Preserving Partial Order Runs in Parametric Time Petri Nets

Étienne André, Thomas Chatain, César Rodriguez

Preserving Partial Order Runs in Parametric Time Petri Nets

ÉTIENNE ANDRÉ, Université Paris 13, Sorbonne Paris Cité, LIPN, CNRS, Villetaneuse, France
THOMAS CHATAIN, LSV, ENS Cachan, INRIA, CNRS, France
CÉSAR RODRÍGUEZ, Université Paris 13, Sorbonne Paris Cité, LIPN, CNRS, Villetaneuse, France

Parameter synthesis for timed systems aims at deriving parameter valuations satisfying a given property. In this paper we target concurrent systems. We use partial-order semantics for parametric time Petri nets as a way to both *1*) cope with the well-known state-space explosion due to concurrency, and *2*) significantly enhance the result of an existing synthesis algorithm. Given a reference parameter valuation, our approach synthesizes other valuations preserving the partial-order executions of the reference parameter valuation. We show the applicability of our approach using a tool applied to asynchronous circuits.

CCS Concepts: •**Security and privacy** → **Logic and verification;** •**Theory of computation** → **Timed and hybrid models;** Parallel computing models;

General Terms: Timed and hybrid systems, concurrency, time Petri nets, unfolding semantics, inverse method, robustness

ACM Reference Format:
Étienne André, Thomas Chatain, and César Rodríguez, 2016. Preserving Partial Order Runs in Parametric Time Petri Nets. *ACM Trans. Embedd. Comput. Syst.* 16, 2, Article 43 (December 2016), 25 pages.
DOI: 10.1145/3012283

1. INTRODUCTION

Parametric verification of timed systems allows designers to model a system incompletely specified, or subject to future changes, by allowing the use of *parameters*, i.e. unknown constants. The parameter synthesis problem aims at deriving a set of parameter valuations which preserve some property (e.g. a safety property, or a more complex property expressed using some temporal logics). Popular formalisms to model and verify parametric concurrent timed systems include parametric timed automata (PTAs) [Alur et al. 1993] or parametric time Petri nets (PTPNs) [Traonouez et al. 2009].

Parameter synthesis for PTAs or PTPNs was tackled with respect to safety or unavoidability of some states (e.g. [Alur et al. 1993; André and Soulat 2011; Jovanović et al. 2015]), or the satisfiability of temporal logic formulas (e.g. [Bruyère and Raskin 2007; Knapik and Penczek 2012]). The underlying decision problems behind these synthesis problems are all undecidable in general, with only two non-trivial exceptions: for a subclass of PTAs called L/U-PTAs [Hune et al. 2002; Bozzelli and La Torre 2009], the emptiness and the universality of the set of parameter valuations for which a state is reachable, or for which there exists an infinite accepting run, is decidable. The same holds for L/U-PTPNs [Traonouez et al. 2009]. Applications of parametric verification techniques for timed systems include the verification of asynchronous circuits with parametric propagation delays using octahedra [Clarisó and Cortadella 2005] and PTAs [Chevallier et al. 2009].

In [André et al. 2009; André et al. 2013], we proposed the inverse method IM: given a PTPN $\mathcal{N}$ and a reference parameter valuation v_0, $\mathsf{IM}(\mathcal{N}, v_0)$ synthesizes other param-

This work is partially supported by the ANR national research program ANR-14-CE28-0002 PACS ("Parametric Analyses of Concurrent Systems").
 1539-9087/2016/12-ART43 $15.00
DOI: 10.1145/3012283

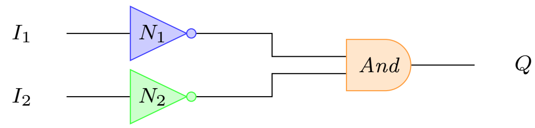

Fig. 1: An asynchronous circuit

eter valuations around v_0 in the form of a linear parameter constraint K such that, for any valuation satisfying K, the time-abstract behavior of the system is identical to the one of v_0. The reference parameter valuation is a valuation of each parameter known a priori, e.g. from simulation (in the case of asynchronous circuits) or from the specification (in the case of protocols). Among various applications, this constraint helps to quantify the system robustness w.r.t. infinitesimal variations of the timing constants. The inverse method was also used to improve the latency in circuit design (e.g. [Chevallier et al. 2009; André and Soulat 2011]). Then, in [André and Soulat 2011], we proposed several extensions of IM, including one, IM^K (called *inverse method with direct return* in [André and Soulat 2011]), that we will specifically consider here: $\mathsf{IM}^K(\mathcal{N}, v_0)$ synthesizes a linear parameter constraint K such that, for any valuation satisfying K, the time-abstract behavior of the system is *included in* the one of v_0. That is, if the system is safe for v_0, then it is also safe for any valuation satisfying the constraint synthesized by IM^K.

In this paper we focus on systems featuring both concurrent behaviors and real-time constraints. Applying formal methods to these systems is a notoriously difficult problem. A high degree of concurrency between the various components of the system often leads to the *state-space explosion* problem, thus hindering exhaustive analyses. Techniques for coping with state-space explosion include *partial-order* (or *unfolding*) semantics and *partial-order reductions* (PORs). Both approaches fundamentally exploit the independence (commutativity) of concurrent actions to yield reduction. However, while the literature contains established partial-order techniques and tools for untimed, asynchronous systems, this is less the case for real-time, distributed ones. A key reason for this is the difficulty to define independence relations for timed systems: seemingly independent (concurrent) actions can be ordered by their occurrence time. This helps explain why little literature exists on POR techniques for time Petri nets [Penczek and Pólrola 2001; Virbitskaite and Pokozy 1999; Yoneda and Schlingloff 1997; Mercer et al. 2002] or networks of timed automata [Bengtsson et al. 1998; Minea 1999; Lugiez et al. 2005; Niebert and Qu 2006]. The situation is similar for partial-order semantics of time Petri nets [Aura and Lilius 2000; Chatain and Jard 2006; Traonouez et al. 2010] or networks of timed automata [Cassez et al. 2006; Bouyer et al. 2006]. Modular verification of time(d) Petri nets was also studied, e.g. in [Peres et al. 2011] and [Zheng et al. 2001], with specific applications to circuits.

In this paper we use partial-order semantics to achieve a double benefit. Not only they cope with the state-space explosion problem but they also enhance the quality of the output of IM^K, i.e. the algorithm outputs a *larger* set of parameter valuations.

Example 1.1. *In this motivating example we illustrate the interest of our technique as well as the fact that the generated sets of parameter valuations are larger. Consider the asynchronous circuit shown in Fig. 1. We consider a classical inertial model, where all logic gates feature a propagation time (also called traversal delay, or latency): whenever an input of the gate is changed, then the output changes only after that propagation time – unless an input changes again. The propagation times of every logic gate are the parameters of the system. Observe that the gates* N_1 *and* N_2 *are structurally concurrent. The circuit is studied in the following precise scenario: initially* $I_1 = 1$ *and* $I_2 = 0$

(and therefore $Q = 0$); then, signal I_1 falls and signal I_2 rises, which causes N_1 to rise (denoted by $N_1^{\nearrow}$) and N_2 to fall ($N_2^{\searrow}$). Depending on the timing delays of the circuit, Q may or may not rise. Basically, if N_1 rises before N_2 falls, and if the propagation time of the And gate is smaller than that of N_2, then Q may rise. Assume that the rise of Q represents, due to external reasons, a safety violation (bad behavior). Additionally, assume that we have a reference parameter valuation v_0 for which Q never rises and which forces $N_1^{\nearrow}$ before $N_2^{\searrow}$. In other words, N_1 systematically reacts much faster than N_2. IM^K will output a constraint on the parameters which preserves the sequential *behavior of the circuit: Q never rises and $N_1^{\nearrow}$ before $N_2^{\searrow}$. Now, this constraint can be viewed as being too tight. Any other constraint preventing Q from raising and allowing the concurrent gates N_1 and N_2 to react in* any *order, would be equally useful for us. Since gates N_1 and N_2 operate concurrently, the ordering of their propagation delays is in principle irrelevant as long as the safety violation (Q rises) does not occur.*

We construct such constraint by preserving the partial-order executions of the circuit, rather than the sequential ones, as IM^K would do.

