachable. Moreover, the state class graph can become infinite by doing this DBM-over-approximation while the exact state class graph is finite.

This is illustrated by the net of figure 2 (the second example proposed in [5]): DBM-over-approximated state class graph remains bounded as long as we do not increase the execution time of task 1 \(t_{13}\). If it is increased to \([10, 23]\), then the DBM approximated state class graph becomes unbounded, while the exact state class graph is still finite.
3 Necessary and sufficient condition for constraint relaxation

As we have seen before, the specificity of the state class graph of a TPN with stopwatches is that some non-DBM polyhedral forms appear. A major issue is then to determine *a priori* when such polyhedral states appear in the state class graph of the Scheduling-TPN we study. One condition for the relaxation of constraints by the DBM-over-approximation in a class is following:

**Proposition 3.1** The parent class includes both suspended and active transitions which are continuously enabled before, during and after the firing of the transition that led to the current class.

In [5], this condition is claimed to be necessary and sufficient. This condition is indeed necessary, but not sufficient as we will now prove it by showing counterexamples.

Let us study the net of figure 3. In the initial class, there is both a suspended transition ($t_1$) and an active one ($t_2$), which are persistent after the firing of $t_4$. In the initial class, $t_4$ is the only firable transition. Then it leads to following class:

$$\{ \{P_1, P_2\}, \{1 \leq \theta_1 \leq 5, \theta_2 = 1\} \}$$

The firing domain of such a class can be represented as a DBM. Then it follows that condition is not sufficient to determine when the computed polyhedron will not be a DBM.

In fact, condition 3.1 needs some additional constraints on the timings of transitions in the firing domain for being relevant. This can be illustrated by the net of figure 4. After firing $t_2$, there is both a suspended transition ($t_1$) and an active one ($t_3$), which are persistent after the firing of $t_4$. The firing of
$P_1, \gamma = 1, \omega = 1 \quad P_2 \quad P_4$

$t_1 [4, 5] \quad t_2 [1, 1] \quad t_4 [2, 4]$

$P_3, \gamma = 1, \omega = 2$

$t_3 [1, 2]$

Fig. 4. Second counterexample

$t_4$ leads to following class:

$\{ \{ P_1, P_3 \}, \{ 3 \leq \theta_1 \leq 4, 0 \leq \theta_3 \leq 1 \} \}$

The firing domain of such a class can be expressed with a DBM. However, if the firing interval associated to transition $t_2$ is decreased to $[0; 1]$, then the firing sequence $t_2.t_4$ leads to:

$\{ \{ P_1, P_3 \}, \{ 3 \leq \theta_1, 0 \leq \theta_3 \leq 1, \theta_1 + \theta_3 \leq 5 \} \}$

We made a non-DBM polyhedral form appear by only changing the firing interval of transition $t_2$, that definitely shows condition 3.1 is not sufficient to define the cases when the over-approximation relaxes constraints.

**Theorem 3.2 (Effective over-approximation)**

Let $C = (M, D)$ be a Scheduling-TPN state class such that $D$ is a DBM ($D = \{ \alpha_i \leq \theta_i \leq \beta_i, \theta_i - \theta_j \leq \gamma_{ij} \}$ is the canonical domain). Let $t_f$ be a firable transition from $C$ : the firing of $t_f$ leads to $C' = (M', D')$. Let $\overline{D'}$ be the DBM-overapproximated domain obtained from $D'$.

$\overline{D'}$ relaxes constraints of $D$ (i.e. domain $D'$ cannot be represented only with DBMs; its representation needs the use of general polyhedra) iff $\exists i \in \text{enabled}(\text{Act}(M)), \exists j \in \text{enabled}(M) - \text{enabled}(\text{Act}(M)), \exists k \in \text{enabled}(M) - \text{disabled}(M, t_f)$, s.t.. $i \neq k$ and $\beta_j + \gamma_{ki} > \beta_k + \gamma_{ji} \lor \alpha_j - \gamma_{ik} < \alpha_k - \gamma_{ij}$

**Proof.** Let $C' = (M', D')$ be a state class of the Scheduling-TPN. Let $C = (M, D)$ be its parent class. For this demonstration, we consider a domain with 4 variables but the proof can be easily extended to $n$ variables. The enabled transitions $t_1, t_2, t_3$ and $t_4$ are associated to the variables $\theta_1, \theta_2, \theta_3$ and $\theta_4$. We assume $t_1, t_2$ and $t_3$ are active, and $t_4$ is not. Finally, we assume that $t_1$ is firable, that the firing of $t_1$ in class $C$ leads to class $C'$ and that $t_2$ remains enabled in $C'$. 


