On Parametric DBMs and Their Applications to Time Petri Nets

Loriane Leclercq(✉), Didier Lime, and Olivier H. Roux

Nantes Université, École Centrale Nantes, CNRS, LS2N, UMR 6004, 44000 Nantes, France
{Loriane.Leclercq,Didier.Lime,Olivier-h.Roux}@ec-nantes.fr

Abstract. In the early stages of development, parameters are useful to verify timed systems that are not fully specified and symbolic algorithms or semi-algorithms are known to compute (integer) values of the parameters ensuring some properties. They rely on the representation of the reachable state-space as convex polyhedra. In the case of Parametric Timed Automata or Parametric Time Petri Nets (PTPN), those convex polyhedra have a special form that can be represented with a dedicated data structure called *Parametric Difference Bound Matrices* (PDBM).

Surprisingly few results are available for PDBMs, so we take a fresh look at them and show how they can be computed efficiently in PTPNs. In the process, we define tropical PDBMs (tPDBMs) that generalize the original definition and allows, in contrast to that former definition, to avoid splitting the represented polyhedra. We argue in particular that while tPDBMs are more costly to handle than their split counterpart, they allow for better convergence.

We have implemented both versions in Roméo, our tool for model-checking time Petri nets, and we compare the performance of the different polyhedron and (t)PDBM-based representations of symbolic states on several classical examples from the literature.

Keywords: Time Petri nets · state classes · parameters · Difference Bound Matrices

1 Introduction

When designing timed systems, the use of parameters for some durations (e.g. timeouts, periods, execution times, etc.) allows for early verification when durations are not yet fully specified or measured and provides admissible values for those parameters such that properties of interest are satisfied. Parametric Timed Automata (PTA) [3] and Parametric Time Petri Nets (PTPNs) [16] are classical formalisms for such parametric timed systems. They have a great expressiveness, but this comes at the price of the decidability of most – if not all – non-trivial

This work has been partially funded by ANR projects ProMiS ANR-19-CE25-0015 and BisoUS ANR-22-CE48-0012.

J. Hillston et al. (Eds.): QEST+FORMATS 2024, LNCS 14996, pp. 107–124, 2024.
https://doi.org/10.1007/978-3-031-68416-6_7

properties in the general case, including the mere existence of values for the parameters such that some discrete configuration (state or marking) is reachable [2]. Nonetheless, symbolic semi-algorithms are available [4,5,16] and some of them have been successfully used to do verification in realistic case-studies [21]. In some restricted cases, e.g., when parameters are integer-valued and bounded a priori, their termination can also be guaranteed [16].

Otherwise, semi-algorithms were designed, in particular for some extensions of PTPNs [19,23]. All those symbolic (semi-)algorithms rely on the use of convex polyhedra to represent the reachable state-space of the models. It has already been noted in [15] that in the case of PTAs (but the observation trivially extends to PTPNs), those convex polyhedra have a particular form: all constraints involve only one clock, or the difference of two clocks, as in the non-parametric case, but they are compared to linear expressions on parameters (with integer coefficients) instead of simply integers. The corresponding polyhedra are thus been called *parametric zones* as a reference to the timed systems.

In [15], this observation has been used to propose an alternative representation of parametric zones, extending the classical *Difference Bound Matrices* (DBMs) [10,14], replacing the integer coefficients in the matrix with arbitrary linear expressions on parameters, giving *Parametric DBMs*. They implemented the four basic operations needed for PTAs model-checking: adding guards, canonicalization, resetting clocks and computing time successors. Since the same clock (or clock difference) can be constrained by two or more incomparable linear expressions on parameters, this requires splitting the parametric zone by determining the constraints on parameters for which each of the linear expressions is smaller than all the others. These parametric DBMs have been shown to be more efficient on several examples of PTAs, in particular the class of (L/U)-PTAs.

In the non-parametric case, the use of DBMs is very suitable to finitely represent *state classes*, providing a finite representation of the infinite state space of TPNs in a *State Class Graph (SCG)* [8,10]. The state classes that compose this SCG can be computed efficiently [11,12] and all operations computed on DBMs [6,7].

We start form this work and extend it in three main directions:

1. we consider tropical PDBMs (tPDBMs), an extension of PDBMs to minimums, and use them for PTPNs instead of PTAs, taking advantage of the state class abstraction of [8,10]. More precisely, we extend to the parametric setting an algorithm from the literature that computes the successor of a state class very efficiently [11,12];
2. we show how we can compute state classes with or without splitting;
3. we have implemented these algorithms in the tool Roméo [20] and we compare on three different case-studies from the literature its performance when using general polyhedra, (t)PDBMs with and without splitting.

The rest of this article is organized as follows: In Sect. 2 we recall the basic definitions, including the formalism of PTPNs. In Sect. 3, we give the algorithms to compute parametric state classes using parametric DBMs (with splitting)

and tropical PDBMs (without splitting). In Sect. 4, we evaluate our algorithms experimentally, and finally we conclude in Sect. 5.

2 Definitions

2.1 Preliminaries

We denote the set of natural numbers (with 0) by $\mathbb{N}$, the set of integers by $\mathbb{Z}$, the set of rational numbers by $\mathbb{Q}$, the set of real numbers by $\mathbb{R}$, and by $\mathbb{R}_{\geq 0}$ its restriction to non-negative numbers. For a finite set X, we denote its size by $|X|$.

Given a set X, we denote by $\mathcal{I}(X)$, the set of non necessarily bounded real intervals that have their finite end-points in X.

Given sets V and X, a V-valuation (or simply valuation when V is clear from the context) of X is a mapping from X to V. We denote by V^X the set of V-valuations of X. When X is finite, given an arbitrary fixed order on X, we often equivalently consider V-valuations as vectors of $V^{|X|}$.

In all this article, we consider a finite set $\mathbb{P}$ of *parameters*, that are symbolic constants that have a fixed unknown value.

2.2 Tropical Semi-ring

Definition 1. *The* parametric tropical semi-ring *is the semi-ring* $\mathbb{T} = (\mathbb{P} \cup -\mathbb{P} \cup \mathbb{Z} \cup \{+\infty\}, \widetilde{\min}, \widetilde{+})$*, where* $-\mathbb{P}$ *contains the inverse of each parameter with respect to* $\widetilde{+}$*.*

Compared to the classical tropical semi-ring [22], we have just added the (finite) set of parameters (and their inverse by $\widetilde{+}$). It is easy to see that the resulting set retains the semi-ring structure.

Definition 2 (Parametric expressions and constraints). *A* parametric tropical expression *is a value in* $\mathbb{T}$*. Equivalently, it is a term generated by grammar* $e ::= e_1 \mathbin{\widetilde{+}} e_2 \mid \widetilde{\min}(e_1, e_2) \mid a \mid p \mid -p$ *where* $p \in \mathbb{P}$ *is a parameter,* $a \in \mathbb{Z} \cup \{+\infty\}$ *is a constant and* e_1, e_2 *are tropical expressions.*

A parametric linear expression *is a parametric tropical expression with no* $\widetilde{\min}$ *operator.*

A parametric tropical constraint *c is a term of the form* $e \prec 0$ *with* $\prec \in \{<, \leq, >, \geq\}$.