The parameter constraint that disallows Q to rise and lets N_1 and N_2 propagate signals in any *order is thus larger than the one that IM^K would generate. In other words, IM^K preserves here the* temporal ordering *fixed by v_0, while our method preserves the* partial-order, untimed, behavior *fixed by v_0 (which also prevents Q from rising).*

Contribution. In this paper, we propose an approach called $\mathsf{IM}^K\mathsf{PO}$ (standing for "inverse method with direct return based on partial orders") that, given a PTPN and a reference parameter valuation, synthesizes further parameter valuations for which the partial-order runs are the same as for the reference valuation. Different from the inverse method with direct return, we define here an *ad-hoc* partial-order semantics, that we use to synthesize parameters generalizing the behaviors of v. We show that $\mathsf{IM}^K\mathsf{PO}$ significantly enhances the result of IM^K, by relaxing the resulting constraint. This is of high interest when dealing with the parametric verification of asynchronous circuits, since a relaxed constraint will improve the allowed latencies in circuit design without leading to global timing violations. Our approach is at first dedicated to acyclic systems (in particular, our main result, Theorem 4.5, deals with acyclic systems): we do not consider it as a significant drawback when dealing with circuit design, since many circuits are acyclic, and circuits in a cyclic environment are often verified using scenarios involving a limited number of clock cycles (see, e.g. [Chevallier et al. 2009]). Still, we provide two extensions of $\mathsf{IM}^K\mathsf{PO}$ to deal with (possibly partially) cyclic systems.

Outline. In Section 2, we define time Petri nets and their parametric extension; we also recall the inverse method (Section 2.3). In Section 3, we introduce our partial-order semantics for TPNs. Section 4 is our main contribution: we define the problem of parameter synthesis for preserving partial-order runs; next we present our method $\mathsf{IM}^K\mathsf{PO}$ which solves the problem for acyclic nets; we also present a first variant $\mathsf{IM}^K\mathsf{PO}'$ of the method that addresses limited cyclic systems, and a second variant $\mathsf{IM}^K\mathsf{PO}^{\text{blocks}}$ that aims at achieving better termination than $\mathsf{IM}^K\mathsf{PO}$ for fully cyclic systems. We illustrate our method $\mathsf{IM}^K\mathsf{PO}$ in Section 5 by applying it to a scenario of the asynchronous circuit of Fig. 1, and we apply $\mathsf{IM}^K\mathsf{PO}'$ to a circuit with a loop. We briefly report on our implementation in Section 6. We conclude in Section 7.

2. PARAMETRIC TIME PETRI NETS

In this section, we first define (non-parametric) time Petri nets and their semantics (Section 2.1); then we introduce notations for parametric models (Section 2.2); finally, we recall the inverse method (Section 2.3).

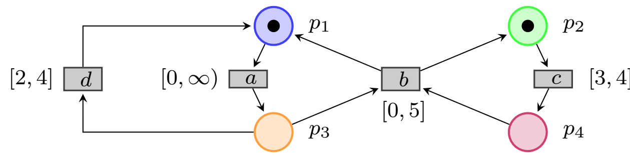

Fig. 2: A safe time Petri net

2.1. Time Petri Nets

We consider only *safe* time Petri nets (TPNs), i.e. TPNs where there is never more than one token in a place.

Definition 2.1 (Time Petri Net (TPN) [Merlin and Farber 1976]). *A* time Petri net *is a tuple* $(P, T, pre, post, efd, lfd, M_0)$ *where* P *and* T *are finite sets of* places *and* transitions *respectively;* pre *and* $post$ *map each transition* $t \in T$ *to its (nonempty)* preset *denoted* ${}^\bullet t \stackrel{\text{def}}{=} pre(t) \subseteq P$ *and its (possibly empty)* postset *denoted* $t^\bullet \stackrel{\text{def}}{=} post(t) \subseteq P$*;* $efd : T \to \mathbb{Q}_+$ *and* $lfd : T \to \mathbb{Q}_+ \cup \{\infty\}$ *associate the* earliest firing delay $efd(t)$ *and* latest firing delay $lfd(t) \geq efd(t)$ *with each transition* t*;* $M_0 \subseteq P$ *is the* initial marking.

As usual, we graphically represent places as circles and transitions as rectangles. We write the time interval $[efd(t), lfd(t)]$ next to the transition. See Fig. 2.

State. A *state* of a safe time Petri net is a triple (M, dob, θ), where $M \subseteq P$ is the *marking*, $\theta \in \mathbb{R}$ is the current time and $dob : M \to \mathbb{R}$ associates a *date of birth* $dob(p) \in \mathbb{R}$ with each token (marked place) $p \in M$. The *initial state* is $(M_0, dob_0, 0)$ and initially, all the tokens carry the date 0 as date of birth: for all $p \in M_0$, $dob_0(p) \stackrel{\text{def}}{=} 0$.

A transition $t \in T$ is *enabled* in a marking M if ${}^\bullet t \subseteq M$. The set of transitions enabled in M is denoted by $En(M)$. Given a state (M, dob, θ) and a transition t enabled in M, we define the *date of enabling* of t as the date of birth of the youngest token in its input places: $doe(t) \stackrel{\text{def}}{=} \max_{p \in {}^\bullet t} dob(p)$.

Again, we consider only *safe* time Petri nets, that is we assume that if a transition $t \in T$ is enabled in a marking M, then $(M \setminus {}^\bullet t) \cap t^\bullet = \emptyset$. Moreover, because in this work we aim at synthesizing new values for the timing constants, we require that even the untimed support is safe, i.e. the TPN remains safe if one replaces all the earliest firing delays by 0 and all the latest firing delays by ∞.

Time delay. The TPN can wait until time $\theta' \geq \theta$ provided no enabled transition overtakes its maximum delay, i.e. $\forall t \in En(M),\ \theta' \leq doe(t) + lfd(t)$. The reached state is (M, dob, θ').

Discrete action. Transition t can fire from state (M, dob, θ) if t is enabled ($t \in En(M)$) and t has reached its minimum firing delay ($\theta \geq doe(t) + efd(t)$). Firing transition t from state (M, dob, θ) leads to state (M', dob', θ), with $M' \stackrel{\text{def}}{=} (M \setminus {}^\bullet t) \cup t^\bullet$ and $dob'(p) \stackrel{\text{def}}{=} dob(p)$ if $p \in M \setminus {}^\bullet t$ and $dob'(p) \stackrel{\text{def}}{=} \theta'$ if $p \in t^\bullet$ (by assumption the two cases are exclusive).