The initial domain $D$, in its canonical form, can be written as follows:

$$
D = \begin{cases} 
\alpha_1 \leq \theta_1 \leq \beta_1, & \alpha_2 \leq \theta_2 \leq \beta_2, \\
\alpha_3 \leq \theta_3 \leq \beta_3, & \alpha_4 \leq \theta_4 \leq \beta_4, \\
-\gamma_{21} \leq \theta_1 - \theta_2 \leq \gamma_{12}, & -\gamma_{31} \leq \theta_1 - \theta_3 \leq \gamma_{13}, \\
-\gamma_{41} \leq \theta_1 - \theta_4 \leq \gamma_{14}, & -\gamma_{32} \leq \theta_2 - \theta_3 \leq \gamma_{23}, \\
-\gamma_{42} \leq \theta_2 - \theta_4 \leq \gamma_{24}, & -\gamma_{43} \leq \theta_3 - \theta_4 \leq \gamma_{34}.
\end{cases} 
$$

(3)

Now, we suppose that at least one of these four inequations is satisfied:

$$
\begin{align*}
\beta_2 + \gamma_{41} &< \beta_4 + \gamma_{21} \\
\alpha_4 - \gamma_{12} &< \alpha_2 - \gamma_{14} \\
\beta_2 + \gamma_{43} &< \beta_4 + \gamma_{23} \\
\alpha_4 - \gamma_{32} &< \alpha_2 - \gamma_{34}
\end{align*}
$$

Let us compute the domain $D'$ obtained from $D$ by firing $t_1$: we begin by doing the variable substitution $\theta_i \leftarrow \theta_i' + \theta_1$ for all active transitions, except the disabled transition $t_1$. We add the inequation $\forall j, \theta_j' \geq 0$. Then we write the new inequations in order to use Fourier-Motzkin method:

$$
\begin{cases} 
\alpha_1 \leq \theta_1, & \theta_1 \leq \beta_1, \\
\alpha_2 - \theta_2' \leq \theta_1, & \theta_1 \leq \beta_2 - \theta_2', \\
\alpha_3 - \theta_3' \leq \theta_1, & \theta_1 \leq \beta_3 - \theta_3', \\
-\gamma_{41} + \theta_4 \leq \theta_1, & \theta_1 \leq \gamma_{14} + \theta_4, \\
-\gamma_{42} + \theta_4 - \theta_2' \leq \theta_1, & \theta_1 \leq \gamma_{24} + \theta_4 - \theta_2', \\
-\gamma_{43} + \theta_4 - \theta_3' \leq \theta_1, & \theta_1 \leq \gamma_{34} + \theta_4 - \theta_3', \\
\alpha_4 \leq \theta_4' \leq \beta_4, & \theta_2' \geq 0, \\
-\gamma_{32} \leq \theta_2' - \theta_3' \leq \gamma_{23}, & \theta_3' \geq 0, \\
-\gamma_{21} \leq -\theta_2' \leq \gamma_{12}, \\
-\gamma_{31} \leq -\theta_3' \leq \gamma_{13}
\end{cases} 
$$

(4)

The Fourier-Motzkin method then consists in writing that the system has solutions if and only if the lower bounds of $\theta_1$ are less or equal to the upper bounds. Then, we can deduce from the previous domain the following list of
We can notice all non-DBM constraints of the four last lines use, at least, one transition which was active in \(C\) (\(t_2\) or \(t_3\)) and one transition which was suspended (\(t_4\)). It follows that the proof of previous necessary condition is immediate: after firing \(t_1\), if class \(C'\) does not contain, at least, one transition which was active in \(C\) and one transition which was inactive, there is no non-DBM polyhedral form in the firing domain of \(C'\).

Nevertheless, one should pay attention to the fact that the reciprocal is false: Let us suppose, for instance, that \(\gamma_{24} = \beta_2 - \alpha_4\) and \(\gamma_{41} = \beta_4 - \alpha_1\) (i.e. these constraints were redundant in \(D\), which is the case when we start from the initial class), then inequation \(\theta'_2 + \theta_4 \leq \beta_2 + \gamma_{41}\) can be obtained by combining \(\theta'_2 \leq \gamma_{21}\) and \(\theta_4 \leq \beta_4\). It is then redundant and we can procede the in same way for the other constraints. In particular, this implies it is impossible to obtain non-DBM polyhedral from when firing a transition starting from the initial class.