A parametric linear constraint *is a parametric tropical constraint* $e \prec 0$ *where e is a parametric linear expression.*

Parametric tropical expressions are related to tropical polynomials, but instead of defining them as functions, we define the action of a parameter valuation on expressions: by replacing each parameter in expression e by its value given by a $\mathbb{Q}$-valuation v_p on $\mathbb{P}$, we obtain a rational number that we denote by $v_p(e)$.

Similarly, for a parametric tropical constraint $c = (e \prec 0)$, we can define $v_p(c)$ as the boolean value $v_p(e) \prec 0$.

2.3 Parametric Time Petri Nets

A time Petri net (TPN) is a Petri net with time intervals associated with each transition. Following [18], we propose a slightly different semantics than the one commonly used, in which firing dates are decided at the moment when transitions are newly enabled. In the following definitions we use and adapt the notations of [18]. We thus define PTPNs as an extension of TPNs in which interval bounds of transitions can be parametric tropical expressions.

Definition 3 (Parametric time Petri net). *A* parametric time Petri net *(PTPN) is a tuple* $\mathcal{N} = (P, T, \mathbb{P}, F, m_0, I_s)$ *where:*

- P *is a finite non-empty set of* places*,*
- T *is a finite set of* transitions *such that* $T \cap P = \emptyset$*,*
- $\mathbb{P}$ *is a finite set of* parameters*;*
- $F \subseteq (P \times T) \cup (T \times P)$ *is the* flow function*,*
- $m_0 \subseteq P$ *the* initial marking*,*
- $I_s : T \to \mathcal{I}(\mathbb{T})$ *is the* static firing interval *function.*

We denote by $\downarrow I_s$ (resp $\uparrow I_s$) the lower (resp. upper) bound of I_s. They can be parametric tropical expressions, like all the end-points of intervals in $\mathcal{I}(\mathbb{T})$.

As before, for any parametric tropical interval I and any parameter $\mathbb{Q}$-valuation v_p, we denote by $v_p(I)$ the interval obtained by replacing each parameter in the end-points by its value.

Clearly instantiating every static interval I_s of a PTPN $\mathcal{N}$ by the parameter valuation v_p leads to a TPN without parameters. We say that v_p is a *compatible* valuation for $\mathcal{N}$ if all static intervals are not empty in $v_p(\mathcal{N})$. This is characterized by the tropical constraint $\forall t.\ \downarrow I_s(t) \leq \uparrow I_s(t)$.

Places of a Petri net can contain *tokens*. A *marking* is then usually an $\mathbb{N}$-valuation of P giving the number of tokens in each place.

Remark 1. The crux of this article concerns the representation of time so, for the sake of simplicity, we consider only *safe* nets, i.e., nets in which there is always at most 1 token in each place (and thus all arcs must have weight 1). There would be no difficulty generalizing the results to *bounded* nets in which there exists a maximal number of tokens in each place.

Since we consider safe nets, we define a marking as the set of the places of P containing a token. We say those places are *marked*.

Given a transition t, we define the sets of its input places $\mathsf{Pre}(t) = \{p \mid (p,t) \in F\}$ and of its output places $\mathsf{Post}(t) = \{p \mid (t,p) \in F\}$.

Definition 4 (Enabled and persistent transitions). *A transition* t *is said to be* enabled *by marking* m *if all its input places are marked:* $\mathsf{Pre}(t) \subseteq m$*. A transition* t *is said to be* persistent *during the firing of a transition* t_f *from marking* m *if it is not fired and still enabled when removing tokens from the input places of* t_f*:* $t \neq t_f \wedge \mathsf{Pre}(t) \subseteq m \backslash \mathsf{Pre}(t_f)$*. We say that* t *is* newly enabled *by firing* t_f *from* m*, if* t *is enabled after the firing of* t_f *but not persistent. We denote by* $\mathsf{en}(m)$*,* $\mathsf{pers}(m, t_f)$ *and* $\mathsf{newen}(m, t_f)$ *respectively the sets of enabled, persistent and newly enabled transitions.*

Definition 5 (States and semantics of a PTPN). *A* state *of a PTPN is a triplet $s = (m, v_p, \theta)$ with $m \subseteq P$ a marking, $v_p \in \mathbb{Q}^{\mathbb{P}}$ a parameter valuation and $\theta : T \to \mathbb{R}_{\geq 0} \cup \{\bot\}$ a function that associates a firing date with every transition $t \in \mathsf{en}(m)$ and $\bot$ to all other transitions. We will use θ_i to denote $\theta(t_i)$.*

The semantics of a PTPN is a Timed Transition System *$(S, S_0, \Sigma, \to)$ with:*

- *$S \subseteq 2^P \times \mathbb{Q}^{\mathbb{P}} \times \mathbb{R}_{\geq 0}^T$ the set of all possible states,*
- *$S_0 = \{(m_0, v_p, \overline{\theta}_0) | \forall t \in T, v_p(I_s(t)) \neq \emptyset \wedge$ if $t \in \mathsf{en}(m_0), \theta_0(t) \in v_p(I_s(t)), \theta_0(t) = \bot$ otherwise$\}$ the set of initial states, all possible states with the initial marking,*
- *a labelling alphabet Σ divided into two types of letters: $t_f \in T$ and $d \in \mathbb{R}_{\geq 0}$,*
- *the transition relation between states $\to \subseteq S \times \Sigma \times S$ and, $(s, a, s') \in \to$, denoted by $s \xrightarrow{a} s'$:*
  - *either $(m, v_p, \theta) \xrightarrow{t_f} (m', v_p, \theta')$ for $t_f \in T$ when:*
    1. *$t_f \in \mathsf{en}(m)$ and $\theta_f = 0$*
    2. *$m' = (m \backslash \mathsf{Pre}(t_f)) \cup \mathsf{Post}(t_f)$*
    3. *$\forall t_k \in T, \theta'_k \in v_p(I_s(t_k))$ if $t_k \in \mathsf{newen}(m, t_f)$, $\theta'_k = \theta_k$ if $t_k \in \mathsf{pers}(m, t_f)$, and $\theta'_k = \bot$ otherwise*
  - *or $(m, v_p, \theta) \xrightarrow{d} (m, v_p, \theta')$ when: $d \in \mathbb{R}_{\geq 0} \backslash \{0\}$, $\forall t_k \notin \mathsf{en}(m), \theta_k = \bot$, $\forall t_k \in \mathsf{en}(m), \theta_k - d \geq 0$, and $\theta'_k = \theta_k - d$.*

A *run* in the semantics of a TPN is a possibly infinite sequence $s_0 a_0 s_1 a_1 s_2 a_2 \cdots$ such for all i, $s_i \xrightarrow{a_i} s_{i+1}$ and $s_0 \in S_0$. Note that v_p is not modified by the transition function, therefore it remains unchanged throughout a run.

2.4 Parametric Reachability Problems

In this article, we consider parametric reachability problems. The simplest one is the existential problem: given a PTPN $\mathcal{N}$ and a marking m, does there exist a valuation v_p of the parameters of $\mathcal{N}$ such that m is reachable in $v_p(\mathcal{N})$. This problem is undecidable [2] but some semi-algorithms were designed [4,5,16].

More difficult is the *synthesis problem*: given a PTPN $\mathcal{N}$ and a marking m, compute all parameter valuations v_p of the $\mathcal{N}$ such that m is reachable in $v_p(\mathcal{N})$.