Timed words. When representing an execution, we often forget the information about the intermediate states and delays, and remember only the (possibly infinite) sequence $((t_1, \theta_1), \ldots, (t_n, \theta_n) \ldots)$ of transitions with their firing dates. This representation is called a *timed word*. The empty timed word is denoted by ϵ. Given a timed word $((t_1, \theta_1), \ldots, (t_n, \theta_n) \ldots)$, its associated *sequence* is the time-abstract word $(t_1, \ldots, t_n \ldots)$. Given a TPN N, we denote by $Sequences(N)$ the set of sequences associated with all timed words of N, among which we distinguish the set $MaxSequences(N)$ of *maximal* sequences, i.e. sequences which are not the prefix of any other sequence.

Remark 2.2. *Notice that the maximality among sequences matches well the maximality among timed words (accepted by N) in the sense that, if a finite timed word $((t_1, \theta_1), \ldots, (t_n, \theta_n))$ is maximal among the timed words accepted by N, then any other timed word $((t_1, \theta'_1), \ldots, (t_n, \theta'_n))$ corresponding to the same sequence $(t_1, \ldots, t_n)$, is also maximal. The reason is that*

— *a finite timed word is maximal iff it reaches a state (M, dob, θ) such that M does not enable any transition; and that*
— *the states reached after $((t_1, \theta_1), \ldots, (t_n, \theta_n))$ and after $((t_1, \theta'_1), \ldots, (t_n, \theta'_n))$ have the same marking.*

2.2. Parametric Time Petri Nets

2.2.1. Parameters and Constraints. Throughout this paper, Θ will denote a finite set $\{\theta_1, \ldots, \theta_H\}$ of *firing times*, for some $H \in \mathbb{N}$. A *firing time valuation* is a function $w\colon \Theta \to \mathbb{R}_+^H$ assigning a non-negative real value with each firing time.

Given a finite set $\Lambda = \{\lambda_1, \ldots, \lambda_j\}$ of *parameters* (i.e. unknown constants), for some $j \in \mathbb{N}$, a *parameter valuation* v is a function $v : \Lambda \to \mathbb{Q}_+$ assigning with each parameter a value in $\mathbb{Q}_+$. For technical convenience, we extend the function v to $\mathbb{Q}_+ \cup \{\infty\}$.

Given a set X of variables, a linear inequality over X is of the form $lt \prec lt'$, where $\prec \in \{<, \leq\}$, and lt, lt' are two linear terms of the form $\sum_{1 \leq i \leq |X|} \alpha_i x_i + d$ where $|X|$ denotes the cardinality of X, $x_i \in X$, $\alpha_i \in \mathbb{Q}_+$, for $1 \leq i \leq |X|$, and $d \in \mathbb{Q}_+$.

A constraint over X is a Boolean combination (disjunctions and conjunctions) of linear inequalities. In the following, we will use constraints over Θ, over $\Theta \cup \Lambda$, and over Λ. A constraint over Λ is called a *parameter constraint*, and can be seen as a polyhedron in j dimensions. A parameter valuation v *satisfies* a parameter constraint K, denoted by $v \models K$, if the expression obtained by replacing each parameter λ in K with $v(\lambda)$ evaluates to true. We consider `true` as a constraint over the parameters Λ, corresponding to the set of all possible values for Λ.

2.2.2. Parametric Time Petri Nets. Parametric time Petri nets (PTPNs) are a parametric extension of TPNs, where the temporal bound of each transition can either be a rational number, ∞ or a parameter [Traonouez et al. 2009; André et al. 2013].[1]

Definition 2.3 (PTPN). *A parametric time Petri net (PTPN) is a tuple $\mathcal{N} \stackrel{\text{def}}{=} (P, T, \Lambda, pre, post, pefd, plfd, M_0, K_0)$ where*

— *P and T are non-empty, disjoint sets of* places *and* transitions *respectively,*
— *$\Lambda \stackrel{\text{def}}{=} \{\lambda_1, \ldots, \lambda_j\}$ is a finite set of* parameters*,*
— *pre and $post$ map each transition $t \in T$ to its (nonempty)* preset*, denoted by ${}^\bullet t \stackrel{\text{def}}{=} pre(t) \subseteq P$, and its (possibly empty)* postset*, denoted by $t^\bullet \stackrel{\text{def}}{=} post(t) \subseteq P$;*
— *functions $pefd\colon T \to \mathbb{Q}_+ \cup \Lambda$ and $plfd\colon T \to \mathbb{Q}_+ \cup \Lambda \cup \{\infty\}$ and associate the* earliest firing delay *$pefd(t)$ and* latest firing delay *$plfd(t)$ with each transition t,*
— *$M_0 \subseteq P$ is the* initial marking*, and*
— *K_0 is the* initial constraint *over Λ giving the initial domain of the parameters, and must at least specify that the firing intervals are nonempty ($\bigwedge_{t \in T} pefd(t) \leq plfd(t)$). (This ensures that, for every valuation v of the parameters satisfying K_0, the instanciated TPN is an actual TPN according to Definition 2.1.)*

K_0 is called initial because parameters are initially bound by this constraint; their value can then be further restricted by the analysis. Restricting the initial valuations

[1]In fact, we could be more permissive by allowing, for each bound, a (convex) linear term over $\Lambda \cup \mathbb{Q}_+$. We stick to $\mathbb{Q}_+ \cup \Lambda \cup \{\infty\}$ for sake of simplicity, but all our results naturally extend to the case of linear terms.

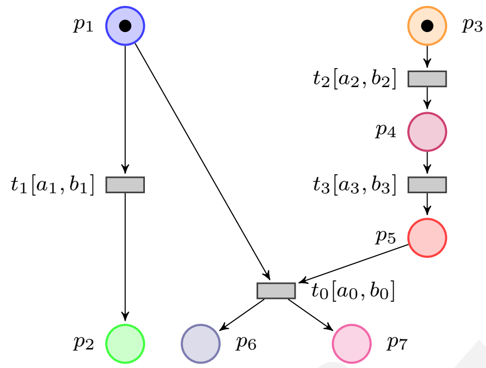

Fig. 3: An example of a PTPN

of the parameters is something classical (see e.g. [Bozzelli and La Torre 2009] for theoretical results) and is used in the inverse method [André et al. 2009]. Additional linear inequalities may of course be given. Fig. 3 shows a PTPN, where the bounds of all firing intervals happen to be parametric. The initial constraint K_0 would be of the form $(a_0 \leq b_0) \wedge (a_1 \leq b_1) \wedge (a_2 \leq b_2) \wedge (a_3 \leq b_3) \wedge K$, for some constraint K.