We have now to prove that, among the four polyhedral constraints we got previously, there is at least one which is not redundant with the constraints on \(\theta'_2 + \theta_4\) that we can deduce from the individual constraints on \(\theta'_2\) and \(\theta_4\) (inequations on \(\theta'_3 + \theta_4\) are not interesting for us, as \(t_3\) may be disabled after firing \(t_1\)), that means:

\[
\begin{align*}
\alpha_4 + \max\{0, -\gamma_{12}\} &\leq \theta'_2 + \theta_4 \leq \beta_4 + \gamma_{21}, \\
\alpha_4 - \gamma_{32} &\leq \theta'_2 + \theta_4 - \theta'_3 \leq \beta_4 + \gamma_{23}.
\end{align*}
\]

This verification is immediate, as we supposed at least one of following
inequations is satisfied:

\[
\begin{align*}
\beta_2 + \gamma_{41} &< \beta_4 + \gamma_{21} \\
\alpha_4 - \gamma_{12} &< \alpha_2 - \gamma_{14} \\
\beta_2 + \gamma_{43} &< \beta_4 + \gamma_{23} \\
\alpha_4 - \gamma_{32} &< \alpha_2 - \gamma_{34}
\end{align*}
\]

So there appears, in the resulting domain, a non-DBM polyhedral form which is not redundant with the other constraints. Consequently the DBM-over-approximation relaxes this constraint.

We have still to prove the reciprocal. In order to do that, let us try to prove the contraposum. Then, suppose that the parent class does not include both an active transition and a suspended transition so that these two transitions remain enabled after the firing of \( t_f \) (it is then immediate that the overapproximated domain is equal to the exact domain) or that none of the two inequations on \( \alpha_k - \gamma_{ij} \) et \( \beta_k + \gamma_{ji} \) is verified (this second case means that the polyhedral constraint possibly resulting is redundant with the constraints obtained separately on \( \theta_i \) and \( \theta_j \), that means we do not have here a non-DBM constraint). Then, the DBM-over-approximation does not relax any constraint. The claimed result is then proven.

\[ \Box \]

4 Improved algorithm for computing the exact state space of a Scheduling-TPN

Practically, when studying Scheduling-TPNs, we observe the DBM-over-approximation often relaxes constraints. That means that the exact computation may be needed in many cases when we want a sharper verification of the system. The major drawback of manipulating general polyhedra is the performance loss in terms of computation speed and memory usage.

The main idea of our algorithm is that we do not always need to manipulate polyhedra when these polyhedra can be stored as DBM. Each time we can use DBM, we use them instead of general polyhedra. Moreover, we use theorem 3.2 to determine a priori when the DBM computation is going to relax constraints. If the necessary and sufficient condition is verified, then we are sure that polyhedral computation is needed. Otherwise, we use the DBM manipulations which are much faster. So, our algorithm mixes DBM manipulation and polyhedral manipulation, in function of the data structure that has to be manipulated. The advantages of our method can be easily understood on a simple example.

Let us consider the net of figure 5. The number of nodes and transitions
\(P_1, \gamma = 1, \omega = 1\)  
\(P_2\)  
\(P_3, \gamma = 1, \omega = 2\)  
\(t_1 [4, 5]\)  
\(t_2 [0, 3]\)  
\(t_4 [2, 4]\)  
\(t_3 [1, 2]\)
are the same for both exact and overapproximated computation (8 nodes and 10 transitions). That does not mean though that state class graph obtained by the DBM-over-approximation and by the polyhedral algorithm are the same. The difference lies here in only one class. By doing the firing sequence $t_2.t_4$, the resulting class $C_4$ is as follows:

$$
\begin{cases}
\{P_1, P_3\}, \\
0 \leq \theta_1 \leq 1, 0 \leq \theta_3 \leq 2, \theta_1 + \theta_3 \leq 5
\end{cases}
$$

The DBM-over-approximation leads to a similar class, except that the last inequation does not appear in the associated domain. Even if this inequation is taken into account or not, there is only one firable transition from this class: $t_3$. After firing $t_3$, the DBM-over-approximation and the exact computation lead to the same class $C_6$:

$$
\begin{cases}
\{P_1\}, \\
1 \leq \theta_1 \leq 5
\end{cases}
$$

For the next classes (which will be obtained by firing $t_1$), it is not necessary to manipulate general polyhedra (unless if the condition of theorem 3.2 is verified): DBM are sufficient. Consequently, we then store the resulting domain as a DBM and make manipulations on this data structure.