2.5 Parametric State Classes

As for TPNs, the number of states from a PTPN is not finite in general because of the density of static intervals. There are several finite representations abstracting the state space of a TPN using various methods and one of them is the state class graph [8,10]. One of its benefits is to be finite as long as the TPN is bounded, i.e. the number of tokens in each place is bounded (by 1 in the case of safe nets). We now review the classical construction in the parametric case.

Definition 6 (Parametric state class). *Let $\sigma = t_1 ... t_n$ be a sequence of transitions. The* parametric state class K_σ *is the set of all states obtained by firing σ in order, with all possible delays* before *each fired transition together with all possible valuations of parameters. Clearly, all states in K_σ share the same marking m, and so we write $K_\sigma = (m, D)$ where D, called the* firing domain*, is the union of all possible firing dates and parameters. A point in this domain is (v_p, θ) with v_p a valuation of the parameters $\mathbb{P}$ and θ the firing dates of enabled transitions.*

With an arbitrary order on transitions and parameters, and ignoring $\perp$ values, such valuations can be seen as points in $\mathbb{R}_{\geq 0}^{|\mathsf{en}(m)|+|\mathbb{P}|}$. We will therefore consider domains as subsets of $\mathbb{R}_{\geq 0}^{|\mathsf{en}(m)|+|\mathbb{P}|}$. And as we will see in Sect. 3, firing domains are a special kind of convex polyhedra in that space. Let $(m, v_p, \theta) \in K_\sigma$, we will denote by $v_p(K_\sigma)$ the non-parametric state class obtained by taking every firing valuation that is compatible with v_p: $v_p(K_\sigma) := \{(m, \theta) \mid (m, v_p, \theta) \in K_\sigma\}$.

We can now naturally extend the notions of enabled, persistent, and newly enabled transitions to state classes: $\mathsf{en}((m, D)) = \mathsf{en}(m)$, $\mathsf{newen}((m, D), t_f) = \mathsf{newen}(m, t_f)$, and $\mathsf{pers}((m, D), t_f) = \mathsf{pers}(m, t_f)$.

A direct consequence of Definition 6 is a match between states of the PTPN reachable by firing of a sequence of transitions σ and the class K_σ; and an equivalence between state classes from the PTPN and from its instantiation.

Proposition 1. *Let K_σ be a parametric state class from a PTPN $\mathcal{N}$ containing a compatible parameter valuation v_p, and K'_σ the corresponding state class from the instantiated TPN $v_p(\mathcal{N})$. Then we have: $v_p(K_\sigma) = K'_\sigma$.*

The proof is straightforward because these state classes contain all states reachable by firing σ and instantiating at the level of firing interval in the PTPN, or later in state classes, does not change the reachable states obtained.

In the non-parametric case, the classical algorithm to compute state classes from [8] allows us to compute $K' = (m', D')$ from $K = (m, D)$ by firing firable transition t_f. In this construction, D' is not empty if and only if there exists θ in D such that for all $t_i \in \mathsf{en}(m)$, $\theta_i \geq \theta_f$. We then call t_f *firable* from (m, D). This extends to parametric state classes as shown in [19] for cost PTPN.

Remark 2. The embedding of parameters into the firing domain allows for the domain of parameters $\mathbb{P}$ to be directly reduced to the valuations that make the class reachable, in particular when checking whether a transition is firable or not (see [19] for an extension of the classical successor algorithm).

It is well-known that the non-parametric state classes can be represented and computed using a special kind of convex polyhedra encoded in the efficient data structure called *difference bound matrix* (DBM) [8,14], but this is not true anymore in the parametrized case. Hence the use of an extension of the Parametric DBMs from [15], that will be detailed in Sect. 3.

Definition 7. *Starting from K_ϵ, we construct an infinite directed tree (labeled by fired transitions) by inductively computing successors by firable transitions. The* Parametric state class graph $\mathcal{G}$ *is obtained by quotienting this tree with the equality relation on state classes (same marking, and same firing domain).*

2.6 An Efficient State Class Successor for TPNs

Without parameter, an efficient way to compute successor classes in quadratic time, taking advantage of the particular form of DBMs, is given in [11,12]. We recall the construction in Algorithm 1.

Algorithm 1. Successor (m', D') of (m, D) by firing firable transition t_f

$m' \leftarrow (m \backslash \mathsf{Pre}(t_f)) \cup \mathsf{Post}(t_f)$; $D'[o,o] = 0$
for all $t_i \in \mathsf{en}(m')$ **do**
  $D'[i,i] = 0$
  if $t_i \in \mathsf{newen}(m, t_f)$ **then**
    $D'[i,o] = -\downarrow I_s(t_i)$; $D'[o,i] = \uparrow I_s(t_i)$
  else
    $D'[i,o] = 0$; $D'[o,i] = D[f,i]$
    for all $t_j \in \mathsf{en}(m)$ **do** $D'[i,o] = \min(D'[i,o], D[i,j])$
for all $t_i \in \mathsf{en}(m')$ **do**
  for all $t_j \in \mathsf{en}(m'), i \neq j$ **do**
    if $t_i \in \mathsf{newen}(m, t_f)$ **or** $t_j \in \mathsf{newen}(m, t_f)$ **then** $D'[i,j] = D'[i,o] + D'[o,j]$
    else $D'[i,j] = \min(D[i,j], D'[i,o] + D'[o,j])$
return (m', D')

3 Tropical PDBM

This section is dedicated to showing that the convex polyhedra used during the previous procedure have a particular form, called *parametric zones* in [16]: all constraints involve only one firing date or a difference between two firing dates, as in the non-parametric case, but they are compared to (minimum of) linear expressions on parameters (with integer coefficients) instead of simple integers.

In [15], the authors propose an alternative representation of parametric zones, extending the classical DBMs and called parametric DBMs with parametric linear expressions as coefficients. We extend this to parametric tropical expressions:

Definition 8. *A* Tropical Parametric Difference Bound Matrix (tPDBM) *over a set of variables* $\{\theta_1, \ldots, \theta_n\}$ *is a square matrix* M_c *of size* $n+1$ *such that for all integers* $i, j \in [0, n]$, $M_c[i,j]$ *is a pair* $(e_{ij}, \prec_{ij})$ *where* e *is a parametric tropical expression and* $\prec_{ij} \in \{<, \leq\}$. *Each* $M_c[i,j]$ *is called a* bound *of* M_c.

We can reduce any parametric tropical expression to the minimum of a set of parametric linear expressions using basic properties of the tropical semi-ring: operator $\widetilde{+}$ distributes over the $\widetilde{\min}$ operator.

Lemma 1. $\widetilde{\min}(f,g) \mathbin{\widetilde{+}} \widetilde{\min}(h,k) = \widetilde{\min}\big(f \mathbin{\widetilde{+}} h, g \mathbin{\widetilde{+}} h, f \mathbin{\widetilde{+}} k, g \mathbin{\widetilde{+}} k\big)$.

We can associate a conjunction of parametric linear constraints $\mathcal{P}(M_c)$ to any tPDBM M_c over set of variables $\{\theta_1, \ldots, \theta_n\}$: assuming θ_0 is a special variable whose value is always 0, and $\forall i, j, M_c[i,j] = \min(\ell_{ij}^1, \ldots, \ell_{ij}^p)$, where all ℓ_{ij}^k are parametric linear expressions[1], we have: $\mathcal{P}(M_c) = \bigwedge_{i,j} \bigwedge_k (\theta_i - \theta_j \prec_{ij} \ell_{ij}^k)$.