Definition 2.4 ($[\![\mathcal{N}]\!]_v$). *Given a PTPN $\mathcal{N} \stackrel{\text{def}}{=} (P, T, \Lambda, pre, post, pefd, plfd, M_0, K_0)$ and a valuation $v : \Lambda \to \mathbb{Q}_+$, we denote by $[\![\mathcal{N}]\!]_v$ the (non-parametric) TPN where each occurrence of a parameter has been replaced by its constant value as in v. Formally, $[\![\mathcal{N}]\!]_v$ is the TPN $(P, T, pre, post, efd, lfd, M_0)$ with $efd(t) \stackrel{\text{def}}{=} v(pefd(t))$ and $lfd(t) \stackrel{\text{def}}{=} v(plfd(t))$ for every $t \in T$. We call $[\![\mathcal{N}]\!]_v$ an* instantiation *of $\mathcal{N}$ with v.*

Lemma 2.5. *Let $\mathcal{N} \stackrel{\text{def}}{=} (P, T, \Lambda, pre, post, pefd, plfd, M_0, K_0)$ be a PTPN and v, v' be two valuations of the parameters, both satisfying the initial constraint K_0. Then every sequence $t_1, \dots, t_n \dots$ which is both in $Sequences([\![\mathcal{N}]\!]_v)$ and in $Sequences([\![\mathcal{N}]\!]_{v'})$, is maximal in $Sequences([\![\mathcal{N}]\!]_v)$ iff it is maximal in $Sequences([\![\mathcal{N}]\!]_{v'})$.*

Proof. Infinite sequences are necessarily maximal. Now, as observed in Remark 2.2, a finite sequence is maximal iff it reaches a marking which enables no transition. This depends only on the sequence, not on the valuation. □

2.3. Preserving Time-Abstract Runs Using IM^K

In [André and Soulat 2011], we presented the *inverse method with direct return* IM^K. It considers a system modeled using a network of PTAs and synthesizes a constraint by taking advantage of a reference parameter valuation. The inverse method was then extended to PTPNs [André et al. 2013]. Given a PTPN $\mathcal{N}$ and a reference parameter valuation v_0, $\mathsf{IM}^K(\mathcal{N}, v_0)$ generalizes v_0 by computing a constraint K over Λ such that, for any v satisfying K, the set of maximal sequences of $[\![\mathcal{N}]\!]_v$ is included in the one of $[\![\mathcal{N}]\!]_{v_0}$. We say that $\mathsf{IM}^K(\mathcal{N}, v_0)$ generalizes v_0 because we have in particular $v_0 \models K$.

IM^K explores a set of symbolic states of the input PTPN. This parametric semantics (not given here for sake of conciseness, but available in [Traonouez et al. 2009; André et al. 2013]) considers symbolic states made of a marking and a constraint over $\Theta \cup \Lambda$, i.e. variables similar to clocks in (P)TAs [Alur et al. 1993; Alur and Dill 1994], with the exception that they decrease with time whereas PTA clocks increase. IM^K maintains a parametric constraint K (initially set to `true`), and performs a breadth-first exploration of this symbolic state space. Then, whenever a v_0-incompatible state is met (i.e.

the constraint associated to which is not satisfied by v_0), IM^K computes the projection of this constraint onto Λ (i.e. eliminates the parametric firing times in Θ using variable elimination techniques such as Fourier-Motzkin [Schrijver 1986]), selects one v_0-incompatible inequality, and adds its negation to K. When a fixpoint is reached (i.e. no new states can be explored), the algorithm returns K. Additional details on IM^K can be found in [André and Soulat 2011; André et al. 2013].

The result of IM^K has several applications. First, it allows designers to replace some system components while keeping the system correctness: changing a parameter valuation with another one that satisfies K will preserve (some of) the admissible behaviors of $[\![\mathcal{N}]\!]_{v_0}$, and will prevent any behavior not allowed in $[\![\mathcal{N}]\!]_{v_0}$. Second, the inverse method (together with its variants) gives a measure of the system robustness (see, e.g. [Markey 2011]), i.e. it quantifies the admissible variability of the timing delays in the model that will still preserve the system correctness: the constraint K gives a precise measure of the variations of the parameters with respect to one another [André et al. 2013].

Theorem 2.6 ([André and Soulat 2011])**.** *Let $\mathcal{N}$ be a PTPN and v_0 be a parameter valuation. Assume $\mathsf{IM}^K(\mathcal{N}, v_0)$ terminates with result K. Then for all valuation v of the parameters satisfying the initial constraint K_0 of the model,*

$$v \models K \iff \mathit{Sequences}([\![\mathcal{N}]\!]_v) \subseteq \mathit{Sequences}([\![\mathcal{N}]\!]_{v_0}) \,.$$

In particular $v_0 \models K$.

Moreover, by Lemma 2.5, a sequence is maximal in $\mathit{Sequences}([\![\mathcal{N}]\!]_v)$ iff it is maximal in $\mathit{Sequences}([\![\mathcal{N}]\!]_{v_0})$. Hence,

$$v \models K \iff \mathit{MaxSequences}([\![\mathcal{N}]\!]_v) \subseteq \mathit{MaxSequences}([\![\mathcal{N}]\!]_{v_0}) \,.$$

Example 2.7. *Consider the PTPN $\mathcal{N}$ depicted in Fig. 3. Consider v_0 such that $a_0 = 0$, $b_0 = 3$, $a_1 = 0$, $b_1 = 1$, $a_2 = 2$, $b_2 = 3$, $a_3 = 1$, $b_3 = 2$. In $[\![\mathcal{N}]\!]_{v_0}$, transition t_0 can never fire, because t_1 must fire before one time unit, whereas transition t_2 can only fire after at least two time units. More precisely, the only (maximal) sequence of transitions allowed in $[\![\mathcal{N}]\!]_{v_0}$ is t_1, then t_2 and then t_3, after which the system cannot evolve.*

Applying IM^K to $\mathcal{N}$ and v_0 gives (besides $a_i \leq b_i$ for $0 \leq i \leq 3$) the constraint $b_1 < a_2$. This requires t_1 to fire strictly before t_2.

3. PARTIAL ORDER SEMANTICS

The inverse method with direct return IM^K allows only valuations v such that all the sequences of $[\![\mathcal{N}]\!]_v$ are also sequences of $[\![\mathcal{N}]\!]_{v_0}$. This can be seen as too rigid. Consider again the PTPN of Fig. 3. Because the initial parameter valuation v_0 is such that $b_1 < a_2$, the constraint output by IM^K forces this ordering and allows only valuations for which the only maximal sequence possible is (t_1, t_2, t_3), like in $[\![\mathcal{N}]\!]_{v_0}$.

With other parameter valuations (recall that we assume $a_i \leq b_i$ for $i \in \{1, 2, 3\}$), three other maximal sequences appear, viz., (t_2, t_1, t_3), (t_2, t_3, t_1) and (t_2, t_3, t_0). It is reasonable that a parameter synthesis method prevents valuations of the parameters which allow the last sequence, because it fires t_0 which differs qualitatively from the reference behavior. But the other sequences do not fire any undesired transition; they just reorder the firing of t_1, t_2 and t_3. Observing carefully the model, one even remarks that t_1 is actually *concurrent* to t_2 and t_3, and that the sequences (t_2, t_1, t_3) and (t_2, t_3, t_1) are simply obtained by changing the index where t_1 is inserted in the sequence (t_2, t_3). For many applications, this change can be considered very minor and does not affect the correct behavior of the system. In the case of the asynchronous circuit of Example 1.1, a designer may want to replace a hardware gate with another one

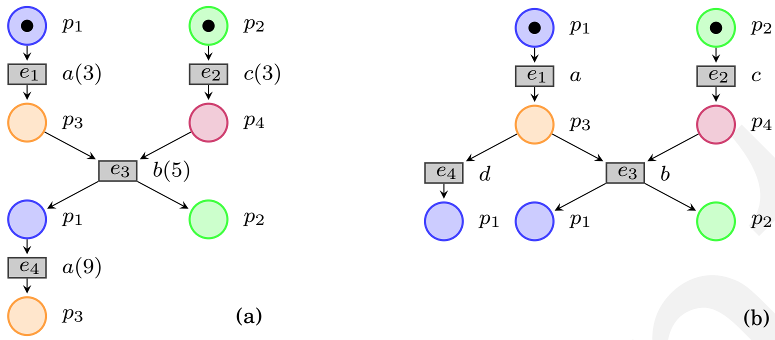

Fig. 4: (a) The graphical representation of a process of the TPN shown in Fig. 2; (b) an unfeasible abstract process of Fig. 2.

that has a different latency, provided the new system respects the correctness condition that the output signal Q never rises.