The core of our method relies in the algorithm for computing the successor of a class:

```
M' = M - t_f * t_f^*  
If (the current domain D is encoded by a DBM AND condition 3.2 is NOT satisfied) then  
    Make the DBM computation of [14] giving D'  
else  
    Make the polyhedral computation, giving D'  
end If  
If (D' can be encoded by a DBM) then  
    Encode D' by a DBM  
end If  
return (M', D')
```

**Algorithm 1:** Mixed method for computing the next class (entry parameter: current class $C = (M, D)$ and fired transition $t_f$; result: next class $C' = (M', D')$)
The method for computing the list of firable transitions from a class $C = (M, D)$ is the same for classes in a DBM form and classes in a more general polyhedral form. Checking if active transition $t_i$ is firable is performed as follows: for all active transitions, $t_j, j \neq i$, we add constraints $\theta_i \leq \theta_j$ to the current domain $D$ and then check if the resulting domain is empty or not. If not, that means that transition $t_i$ is firable from class $C = (M, D)$. The only difference is the nature of the domain we manipulate: Either a DBM, or a more general polyhedron.

The classical state class graph method presented in section 2, the DBM-over-approximation introduced by Lime et al. in [14] and our improved algorithm have been implemented in Romeo [9], an analysis tool for Time Petri Nets developed at IRCCyN. While DBM manipulation uses a home-made library, polyhedral manipulation is done thanks to New Polka, a polyhedral library [12].

In the next part, we compare the results given by the three methods. Our mixed method with the results obtained thanks to the general polyhedral computation and the DBM-over-approximation.

5 Experimental results

To illustrate the merits of our work, we introduce a benchmark that compares the efficiency, in terms of computation speed, of the DBM-over-approximation proposed by Lime et al. in [14], the classical polyhedral computation and our mixed algorithm. All three methods have indeed been implemented in Romeo [9], an analysis tool for Time Petri Nets developed at IRCCyN. We have executed the different algorithms on examples coming from real-time systems. The main results are summarized in table 1: We give the number of nodes and transitions of the resulting state class graph and the computation duration on a PowerPC G4, 1.33 GHz, 1GB RAM.

NA (for Not Available) means that the computation could not yield a result on the machine used. For the DBM-over-approximation, NA means that the overapproximate state space leads to an infinite number of marking whereas the Scheduling-TPN is bounded.

For sure, when there are only a few classes which can be turned into a DBM, our mixed method is less efficient and may be a little less fast than the original polyhedral method (because of the test to check if the resulting polyhedron can be written like a DBM). But for larger systems, we observe a significative gain of time when computing the exact state class graph with our mixed method. This is illustrated by examples 4, 5 or 6. Moreover, fully polyhedral computation can sometimes lead to memory overflow while our
mixed method performs the computation without any difficulty (Examples 3 and 11).

A first conclusion is that for all systems of practical interest, our mixed method is far more efficient than the general fully polyhedral method at computing the exact state space of a Scheduling-TPN.

In general, the DBM-approximation is, unsurprisingly, faster than the exact computation. That is not any more the case when the number of states added by the DBM-over-approximation becomes important. This extreme case appears on examples 9, 10 and 11: The states added by the DBM-over-approximation lead to an infinite number of marking whereas the Scheduling-TPN is bounded. However exact state-space computation on bounded Scheduling-TPNs does not necessarily terminate (since the accessibility problem is undecidable [3]) and the DBM over-approximation can, in this case, make it possible to obtain a finite (but approximate) abstraction of the state space as one can see on example 12.

6 Conclusion

In this paper, we studied a criterion that allows us to know a priori if a non-DBM polyhedral form will appear in the domain of a state class of a TPN with stopwatches, when computing the successor of a state class. We proved a necessary and sufficient condition in order to determine if the DBM-over-approximation relaxes constraints compared to the exact computation.
Starting from this condition, we proposed an efficient algorithm for computing the exact state class graph of a Scheduling-TPN. Tests show that our algorithm is for all systems of practical interest better (in terms of execution time and memory) than the classical approach at computing the exact state-space.

In particular, it allows us to compute the state class graph of some Scheduling-TPNs, for which the memory consumption of the fully polyhedral algorithm is too high and for which the DBM-over-approximation introduces an infinite number of markings (and would anyway compute only an approximate state-space).

It is very interesting to note that, while it allows us to check timed properties by itself (by the use of observers for instance), it may also act as a (slower but still efficient) replacement for the DBM over-approximation in the methods of [5] and [15], in the cases when the DBM over-approximation introduces an infinite number of markings while the net is actually bounded and prevents these methods to yield results.

Now, improvements can also be made on the over-approximation method. The DBM-over-approximation proposed by Lime et al. [14] is obviously too coarse for some applications, and we thus need to refine it. Future work also include the investigation of discrete semantics which provide under-approximations but which transpose the problem of verification to finite state-spaces.

References