Given a parameter valuation v_p, we say that M_c is *satisfiable* by v_p if and only if $v_p(\mathcal{P}(M_c)) := \bigwedge_{i,j} \bigwedge_k (\theta_i - \theta_j \prec_{ij} v_p(\ell_{ij}^k))$ is not empty.

Usually, we consider one θ variable for each enabled transition in a given marking, just like in classical state classes.

The parameter constraints will be stored in a set separated from the guards.

Definition 9. *A* constrained tPDBM *is a set (P_c, M_c) with P_c a conjunction of linear constraints on parameters, and M_c a tPDBM.*

As before we can associate a conjunction of linear constraints $\mathcal{P}((P_c, M_c))$ to any constrained tPDBM (P_c, M_c) (overloading the notation):

$$\mathcal{P}((P_c, M_c)) = P_c \wedge \mathcal{P}(M_c)$$

We say that (P_c, M_c) is *satisfiable* if and only if $\mathcal{P}((P_c, M_c))$ is not empty.

We denote by $[\![c]\!]$ the set of valuations satisfying a given a parametric tropical constraint c. We extend the notation to conjunctions of tropical constraints and finally to tPDBMs $[\![(P_c, M_c)]\!] := [\![\mathcal{P}((P_c, M_c))]\!] = [\![P_c]\!] \cap [\![\mathcal{P}(M_c)]\!]$. In particular $[\![P_c]\!] \neq \emptyset$ iff the constraints are satisfiable.

Given an arbitrary order on parameters and θ variables, $[\![(P_c, M_c)]\!]$ can be seen as a convex polyhedron just like the firing domains of state classes, and in Sect. 3.2 we will show how we can compute those domains as tPDBMs.

From now on, to simplify the presentation, we will ignore the fact that constraints might be strict in the net and in tPDBMs (except when splitting using PDBMs). We will therefore only use $\leq$ operators in all the tPDBMs and the constraints, but strict constraints can be handled exactly as in [15].

3.1 Reachability Algorithm

We use a classic forward reachability algorithm to explore reachable states in order to know if a given marking is reachable, and if it is, to give the possible parameter domain. The only difference is in the criterion for a firable transition. In fact without parameter it is easy to check whether a transition is firable. However, in the parametric case, this may lead to new parameter constraints. Suppose that a parametric class C is represented as a constrained tPDBM (P_c, M_c). Let $t_f \in \mathsf{en}(C)$, we will compute the new parametric constraints to fire t_f with: $\mathsf{fir_p}(C, t_f) := P_c \cap \bigcap_{i \neq f, t_i \in \mathsf{en}(C)} \{M_c[f,i] \geq 0\}$. We will use this new parameter constraints set to compute the successor using Algorithm 2.

[1] This is always possible, by distributing $\widetilde{+}$ and by adding as many $+\infty$ expressions in the resulting $\widetilde{\min}$ as needed.

Algorithm 2. Reachability of a marking m_t from a class $C_0 = (m_0, D_0)$

```
1: Seen ← ∅ ; New ← {C_0}
2: for all C = (m, (P_c, M_c)) ∈ New do
3:    New ← New\{C} ; Seen ← Seen ∪ {C}
4:    for all t_f ∈ en(C) do
5:       if [[fir_p(C, t_f)]] ≠ ∅ then
6:          C' = (m', (P'_c, M'_c)) ← Next_p(C, t_f)
7:          if m' = m_t then return true
8:          if C' ⋢ Seen then New ← New ∪ C'
9: return false
```

3.2 Successor Operator

In this section, we use a state class $C = (m, D)$ from a TPN $\mathcal{N}$ and $C_p = (m, (P_c, M_c))$ a parametric state class from a PTPN $\mathcal{N}_p$, sharing the marking m.

Definition 10. *The* $\mathsf{Next}(C, t_f)$ *operator (resp. its parametric version* $\mathsf{Next_p}(C_p, t_f)$*) is the set of states reachable from* C *(resp.* C_p*) by firing transition* t_f*.*

$$\mathsf{Next}(C, t_f) = \{s \mid \exists s' \in C, s' \xrightarrow{t_f} s \in \mathcal{N}\}$$
$$\mathsf{Next_p}(C_p, t_f) = \{s_p \mid \exists s'_p \in C_p, s'_p \xrightarrow{t_f} s_p \in \mathcal{N}_p\}$$

We will compute these sets of states using $\mathsf{Succ}(C, t_f)$, the successor domain computed with Algorithm 1, and an extension of the construction to the parametric case $\mathsf{Succ_p}((m, M_c), t_f)$, replacing all $+$ and min operators by $\widetilde{+}$ and $\widetilde{\mathsf{min}}$.

Proposition 2. **(Next() and Next$_\mathsf{p}$() computation).**

$$\mathsf{Next}(C, t_f) = (m', [\![\mathsf{Succ}(C, t_f)]\!]) \tag{1}$$
$$\mathsf{Next_p}(C_p, t_f) = \left(m', [\![(\mathsf{fir_p}(C_p, t_f), \mathsf{Succ_p}((m, M_c), t_f))]\!]\right) \tag{2}$$

with $m' = (m \backslash \mathsf{Pre}(t_f)) \cup \mathsf{Post}(t_f)$.

The non-parametric case (1) was proved in [11].

As a direct consequence of the constructions, we get that for a given constrained tPDBM (P_c, M_c) corresponding to a state class C_p with marking m:

Proposition 3. $\forall v_p \in [\![\mathsf{fir_p}(C_p, t_f)]\!]$. $\mathsf{Succ}((m, v_p(M_c)), t_f) = v_p(\mathsf{Succ_p}((m, M_c), t_f))$.

The proof is in Appendix B and uses the formulas from Algorithm 1 to show an equivalence between coefficients from both matrices after instantiation.

Correctness of the firable condition is given by the following lemma, whose proof is direct by construction of $\mathsf{fir_p}(C_p, t_f)$.

Lemma 2. *From a state class C_p, a transition t_f is firable iff $[\![\mathsf{fir}_{\mathsf{p}}(C_p, t_f)]\!] \neq \emptyset$.*

Finally, the proof of Proposition 2 is in Appendix A. It uses Propositions 1 and 3, Lemma 2 and the fact that an instantiated state is included in an instantiated state class iff this parametric state is in the parametric state class.

Corollary 1. *For all non-empty tPDBMs (P_c, M_c), $[\![P_c]\!]$ is equal to the projection on parameters of $[\![(P_c, M_c)]\!]$.*

The proof uses the fact that P_c corresponds to the conjunction of all firing constraints needed to fire the sequence of transition leading to (P_c, M_c).

From the previous results, it follows directly that domains can be computed as constrained tPDMs and that their associated polyhedron $\mathcal{P}(P_c, M_c)$ is clearly a parametric zone.

Proposition 4. *The firing domains of parametric state classes can be represented by parametric zones.*

3.3 Inclusion

Constrained tPDBMs admit a canonical form, in the sense that given a tPDBM (P_c, M_c), $\forall v_p \in [\![P_c]\!]$, $v_p(M_c)$ is a DBM in canonical form.

This canonical form forces the bounds to be the tightest, so we can directly compare these canonical forms for the inclusion operation.

First, we say that a constrained tPDBM is included in another if their associated polyhedra satisfy the same inclusion. We can however compute inclusion of tPDBMs in a much simpler way than inclusion of polyhedra (recall we ignore strict constraints for clarity).