In this section, we formalize this intuition using partial-order semantics for TPNs. In Section 4, we will propose an alternative to IM^K which relaxes the inverse method to output a weaker constraint, i.e. a set of parameter valuations larger than in the original IM^K. The new method does not guarantee the preservation of the sequential behavior (sequences) but only of the *partial-order behavior* of the system.

3.1. Partial-Order Representation of Runs: Processes

A processes is a representation of an execution of a (time) Petri net. Executed actions (called events) are not totally ordered, as in timed words. For untimed Petri nets, only causality orders the events. For time Petri nets, the firing time of each event can still be represented together with the event, but the partial-order causality indicates the structural dependencies between events due to creation and consumption of tokens.

An execution of a TPN N is represented as a labeled acyclic Petri net where every transition (called *event* and labeled by a transition t of N and a firing date) stands for an occurrence of t, and every place (called *condition* and labeled by a place p of N) refers to a token produced by an event in place p or to a token of the initial marking. The arcs represent the creation and consumption of tokens. Because fresh conditions are created for the tokens created by each event, every condition has either no input arc (if it is an initial condition) or a single input arc, coming from the event that created the token. Symmetrically, each place has no more than one output arc since a token can be consumed by only one event in an execution.

Figure 4 (a) shows an example process. This process corresponds to the sequential execution $\big((a,3),(c,3),(b,5),(a,9)\big)$. The dates of the events are in parentheses. Observe that the process also represents the timed word $\big((c,3),(a,3),(b,5),(a,9)\big)$.

Below, we will define the processes of a safe TPN as the image of a mapping Π from its timed words to their partial-order representation as processes. The resulting processes are those described in [Aura and Lilius 2000].

3.1.1. Coding of Events and Conditions. Formally defining the processes of a TPN requires to formalize the notion of event. We use a canonical coding like in [Engelfriet 1991]. Each process will be a set $\mathcal{E}$ of pairs $(e, \theta(e))$, where e is an *event* and $\theta(e) \in \mathbb{R}$ is its firing date. We denote by $E_{\mathcal{E}}$ (or simply E) the set of events in $\mathcal{E}$. Each event e is itself a pair $({}^\bullet e, \tau(e))$ that codes an occurrence of the transition $\tau(e)$ in the process. The preset ${}^\bullet e$ is a set of conditions. Conditions are of the form $({}^\bullet b, \pi(b))$, and encode the arrival of a token created by the event ${}^\bullet b$ into place $\pi(b)$. Observe how the definition uses *mutual recursivity* to define the identity of events and conditions.

We illustrate this coding in Fig. 4 (a). The initial condition, labeled with p_1, is coded as $(\bot, p_1)$. Event e_1 (labeled with a) is coded as $\big(\{(\bot, p_1)\}, a\big)$. Its output condition is coded as (e_1, p_3). Event e_2 as $\big(\{(\bot, p_2)\}, c\big)$. And e_3 as $\big(\{(e_1, p_3), (e_2, p_4)\}, b\big)$.

We say that the event $e \overset{\text{def}}{=} ({}^\bullet e, \tau(e))$ *consumes* the conditions in ${}^\bullet e$. Symmetrically the set $\{(e,p) \mid p \in \tau(e)^\bullet\}$ of conditions *created* by e is denoted by $e^\bullet$. A virtual initial event $\bot$ is used as preset for initial conditions. We define $\bot^\bullet \overset{\text{def}}{=} \{\bot\} \times M_0$ and $\theta(\bot) \overset{\text{def}}{=} 0$.

We summarize the coding of events by defining the *event domain* D_N of a TPN N. The set D_N overapproximates the set of all events generated by the behavior of N.

Definition 3.1 (D_N)**.** *We define D_N as the smallest set such that for every $B \subseteq \bigcup_{e \in D_N \cup \{\bot\}} e^\bullet$ and for every $t \in T$, if $\pi(B) = {}^\bullet t$, then the event $(B, t) \in D_N$. Notice that this inductive definition is initialized by the fact that the initial conditions are in $\bigcup_{e \in D_N \cup \{\bot\}} e^\bullet$.*

For every set $E \subseteq D_N$ of events, we denote by $\uparrow(E)$ the set $\bigcup_{e \in E \cup \{\bot\}} e^\bullet \setminus \bigcup_{e \in E} {}^\bullet e$ of conditions that have been created by an event of E, and not consumed by any of them. For a process $\mathcal{E} \subseteq D_N$, notation $\uparrow(E_\mathcal{E})$ represents the set of conditions that remain at the end of the process.

We now have all necessary tools to define the process semantics of a TPN N. We define the processes of N by mapping every timed word of N into a set of timed events in $D_N \times \mathbb{R}$ (a process). Function $\Pi\colon (T \times \mathbb{R})^* \to 2^{D_N \times \mathbb{R}}$ in the following definition formalizes this mapping.

Definition 3.2. *Function Π maps each finite timed word $((t_1, \theta_1), \ldots, (t_n, \theta_n))$ of a safe TPN N to a process (set of events), as follows:*

— $\Pi(\epsilon) \overset{\text{def}}{=} \emptyset$

— $\Pi\big((t_1, \theta_1), \ldots, (t_{n+1}, \theta_{n+1})\big) \overset{\text{def}}{=} \mathcal{E} \cup \{(e, \theta_{n+1})\}$, *where* $\mathcal{E} \overset{\text{def}}{=} \Pi\big((t_1, \theta_1), \ldots, (t_n, \theta_n)\big)$ *and event* $e \overset{\text{def}}{=} (\{b \in \uparrow(E_\mathcal{E}) \mid \pi(b) \in {}^\bullet t_{n+1}\}, t_{n+1})$ *represents the last firing of the sequence.*

Clearly, Π is increasing w.r.t. the prefix order for (timed) words and the inclusion order for processes (which we also call prefix): for any timed word $\sigma \cdot \sigma'$, $\Pi(\sigma) \subseteq \Pi(\sigma \cdot \sigma')$. This allows us to define Π for infinite timed words as a limit.

A set $\mathcal{E} \subseteq D_N \times \mathbb{R}$ of dated events is a process *of a TPN N iff it is the image by Π of a timed word of N.*

For every condition $b \in \uparrow(E_\mathcal{E})$, the date of birth of the token in place $p = \pi(b)$ after a process $\mathcal{E}$ is $dob_\mathcal{E}(p) \overset{\text{def}}{=} \theta({}^\bullet b)$. This allows us to define the state that is reached after a finite process $\mathcal{E}$ of N as: $RS(\mathcal{E}) \overset{\text{def}}{=} (\pi(\uparrow(E)), dob_\mathcal{E}, \max_{e \in E \cup \{\bot\}} \theta(e))$.

Finally, we define the relation $\to$ on the events as: $e \to e' \iff e^\bullet \cap {}^\bullet e' \neq \emptyset$. The reflexive transitive closure $\to^*$ of $\to$ is called the *causality* relation. Two events of a process that are not causally related are called *concurrent*. For every event e, we denote by $\lceil e \rceil \overset{\text{def}}{=} \{f \in D_N \mid f \to^* e\}$ the *causal past* of e, and for any set $E \subseteq D_N$ of events, $\lceil E \rceil \overset{\text{def}}{=} \bigcup_{e \in E} \lceil e \rceil$.