Proposition 5. *For two constrained tPDBMs $D^1 = (P_c^1, M_c^1)$ and $D^2 = (P_c^2, M_c^2)$ of same size with $\forall i, j.\ \prec_{ij}^1 = \prec_{ij}^2 = \leq$, $[\![D^1]\!] \subseteq [\![D^2]\!] \iff [\![P_c^1]\!] \subseteq [\![P_c^2]\!]$ and $\forall v_p \in [\![P_c^1]\!], \forall i, j, v_p(M_c^1[i,j]) \leq v_p(M_c^2[i,j])$.*

The proof is in Appendix C. Checking $[\![P_c^1]\!] \subseteq [\![P_c^2]\!]$ is easy since we have two convex polyhedra. The second condition is a bit trickier. If we had PDBMs, with only parametric linear expressions, this could be done by solving a linear program, as noted in [15]: $\forall v_p \in [\![P_c^1]\!], v_p(\ell) \leq v_p(\ell')$ can be checked by minimizing $\ell' - \ell$ over $[\![P_c^1]\!]$ and checking if the result is non-negative.

With parametric tropical expressions, we can do something similar but more complex: suppose we want to check $\forall v_p \in [\![P_c^1]\!], v_p(\widetilde{\mathsf{min}}(\ell_1, \ell_2)) \leq v_p(\widetilde{\mathsf{min}}(\ell_1', \ell_2'))$ to keep things simple, but this generalizes easily. First, this is equivalent to $\forall v_p \in [\![P_c^1]\!], (v_p(\widetilde{\mathsf{min}}(\ell_1, \ell_2)) \leq v_p(\ell_1')$ and $v_p(\widetilde{\mathsf{min}}(\ell_1, \ell_2)) \leq v_p(\ell_2'))$, which is the same as $\forall v_p \in [\![P_c^1]\!], v_p(\widetilde{\mathsf{min}}(\ell_1, \ell_2)) \leq v_p(\ell_1')$ and $\forall v_p \in [\![P_c^1]\!], v_p(\widetilde{\mathsf{min}}(\ell_1, \ell_2)) \leq v_p(\ell_2')$.

Second, $\forall v_p \in [\![P_c^1]\!], v_p(\widetilde{\mathsf{min}}(\ell_1, \ell_2)) \leq v_p(\ell_1')$ is equivalent to $\forall v_p \in [\![P_c^1]\!] \cap (\ell_1 \leq \ell_2), v_p(\ell_1) \leq v_p(\ell_1')$ and $\forall v_p \in [\![P_c^1]\!] \cap (\ell_1 > \ell_2), v_p(\ell_2) \leq v_p(\ell_1'))$ and similarly for ℓ_2'. And thus we have reduced the problem to checking a set of similar problems on parametric linear expressions, which can be done as described above.

3.4 Splitting and Classical PDBMs

Instead of keeping tPDBMs with parametric tropical expressions using $\widetilde{\min}$ operators, we could also split all those $\widetilde{\min}$ to obtain only parametric linear expressions and thus simple PDBMs, like in [15]. This is basically what we have done above when partitioning P_c^1 with constraints $(\ell_1 \leq \ell_2)$ and $(\ell_1 > \ell_2)$, except the idea here is to do it once and for all and, when computing the successor of a PDBM by firing some transition, to compute copies of the PDBM for each split. Of course, each copy produced by splitting a $\widetilde{\min}$ will then be split according to any other constraint with a $\widetilde{\min}$, etc. leading to possibly a lot of copies; but each of them is simpler to handle, in particular for inclusion checking, because it contains only parametric linear expressions. More details can be found in [15] for this approach. Note that it also has an impact on the convergence of the reachability algorithm as will be shown in Sect. 4.4.

4 Case Studies

We report here on experiments using the tool Roméo [20], on an i9 MacBook Pro, with 32 Gb of RAM. Roméo and all models are available at https://romeo.ls2n.fr/.

4.1 Fischer's Mutual-Exclusion Protocol

Fischer's mutual-exclusion protocol is a well-known benchmark in the literature of real-time verification, introduced in [1,17].

The system consists of n identical processes which access a critical section by mutual exclusion. The processes communicate via a shared variable called `last`, which is equal to the id (with values between 1 and n) of the last process requesting access to the critical section, or 0 if the critical section is available and not requested. Process i behaves as follows: after remaining inactive for a certain amount of time, it checks whether the common resource is free (test `last` $= 0$) and, if it is, sets `last` to its id no later than B time units after it has tested `last` $= 0$. It then waits for a time greater than A, checks that `last` is still equal to i, and enters the critical section. If `last` is not equal to i (which means that another process has requested access in the meantime), process i must try again later. As in [24], A and B are parameters of the system. The correctness of the protocol depends on their values.

For n processes, mutual exclusion is checked with Roméo by the CTL formula $AG\Big(\big(\sum_{i=1}^{n} m(Critical[i])\big) \leq 1\Big)$ where $m(Critical[i])$ is the number of tokens in the place $Critical$ of the process i. The property holds iff $A > B$. A comparison of computation times and memory consumption for the mutual exclusion checking for one parameter (setting B to 1) and for 2 parameters is given in Table 1. DNF (Do Not Finish) means that the computation did not finish within 90 min.

Table 1. Results for Fisher's protocol for one (with $B = 1$) and 2 parameters

Nb. of par.	Nb. of proc.	General Polyhedron		Tropical PDBM		PDBM (with splitting)	
		Time (s)	Mem. (Mb)	Time (s)	Mem. (Mb)	Time (s)	Mem. (Mb)
1	4	1.6	16	**0.4**	**9**	1.2	50
1	5	8.2	71	**1.8**	**47.7**	5.2	260
1	6	49	367	**11**	**283**	26	1460
1	7	296	2155	**67**	**1745**	133	4293
1	8	DNF	DNF	**404**	**9616**	DNF	DNF
2	4	1.6	16	**0.6**	**12**	1	62
2	5	8	71	**3.4**	**61**	4.8	322
2	6	48	360	**23**	**350**	26	1740
2	7	305	**2086**	148	2104	**143**	9700
2	8	DNF	DNF	DNF	DNF	DNF	DNF

4.2 Level Crossing

We now use a classical case-study presented in [9]. It is a level crossing model with n tracks with independent trains, and a shared barrier protecting them, in order to obtain problems of increasing complexity. The model is made up of n train models (on their own track) and the level crossing barrier model. These models communicate via a shared variable counting the number of trains approaching or in the level crossing zone. For train i, the place modelling the fact that the train is on the level crossing is $On[i]$. The property to check is therefore $AG\Big(\big(\sum_{i=1}^{n} m(On[i])\big) \geq 1 \Rightarrow m(Closed) == 1\Big)$ where n is the number of trains. We have 3 parameters a, b and c representing respectively the minimum and maximum closing times of the barrier and the minimum duration between the approach of a train and its entry in the crossing area.

The corresponding model is recalled in Fig. 1 in Appendix D.

The property is true iff $c \in]0, 5]$, $a \in [0, 5[$, $b \in [0, 5[$, $c > b$, and $b \geq a$.

The results are given in Table 2, for two parameters (setting parameter c to 3), and for all three parameters.

Table 2. Results for Level crossing with 2 parameters ($c = 3$) and 3 parameters.