3.2. Characterization of Processes

Since timed processes are defined as sets of dated events, a natural problem is to decide whether an arbitrary set of dated events $\mathcal{E} \subseteq D_N \times \mathbb{R}$ is a process. The answer is nontrivial and was treated in [Aura and Lilius 2000]. We give a summary here. The following lemma shows that the events present in any process of a TPN guarantee certain structural relations:

Lemma 3.3. *Let N be a safe TPN. For every process $\mathcal{E}$ of N, the set E of events in $\mathcal{E}$ is a subset of D_N and satisfies:*

— $\lceil E \rceil = E$ *(i.e. E is causally closed) and*
— $\nexists e, e' \in E \quad e \neq e' \ \wedge\ {}^\bullet e \cap {}^\bullet e' \neq \emptyset$ *(E is said* conflict free*).*

Proof. When a new event e is added to the set E of events of a process (see Definition 3.2), all the conditions in ${}^\bullet e$ are final conditions of E. This implies that the causal predecessors of e are in E and that e is not in conflict with any event of E. We conclude by induction on the size of the process: if E is causally closed and conflict free, then $E \cup \{e\}$ also is. □

Definition 3.4 (Abstract process, $\mathit{Processes}(N)$, $\mathit{MaxProcesses}(N)$)**.** *Let N be a TPN. A set of events $E \subseteq D_N$ is an* abstract process *of N iff it is causally closed and conflict free. It is* feasible *if additionally there exists some process $\mathcal{E}$ of N such that $E = E_{\mathcal{E}}$.*

We denote by $\mathit{Processes}(N)$ the set of feasible abstract processes of N. Also, we denote by $\mathit{MaxProcesses}(N)$ the set of $\subseteq$-maximal processes in $\mathit{Processes}(N)$.

Finally, let $\mathcal{N}$ bet a PTPN. An abstract process *of $\mathcal{N}$ is any set $E \subseteq D_{\mathcal{N}}$ of events that is causally closed and conflict free (identical definition to that of TPNs).*

Abstract processes are the untimed partial-orders of events that satisfy the same structural properties (causally close, conflict free) as for processes of a TPN (Lemma 3.3). In a TPN, some (but potentially not all) abstract processes are feasible. The untimed support of the process shown in Fig. 4 (a) is obviously feasible. For the TPN in Fig. 2, an unfeasible abstract process is show in Fig. 4 (b). The process is unfeasible because events e_3 and e_4 are in conflict (both consume one same condition).

In a PTPN, an abstract process might be feasible for an instantiation with one parameter valuation and unfeasible for the instantiation with a different one. For instance, let $\mathcal{N}$ be the PTPN shown in Fig. 3. Let v be a parameter valuation mapping $\langle a_1, b_1 \rangle$ to $\langle 0, 9 \rangle$ and $\langle a_i, b_i \rangle$ to $\langle 1, 1 \rangle$ for $i \neq 1$. That is, transition t_1 has plenty of time to fire. As a result, $\mathcal{E} \stackrel{\text{def}}{=} \Pi((t_2, 1), (t_3, 2), (t_0, 3))$ is a process of $[\![\mathcal{N}]\!]_v$, and the untimed support of $\mathcal{E}$ is a feasible abstract process of $[\![\mathcal{N}]\!]_v$. Such abstract process might be unfeasible for other parameter valuations, e.g. if t_1 is required to fire before t_0 can do it. Consider parameter valuation v', mapping $\langle a_1, b_1 \rangle$ to $\langle 0, 1 \rangle$ and $\langle a_i, b_i \rangle$ to $\langle 1, 1 \rangle$ for $i \neq 1$. Now t_1 needs to fire much earlier. As a result, $\mathcal{E}$ is not a process of $[\![\mathcal{N}]\!]_{v'}$.

The following lemma characterizes the possible firing dates for the events of an abstract process under the time constraints of a TPN N. Before we present the lemma, let us introduce new notation. For an abstract process $E \subseteq D_N$, we denote by $\mathit{ConflictingExtensions}(E)$ the set of events $e \in D_N \setminus E$ that were eventually enabled during the process (${}^\bullet e \subseteq \bigcup_{f \in E \cup \{\perp\}} f^\bullet$) but did not fire because they were eventually disabled by an event $f \in E$ such that ${}^\bullet e \cap {}^\bullet f \neq \emptyset$.

Lemma 3.5 (Possible dates for a finite abstract process)**.** *Let N be a safe TPN. Let $\mathcal{E} \subseteq D_N \times \mathbb{R}$ be a finite set of dated events such that the set E of events in $\mathcal{E}$ is an abstract process. Then $\mathcal{E}$ is a process of N iff:*

— *Firing delays are met, that is,*

$$\forall e \in E \quad \mathit{efd}(\tau(e)) \leq \theta(e) - \mathit{doe}(e) \leq \mathit{lfd}(\tau(e)),$$

where $\mathit{doe}(e) \stackrel{\text{def}}{=} \max_{b \in {}^\bullet e} \theta({}^\bullet b)$ is the date when the event e was enabled;

— *Events eventually enabled by E but later disabled by some other event in E are required to not overtake their latest firing delay (notice that this concerns events which are* not *in E):*

$$\forall e \in \mathit{ConflictingExtensions}(E) \quad \mathit{dod}(e) \leq \mathit{doe}(e) + \mathit{lfd}(\tau(e)),$$

where $\mathit{dod}(e) \overset{\text{def}}{=} \min\{\theta(f) \mid f \in E \ \wedge \ {}^{\bullet}f \cap {}^{\bullet}e \neq \emptyset\}$ is the date when e was disabled (because an event f consumed one condition in ${}^{\bullet}e$);

— *Events enabled at the end of the process did not overtake their latest firing delay:*

$$\forall e \in D_N \quad {}^{\bullet}e \subseteq \uparrow(E) \implies \theta_{\mathit{end}} \leq \mathit{doe}(e) + \mathit{lfd}(\tau(e))$$

where $\theta_{\mathit{end}} \overset{\text{def}}{=} \max_{f \in E \cup \{\perp\}} \theta(f)$ is the date that is reached at the end of the process.

The proof can be found in [Aura and Lilius 2000].

Let $E = \{e_1, \ldots, e_n\} \subseteq D_N$ be an abstract process of some TPN N. The conditions in Lemma 3.5 can be summarized in the following constraint K_E^{θ} over Θ (precisely, on the variables $\theta(e_1), \ldots, \theta(e_n)$). The result is that a valuation that assigns values $\theta_1, \ldots, \theta_n \in \mathbb{R}$ to the variables $\theta(e_i)$, satisfies K_E^{θ} iff $\{(e_1, \theta_1), \ldots, (e_n, \theta_n)\}$ is a process of N.

Definition 3.6 (K_E^{θ})**.** *We denote by K_E^{θ} the constraint on the $\theta(e)$, $e \in E$, defined as the conjunction of the following:*

— $\bigwedge_{e \in E} \mathit{efd}\big(\tau(e)\big) \leq \theta(e) - \mathit{doe}(e) \leq \mathit{lfd}\big(\tau(e)\big)$
— $\bigwedge_{e \in \mathit{ConflictingExtensions}(E)} \mathit{dod}(e) \leq \mathit{doe}(e) + \mathit{lfd}\big(\tau(e)\big)$
— $\bigwedge_{e \in D_N, {}^{\bullet}e \subseteq \uparrow(E)} \theta_{\mathit{end}} \leq \mathit{doe}(e) + \mathit{lfd}\big(\tau(e)\big)$

Notice that the notations $\mathit{doe}(e)$, $\mathit{dod}(e)$ and θ_{end} hide terms of the form $\max\{\ldots\}$ and $\min\{\ldots\}$. Inequalities containing such terms can be expanded to linear constraints over $\Theta \cup \Lambda$. For instance the inequality $\theta_{\text{end}} \leq \mathit{doe}(e) + \mathit{lfd}\big(\tau(e)\big)$ becomes

$$\bigwedge_{f \in E} \bigvee_{b \in {}^{\bullet}e} \theta(f) \leq \theta({}^{\bullet}b) + \mathit{lfd}\big(\tau(e)\big).$$

Furthermore, in the definition of θ_{end}, it is sufficient to consider only the set $\mathit{maxEvents}_E$ of events which are maximal in E w.r.t. $\to$. We get

$$\bigwedge_{f \in \mathit{maxEvents}_E} \bigvee_{b \in {}^{\bullet}e} \theta(f) \leq \theta({}^{\bullet}b) + \mathit{lfd}\big(\tau(e)\big).$$

Example 3.7. *Consider the process shown in Fig. 4 (a). Replace the firing dates by variables $\theta(e_1)$, $\theta(e_2)$, $\theta(e_3)$, $\theta(e_4)$. The following constraints describe the set all possible processes that share the same partial-order structure (abstract process) as that of Fig. 4 (a). Notice that only one event in the process is maximal w.r.t. $\to$, hence $\theta_{\mathit{end}} = \theta(e_4)$.*

$$\begin{cases} 0 \leq \theta(e_1) \leq \infty & \textit{(firing delay of } e_1\textit{)} \\ 3 \leq \theta(e_2) \leq 4 & \textit{(firing delay of } e_2\textit{)} \\ 0 \leq \theta(e_3) - \max\{\theta(e_1), \theta(e_2)\} \leq 5 & \textit{(firing delay of } e_3\textit{)} \\ 0 \leq \theta(e_4) - \theta(e_3) \leq \infty & \textit{(firing delay of } e_4\textit{)} \\ \theta(e_3) \leq \theta(e_1) + 4 & \textit{(occurrence of } d \textit{ enabled after } e_1 \textit{ and disabled by } e_3\textit{)} \\ \theta(e_4) \leq \theta(e_3) + 4 & \textit{(occurrence of } c \textit{ enabled at the end of the process)} \\ \theta(e_4) \leq \theta(e_4) + 4 & \textit{(occurrence of } d \textit{ enabled at the end of the process)} \end{cases}$$

The first four constraints regard the firing delays of events in E and are fairly obvious. While less intuitive, the last three are also necessary to correctly characterize the set of all allowed firing dates for events.

4. PRESERVING PARTIAL ORDER RUNS

In this section, we define parameter constraints for abstract processes (Section 4.1). We then introduce our method $\mathsf{IM}^K\mathsf{PO}$ (Section 4.2). We then design two extensions of the methods: one designed for systems with a limited cyclicity (Section 4.3) and one for general cyclic systems (Section 4.4).

4.1. Constraint on Parameters for an Abstract Process

We can now come back to our parameter synthesis problem. We consider a *parametric* TPN $\mathcal{N}$. Given an abstract process E of $\mathcal{N}$, the first step is to find a constraint K on the parameters such that for every valuation v of the parameters it holds that $E \in \mathit{MaxProcesses}(\llbracket \mathcal{N} \rrbracket_v)$ iff $v \models K$.

We first generalize the constraint K_E^θ and replace the instantiated values of the $\mathit{efd}(t)$ and $\mathit{lfd}(t)$ by the parameters given by the PTPN. We get a constraint over *both* the parameters of the model and the $\theta(e)$, $e \in E$, i.e. a constraint over $\Theta \cup \Lambda$.

Definition 4.1. *For an abstract process E of PTPN $\mathcal{N}$, we define $K_E^{\theta\lambda}$ as:*

— $\bigwedge_{e \in E} \mathit{pefd}\big(\tau(e)\big) \leq \theta(e) - \mathit{doe}(e) \leq \mathit{plfd}\big(\tau(e)\big)$ *(1)*
— $\bigwedge_{e \in \mathit{ConflictingExtensions}(E)} \mathit{dod}(e) \leq \mathit{doe}(e) + \mathit{plfd}\big(\tau(e)\big)$ *(2)*
— $\bigwedge_{e \in D_N, {}^\bullet e \subseteq \uparrow(E)} \theta_{\mathit{end}} \leq \mathit{doe}(e) + \mathit{plfd}\big(\tau(e)\big)$ *(3)*

For instance, the PTPN of Fig. 3 has two maximal abstract processes: one where transitions t_1, t_2 and t_3 fire (giving rise to, resp., events e_1, e_2, e_3), the second with t_2, t_3 and t_0 (giving rise to, resp., events e_2, e_3 and e_0). With the reference valuation of the parameters v_0 where $a_0 = 0$, $b_0 = 3$, $a_1 = 0$, $b_1 = 1$, $a_2 = 2$, $b_2 = 3$, $a_3 = 1$ and $b_3 = 2$, only the first abstract process $\{e_1, e_2, e_3\}$ can be executed.

The constraints for these abstract processes are

$$K^{\theta\lambda}_{\{e_1,e_2,e_3\}} \stackrel{\text{def}}{=} \begin{cases} a_1 \leq \theta(e_1) \leq b_1 & \text{(firing delay of } e_1\text{)} \\ a_2 \leq \theta(e_2) \leq b_2 & \text{(firing delay of } e_2\text{)} \\ a_3 \leq \theta(e_3) - \theta(e_2) \leq b_3 & \text{(firing delay of } e_3\text{)} \\ \theta(e_1) \leq \theta(e_3) + b_0 & \text{(occurrence of } t_0 \text{ enabled by } e_3 \text{ disabled by } e_1\text{)} \end{cases}$$

and

$$K^{\theta\lambda}_{\{e_2,e_3,e_0\}} \stackrel{\text{def}}{=} \begin{cases} a_2 \leq \theta(e_2) \leq b_2 & \text{(firing delay of } e_2\text{)} \\ a_3 \leq \theta(e_3) - \theta(e_2) \leq b_3 & \text{(firing delay of } e_3\text{)} \\ a_0 \leq \theta(e_0) - \theta(e_3) \leq b_0 & \text{(firing delay of } e_0\text{)} \\ \theta(e_0) \leq b_1 & \text{(occurrence of } t_1 \text{ disabled by } e_0\text{)} \end{cases}$$

We can check that, with v_0, there exists a valuation for the dates $\theta(e_1), \theta(e_2), \theta(e_3)$ which satisfies the constraint $K^{\theta\lambda}_{\{e_1,e_2,e_3\}}$ (take for instance $\theta(e_1) = 0$, $\theta(e_2) = 2$ and $\theta(e_3) = 3$) but there exists no valuation of the dates $\theta(e_2), \theta(e_3), \theta(e_0)$ satisfying $K^{\theta\lambda}_{\{e_2,e_3,e_0\}}$ (the constraint implies $a_2 + a_3 + a_0 \leq \theta(e_0) \leq b_1$). This confirms that $\{e_1, e_2, e_3\}$ is the only maximal abstract process feasible in $\llbracket \mathcal{N} \rrbracket_{v_0}$.

What matters for our parameter synthesis problem is not the values of the firing dates of the events of a process, but rather the condition on the parameters under which an abstract process is feasible for *some* firing dates. Using variable elimination techniques (e.g. Fourier-Motzkin), we can compute for an abstract process $E = \{e_1, \dots, e_n\}$, a constraint equivalent to $\exists\theta(e_1) \dots \exists\theta(e_n)\ K_E^{\theta\lambda}$.