Nb. of Par.	Nb. of Trains	General Polyhedron		Tropical PDBM		PDBM (with splitting)	
		Time (s)	Mem. (Mb)	Time (s)	Mem. (Mb)	Time (s)	Mem. (Mb)
2	2	0.1	1.4	**0**	**0.6**	0.1	2
2	3	1.8	10	**0.6**	**4.9**	1.2	59
2	4	85	237	**26**	**89**	40.9	1445
2	5	DNF	DNF	**1242**	**2488**	DNF	DNF
3	2	0.1	1.3	**0**	**0.7**	0.1	2.9
3	3	5.1	24	**2.3**	**9.1**	8.9	252
3	4	665	953	**201**	**261**	945	1716
3	5	DNF	DNF	DNF	DNF	DNF	DNF

4.3 Producerconsumer

We now consider the classical producer-consumer problem described by E. W. Dijkstra [13]. The aim of this experiment is to measure the effect of the number of parameters as a function of the abstraction chosen.

We have 6 producers and 6 consumers. We will take a single parameter p for the producers such that producer i produces at date $10 + i * p$. And we will take one parameter c_i per consumer i such that consumer i consumes c_i time units starting from the moment something is available to consume. The size of the buffer is bounded.

To vary the number of parameters, we set c_i to 0 for a decreasing number of parameters. We also vary the buffer size between 2 and 4. As an example, for 6 parameters we obtain that the buffer is bounded by 4 iff $\big(p > 0 \wedge c_2 \geq 0 \wedge c_4 \geq 0 \wedge c_3 \geq 0 \wedge c_5 \geq 0 \wedge c_1 \geq 0 \wedge \big((4p - c_1 > 0) \vee (4p - c_2 > 0) \vee (4p - c_3 > 0) \vee (4p - c_4 > 0) \vee (4p - c_5 > 0)\big)$.

The results are given in Table 3.

Table 3. Results for Producer-consumer.

Nb. of par.	Size of buffer	General Polyhedron		Tropical PDBM		PDBM (with splitting)	
		Time (s)	Mem. (Mb)	Time (s)	Mem. (Mb)	Time (s)	Mem. (Mb)
3	2	0.7	**9**	0.5	**9**	**0.3**	30
4	2	0.9	11	0.5	**10**	**0.4**	36
5	2	1.5	18	**0.8**	**16**	3.2	84
6	2	8.7	66	**8.4**	**62**	315	871
7	2	**946**	639	1247	**623**	DNF	DNF
3	3	1.0	11	0.6	**10**	**0.4**	36
4	3	1.8	14	0.7	**12**	**0.6**	45
5	3	2.2	23	**1.2**	**20**	6.1	108
6	3	**22**	**111**	29	116	937	1404
7	3	DNF	DNF	DNF	DNF	DNF	DNF
3	4	1.5	**12**	0.8	14	**0.6**	50
4	4	1.7	**14**	0.9	15	**0.8**	57
5	4	2.4	**22**	**1.6**	24	6.8	123
6	4	**31**	**134**	47	136	DNF	DNF
7	4	DNF	DNF	DNF	DNF	DNF	DNF

4.4 Discussion

We can see that the method we propose, using tPDBMs (without splitting) is mostly better than the general polyhedron or the PDBM (with splitting) approaches, sometimes much better permitting to obtain results unattainable for the two other approaches. Even when it does not give the best results, its performance is nonetheless similar to the better ones.

In the Producer-consumer example, we see that when the number of parameters increases compared to the number of enabled transitions, the benefits of using PDBMs tend to decrease. This seems expected since then most operations are performed on polyhedra of similar dimensions in the generic polyhedra and PDBM approaches.

Another interesting thing is that the approach using general polyhedra performs often as well as the PDBM approach (and sometimes quite better for larger numbers of parameters) even though testing inclusion of PDBMs is the most efficient of the three approaches[2].

Of course, our PDBM computation is not directly comparable to the one in [15], even if the idea is similar, because we work on TPNs, with a different clock abstraction, namely state classes. Nonetheless, this similar performance can be at least partially explained by a less efficient convergence with PDBMs, which nullifies the gains from the much simpler individual inclusion tests.

Consider a simple net with three transitions enabled in parallel and with a parameter a: $t_1[3,5]$, $t_2[a,9]$, and $t_3[4,6]$, each t_i as an entering arc coming from a corresponding place p_i. With either the general polyhedron approach or the tPDBM approach, after firing t_1 then t_3, we obtain that only p_2 is marked and a single tPDBM: $D = \{t_2 \in [\max(0, a-6), 5] \wedge a \in [0,9]\}$. Similarly, after firing t_3 then t_1, we obtain again a single tPDBM: $D' = \{t_2 \in [\max(0, a-5), 5] \wedge a \in [0,9]\}$. It is then easy to see that $D' \subseteq D$.

With the splitting approach, after firing t_1 then t_3, we obtain that only p_2 is marked, and we have four PDBMs because of splitting: $D_1 = \{t_2 \in [0,5] \wedge a \in [0,3)\}$, $D_2 = \{t_2 \in [0,5] \wedge a \in [3,5)\}$, $D_3 = \{t_2 \in [0,5] \wedge a \in [5,6]\}$, and $D_4 = \{t_2 \in [a-6,5] \wedge a \in (6,9]\}$. Similarly, after firing the sequence t_3, t_1, we obtain three PDBMs because of splitting: $D'_1 = \{t_2 \in [0,5] \wedge a \in [0,4)\}$, $D'_2 = \{t_2 \in [0,5] \wedge a \in [4,5)\}$, and $D'_3 = \{t_2 \in [a-5,5] \wedge a \in [5,9]\}$.

Except for $D'_2 \subseteq D_2$, we do not have pairwise inclusions of the D'_i in the D_j, so here the exploration would continue when splitting, while it would have stopped when not splitting. Note that even if we merge D_1, D_2 and D_3 in some D_{123} PDBM, because they share the $t_2 \in [0,5]$ constraints, and similarly if we merge D'_1 and D'_2 into some D'_{12}, we would indeed have $D'_{12} \subseteq D_{123}$ but still not $D'_3 \subseteq D_4$ or $D'_3 \subseteq D_{123}$, and detecting this kind of merges could have a significant cost in general.

5 Conclusion

We have proposed algorithms to efficiently compute the parametric state classes of parametric time Petri nets using tropical parametric DBMs (tPDBMs). While the idea of PDBMs has first been proposed in [15] for parametric timed automata, we adapt here to a different formalism and a different state-space abstraction. We also generalize the notion of PDBM to include tropical expressions in the

[2] Inclusion tests are typically done many times for a given class, possibly for all the previous classes with the same marking. Successor computation, which is somewhat costly in PDBMs due to potential splits, is only done once per class.

matrix, which allows us to avoid splitting. Finally, we have implemented the technique and proposed some experimental evidence that this generalization is often beneficial. Further work includes (i) trying to take advantage of the tPDBM specific form to compute integer hulls of tPDBMs more efficiently to use the algorithms of [16], (ii) explore a posteriori merging of split PDBMs, (iii) further extend the operations on tPDBMs to work for more complex algorithms like the controller synthesis of [18].

A Proposition 2

Proof. The proof of (1) is detailed in [11].

Proof of (2): Let $K_\sigma = (m, (P_c, M_c))$ be the state class containing all states reachable by firing transition sequence σ in $\mathcal{N}_p$. Recall from Proposition 1 that for all K_σ, we get that $\forall v_p \in [\![P_c]\!].v_p(K_\sigma) = K'_\sigma$ the corresponding state class from $\mathcal{N} = v_p(\mathcal{N}_p)$. Using the same argument, for all state $s_p = (m, v_p, \theta)$ from $\mathcal{N}_p$, we have that $v_p(s_p) \in v_p(K_\sigma) \Leftrightarrow s_p \in K_\sigma$.

By the induction hypothesis and using Lemma 2, all parameter valuations in $[\![P_c]\!]$ ensure that all transitions from σ where firable. Then the new conditions $\bigcap_{i \neq f, t_i \in \mathsf{en}(C)} \{M_c[f,i] \geq 0\}$ ensure that transition t_f is firable. Finally, $[\![\mathsf{fir}_\mathsf{p}(K_\sigma, t_f)]\!]$ corresponds to all valuations ensuring that the sequence of transition $\sigma \cdot t_f$ is firable. Now we prove the part corresponding to the tPDBM:

- $\supseteq$: We take $s_p = (m', v_p, \theta) \in \big(m', [\![(\mathsf{fir}_\mathsf{p}(K_\sigma, t_f), \mathsf{Succ}_\mathsf{p}((m, M_c), t_f))]\!]\big)$. Then clearly $v_p \in [\![\mathsf{fir}_\mathsf{p}(K_\sigma, t_f)]\!] = [\![P_c]\!] \cap [\![\bigcap_{i \neq f, t_i \in \mathsf{en}(C)} \{M_c[f,i] \geq 0\}]\!]$.
  We also have that $\theta \in [\![\mathsf{Succ}_\mathsf{p}((m, M_c), t_f)]\!]$. Using Proposition 3, we get that $v_p(\theta) \in [\![\mathsf{Succ}((m, v_p(K_\sigma)), t_f)]\!]$. And then using (1), $v_p(s_p) \in \mathsf{Next}(v_p(K_\sigma), t_f) = v_p(K_{\sigma.t_f})$. Since by definition $K_{\sigma.t_f} = \mathsf{Next}_\mathsf{p}(K_\sigma, t_f)$, we conclude that $s_p \in \mathsf{Next}_\mathsf{p}(K_\sigma, t_f)$.
- $\subseteq$: Let $s_p = (m', v_p, \theta) \in \mathsf{Next}_\mathsf{p}(K_\sigma, t_f) = K_{\sigma \cdot t_f}$. We have that $v_p(s_p) \in v_p(K_{\sigma \cdot t_f}) = \mathsf{Next}(v_p(K_\sigma), t_f)$, which using (1) is $v_p(s_p) \in (m', [\![\mathsf{Succ}((m, v_p(K_\sigma)), t_f)]\!])$. Lemma 3 gives us that $v_p(s_p) \in (m', [\![v_p(\mathsf{Succ}_\mathsf{p}((m, M_c), t_f))]\!])$. Finally $s_p \in K_{\sigma \cdot t_f}$, so its parameter valuation ensures that $\sigma \cdot t_f$ is firable. Hence $v_p \in [\![\mathsf{fir}_\mathsf{p}(K_\sigma, t_f,]\!])$, which allows us to conclude. □

B Proposition 3

Proof. We will denote by $\widetilde{\mathsf{min}}_{k \in T'}$ the $\widetilde{\mathsf{min}}$ for all t_k's from a set of transitions $T' \subseteq T$. Then $\forall t_i \in \mathsf{pers}(m, t_f)$ using Algorithm 1 line 8 we have:

$$v_p(\mathsf{Succ}_\mathsf{p}((m, M_c), t_f))[i, o] = v_p\big(\widetilde{\mathsf{min}}_{k \in \mathsf{en}(m)}(0, M_c[k, i])\big) = \min_{k \in \mathsf{en}(m)} \big(0, v_p(M_c[k, i])\big)$$
$$= \mathsf{Succ}((m, v_p(M_c)), t_f)[i, o]$$
$$v_p(\mathsf{Succ}_\mathsf{p}((m, M_c), t_f))[o, i] = v_p\big(M_c[i, f]\big) = \mathsf{Succ}((m, v_p(M_c)), t_f)[o, i]$$

Without loss of generality, suppose that $t_i, t_j \in \mathsf{pers}(m, t_f)$, the cases for $t_i \in \mathsf{newen}(m, t_f)$ and $t_j \in \mathsf{newen}(m, t_f)$ are similar.

$$\begin{aligned}
v_p(\mathsf{Succ_p}((m, M_c), t_f))[i,j] &= v_p\Big(\widetilde{\min}\big(M_c[i,j], M'_c[i,o] \mathbin{\widetilde{+}} M'_c[o,j]\big)\Big) \\
&= \min\big(v_p(M_c[i,j]), v_p(M'_c[i,o]) + v_p(M'_c[o,j])\big) \\
&= \min\big(v_p(M_c[i,j]), v_p(\mathsf{Succ_p}((m, M_c), t_f)[i,o]) \\
&\qquad + v_p(\mathsf{Succ_p}((m, M_c), t_f)[o,j])\big) \\
&= \min\big(v_p(M_c[i,j]), \mathsf{Succ}((m, v_p(M_c)), t_f)[i,o]) \\
&\qquad + v_p(\mathsf{Succ}((m, v_p(M_c)), t_f)[o,j])\big) \\
&= \mathsf{Succ}((m, v_p(M_c)), t_f)[i,j]
\end{aligned}$$

□

C Proposition 5

Proof. – Assume $[\![D^1]\!] \subseteq [\![D^2]\!]$. This means $[\![D^1]\!] \subseteq [\![D^2]\!]$. Then their projection on parameters satisfy the same inclusion and by Corollary 1, we have $[\![P_c^1]\!] \subseteq [\![P_c^2]\!]$.
Furthermore, for all $v_p \in [\![P_c^1]\!]$, v_p is thus in the projection on parameters of both $[\![D^1]\!]$ and $[\![D^2]\!]$ and we have $[\![v_p(\mathcal{P}(D^1))]\!] \subseteq [\![v_p(\mathcal{P}(D^2))]\!]$, which also implies that $[\![v_p(\mathcal{P}(M_c^1))]\!] \subseteq [\![v_p(\mathcal{P}(M_c^2))]\!]$. This finally implies that $\forall i,j, v_p(M_c^1)[i,j] \leq v_p(M_c^2)[i,j]$ because $v_p(M_c^1)$ and $v_p(M_c^2)$ are DBMs in canonical form. Finally, since $v_p(M_c^1)[i,j]$ is clearly the same thing as $v_p(M_c^1[i,j])$, and similarly for M_c^2, we get the expected result.
– Assume now $[\![P_c^1]\!] \subseteq [\![P_c^2]\!]$ and $\forall v_p \in [\![P_c^1]\!], \forall i,j, v_p(M_c^1[i,j]) \leq v_p(M_c^2[i,j])$. Due to the form of constraints in M_c^1 and M_c^2, this implies that $[\![(P_c^1, M_c^1)]\!] \subseteq [\![(P_c^1, M_c^2)]\!]$. And since $[\![P_c^1]\!] \subseteq [\![P_c^2]\!]$, we also directly have $[\![(P_c^1, M_c^2)]\!] \subseteq [\![(P_c^2, M_c^2)]\!]$, and the expected result follows.

□

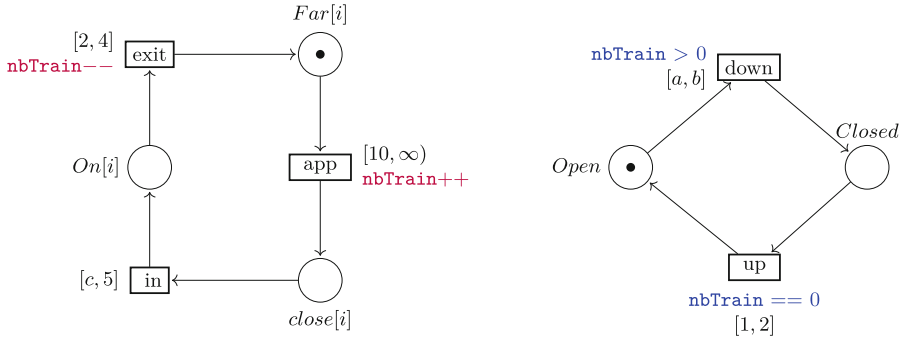
D Models for the Experiments

We consider Petri Nets where preconditions (guards) and post-conditions (updates) over a set of variables are associated with transitions. A transition is enabled if there are enough tokens in its input places and if the guard is true. When a transition fires the corresponding update is executed modifying the values of the variables. The variables take their values in finite sets (countable and bounded) such as bounded integers or enumeration type.

As we are considering bounded PTPN with a single bounded integer variable, all operations and tests over the variable, including zero testing, can be carried out using a Petri net, so we remain in the PTPN class.

In the figures, guards are drawn in blue and updates in purple.

The model for Fischer's protocol is similar to the one in [24] and the model for level crossing is given in Fig. 1.



1.a) Model of train i

1.b) Model of the barrier

Fig. 1. Level crossing

References

1. Abadi, M., Lamport, L.: An old-fashioned recipe for real time. In: de Bakker, J.W., Huizing, C., de Roever, W.P., Rozenberg, G. (eds.) REX 1991. LNCS, vol. 600, pp. 1–27. Springer, Heidelberg (1992). https://doi.org/10.1007/BFb0031985
2. Alur, R., Henzinger, T., Vardi, M.: Parametric real-time reasoning. In: Conference Proceedings of the Annual ACM Symposium on Theory of Computing (1997). https://doi.org/10.1145/167088.167242
3. Alur, R., Henzinger, T.A., Vardi, M.Y.: Parametric real-time reasoning. In: ACM Symposium on Theory of Computing, pp. 592–601 (1993)
4. André, É., Chatain, T., Encrenaz, E., Fribourg, L.: An inverse method for parametric timed automata. Int. J. Found. Comput. Sci. **20**(5), 819–836 (2009)
5. André, É., Lime, D., Roux, O.H.: Integer-complete synthesis for bounded parametric timed automata. In: Bojańczyk, M., Lasota, S., Potapov, I. (eds.) RP 2015. LNCS, vol. 9328, pp. 7–19. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24537-9_2
6. Bengtsson, J.: Clocks, DBMs and states in timed systems. Ph.D. thesis, Department of Information Technology, Uppsala University, Sweden (2002)
7. Bengtsson, J., Yi, W.: On clock difference constraints and termination in reachability analysis of timed automata. In: Dong, J.S., Woodcock, J. (eds.) ICFEM 2003. LNCS, vol. 2885, pp. 491–503. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-39893-6_28
8. Berthomieu, B., Diaz, M.: Modeling and verification of time dependent systems using time Petri nets. IEEE Trans. Softw. Eng. **17**(3), 259–273 (1991)
9. Berthomieu, B., Vernadat, F.: State class constructions for branching analysis of time Petri nets. In: Garavel, H., Hatcliff, J. (eds.) TACAS 2003. LNCS, vol. 2619, pp. 442–457. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36577-X_33
10. Berthomieu, B., Menasche, M.: An enumerative approach for analyzing time Petri nets. In: Proceedings IFIP, pp. 41–46. Elsevier Science Publishers (1983)
11. Boucheneb, H., Mullins, J.: Analyse des réseaux temporels: Calcul des classes en $O(n^2)$ et des temps de chemin en $O(m \times n)$. TSI. Technique et science informatiques **22**(4), 435–459 (2003)

12. Bourdil, P.A., Berthomieu, B., Dal Zilio, S., Vernadat, F.: Symmetry reduction for time Petri net state classes. Sci. Comput. Program. **132**, 209–225 (2016)
13. Dijkstra, E.: Co-operating sequential processes. In: Programming Languages: NATO Advanced Study Institute: Lectures Given at a Three Weeks Summer School Held in Villard-le-Lans, 1966/ed. by F. Genuys, pp. 43–112. Academic Press Inc. (1968)
14. Dill, D.L.: Timing assumptions and verification of finite-state concurrent systems. In: Sifakis, J. (ed.) CAV 1989. LNCS, vol. 407, pp. 197–212. Springer, Heidelberg (1990). https://doi.org/10.1007/3-540-52148-8_17
15. Hune, T., Romijn, J., Stoelinga, M., Vaandrager, F.W.: Linear parametric model checking of timed automata. J. Log. Algebraic Program. **52–53**, 183–220 (2002)
16. Jovanović, A., Lime, D., Roux, O.H.: Integer parameter synthesis for real-time systems. IEEE Trans. Softw. Eng. (TSE) **41**(5), 445–461 (2015)
17. Lamport, L.: A fast mutual exclusion algorithm. ACM Trans. Comput. Syst. **5**(1), 1–11 (1987)
18. Leclercq, L., Lime, D., Roux, O.H.: A state class based controller synthesis approach for time Petri nets. In: Gomes, L., Lorenz, R. (eds.) PETRI NETS 2023. LNCS, vol. 13929, pp. 393–414. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-33620-1_21
19. Lime, D., Roux, O.H., Seidner, C.: Cost problems for parametric time Petri nets. Fund. Inform. **183**(1–2), 97–123 (2021). https://doi.org/10.3233/FI-2021-2083
20. Lime, D., Roux, O.H., Seidner, C., Traonouez, L.-M.: Romeo: a parametric model-checker for Petri nets with stopwatches. In: Kowalewski, S., Philippou, A. (eds.) TACAS 2009. LNCS, vol. 5505, pp. 54–57. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-00768-2_6
21. Parquier, B., et al.: Applying parametric model-checking techniques for reusing real-time critical systems. In: Artho, C., Ölveczky, P.C. (eds.) FTSCS 2016. CCIS, vol. 694, pp. 129–144. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-53946-1_8
22. Pin, J.E.: Tropical semirings. In: Gunawardena, J. (ed.) Idempotency (Bristol, 1994), pp. 50–69. Publ. Newton Inst. 11. Cambridge University Press, Cambridge (1998). https://hal.science/hal-00113779
23. Traonouez, L.M., Lime, D., Roux, O.H.: Parametric model-checking of stopwatch Petri nets. J. Univ. Comput. Sci. **15**(17), 3273–3304 (2009). A publication of Graz University of Technology and Universiti Malaysia Sarawak
24. Tripakis, S., Yovine, S.: Analysis of timed systems using time-abstracting bisimulations. Formal Methods Syst. Des. **18**(1), 25–68 (2001)