Definition 4.2. *Let $E = \{e_1, \dots, e_n\} \subseteq D_N$ be an abstract process of a PTPN $\mathcal{N}$. We define the constraint K_E^λ on the parameters of $\mathcal{N}$ as the result of eliminating the variables $\theta(e_1), \dots, \theta(e_n)$ in the constraint $K_E^{\theta\lambda}$.*

The constraint K_E^λ characterizes a set of parameter valuations v such that the instantiated model $\llbracket \mathcal{N} \rrbracket_v$ can execute the abstract process E. Coming back to the example of Fig. 3, for the abstract process $\{e_1, e_2, e_3\}$, we get the constraint:

$$K^{\lambda}_{\{e_1,e_2,e_3\}} \stackrel{\text{def}}{=} \begin{cases} \quad a_1 \leq b_2 + b_3 + b_0 \\ \wedge \ a_1 \leq b_1 \wedge a_2 \leq b_2 \wedge a_3 \leq b_3 \end{cases}$$

The first line means that t_1 is able to fire before t_0 reaches its latest firing delay. The second line simply means that the firing intervals of the transitions are nonempty.

For the abstract process $\{e_2, e_3, e_0\}$, the constraint $K^{\lambda}_{\{e_2,e_3,e_0\}}$ is $a_2+a_3+a_0 \leq b_1$ (omitting the conditions about the firing intervals). Notice that $K^{\lambda}_{\{e_2,e_3,e_0\}}$ and $K^{\lambda}_{\{e_1,e_2,e_3\}}$ do not exclude each other, which means that there are parameter valuations v for which the instantiated TPN $[\![\mathcal{N}]\!]_v$ can execute both abstract processes.

4.2. Parameter Synthesis Preserving Partial Order Semantics

We now have all the necessary bricks to define our procedure $\mathsf{IM}^K\mathsf{PO}$ (standing for "inverse method based on partial-orders") for synthesizing parameters in a PTPN $\mathcal{N}$ that guarantee the preservation of partial-order semantics. More precisely, given $\mathcal{N}$ and v_0 we are looking for a constraint on the parameters Λ of $\mathcal{N}$ guaranteeing that the set of maximal processes of $[\![\mathcal{N}]\!]_{v_0}$ contains the set of maximal processes of $[\![\mathcal{N}]\!]_v$ for any v satisfying the constraint. Note that this requirement concerns only *maximal* processes: asking for preservation of all processes would limit the freedom in the interleavings of concurrent transitions. For the PTPN of Fig. 3, the only (maximal) sequence feasible with the initial valuation v_0 (given above) is (t_1, t_2, t_3). Consider another valuation v that would force (t_2, t_1, t_3) (which we consider correct). A (non-maximal) timed word with only t_2 yields a (non-maximal) abstract process which is *not* feasible under v_0. On the other hand, the *maximal* abstract processes are the same for both valuations.

The first version of our $\mathsf{IM}^K\mathsf{PO}$ procedure terminates for PTPNs where all the abstract processes are finite. It relies on the computation of the *unfolding* of the untimed support of the PTPN: the unfolding is a compact representation of all the processes of an (untimed) Petri net, which corresponds to the superimposition of all feasible processes (see Fig. 6). Efficient tools exist for computing unfoldings [Khomenko 2012; Schwoon 2014]. The procedure $\mathsf{IM}^K\mathsf{PO}(\mathcal{N}, v_0)$ operates as follows:

(1) Compute the unfolding of the untimed support of $\mathcal{N}$ (i.e. the Petri net obtained from $\mathcal{N}$ by removing all the temporal constraints efd and lfd). The unfolding has finite depth when the length of the abstract processes is bounded; hence it can be computed entirely.
(2) Extract the set MP of maximal processes[2]; they are the abstract processes of our PTPN $\mathcal{N}$.
(3) For every $E \in MP$, construct the constraint K^{λ}_E on the parameters of $\mathcal{N}$ under which the process is feasible.
(4) Output the conjunction of the initial constraint K_0 (coming from $\mathcal{N}$) with the negation of all constraints associated to processes which are not feasible under v_0:

$$K_0 \wedge \bigwedge_{E \in MP, \text{ with } v_0 \not\models K^{\lambda}_E} \neg K^{\lambda}_E \ .$$

Theorem 4.3. *Let $\mathcal{N}$ be a PTPN, let v_0 be a parameter valuation. Assume $\mathsf{IM}^K\mathsf{PO}(\mathcal{N}, v_0)$ terminates with result K. Then for all valuation v of the parameters satisfying the*

[2]The maximal processes can be extracted for instance by a SAT solver using an appropriate SAT encoding, or using the optimal partial-order reduction algorithm of [Rodríguez et al. 2015].

initial constraint K_0 of the model,

$$v \models K \iff \mathit{MaxProcesses}(\llbracket\mathcal{N}\rrbracket_v) \subseteq \mathit{MaxProcesses}(\llbracket\mathcal{N}\rrbracket_{v_0}) \,.$$

In particular $v_0 \models K$.

Proof. Let v be a parameter valuation such that $\mathit{MaxProcesses}(\llbracket\mathcal{N}\rrbracket_v) \subseteq \mathit{MaxProcesses}(\llbracket\mathcal{N}\rrbracket_{v_0})$. For every maximal process $E = \{e_1, \dots, e_n\} \in \mathit{MP}$ we have that $v_0 \not\models K_E^\lambda$ implies that there exists no valuation $\theta_1, \dots, \theta_n \in \mathbb{R}$ of the variables $\theta(e_1), \dots, \theta(e_n)$ such that $\{(e_1, \theta_1), \dots, (e_n, \theta_n)\}$ is a process of $\llbracket\mathcal{N}\rrbracket_{v_0}$. Then $E \notin \mathit{MaxProcesses}(\llbracket\mathcal{N}\rrbracket_{v_0})$. We deduce that E is not a maximal abstract process of $\llbracket\mathcal{N}\rrbracket_v$; actually it cannot be a non-maximal process either: this would mean that a transition t is enabled at the end, and this transition would also make E non-maximal for $\llbracket\mathcal{N}\rrbracket_{v_0}$ since no valuation of the parameters can prevent the system from firing transitions when transitions are enabled, except valuations which make the firing intervals empty, which is excluded by assumption. Hence $v \not\models K_E^\lambda$, i.e. $v \models \neg K_E^\lambda$. As a result $v \models \bigwedge_{E \in MP,\, v_0 \not\models K_E^\lambda} \neg K_E^\lambda$, and because v satisfies K_0, it satisfies K.

Now, let v be a valuation such that $\mathit{MaxProcesses}(\llbracket\mathcal{N}\rrbracket_v) \not\subseteq \mathit{MaxProcesses}(\llbracket\mathcal{N}\rrbracket_{v_0})$. Let E be an abstract process in $\mathit{MaxProcesses}(\llbracket\mathcal{N}\rrbracket_v) \setminus \mathit{MaxProcesses}(\llbracket\mathcal{N}\rrbracket_{v_0})$. Then $v_0 \not\models K_E^\lambda$ (which implies that $\neg K_E^\lambda$ appears in the conjunction K) and, on the other hand $v \models K_E^\lambda$, Hence $v \not\models K$. $\square$

Example 4.4. *For the PTPN of Fig. 3, we already said that there are two maximal abstract processes $\{e_1, e_2, e_3\}$ and $\{e_2, e_3, e_0\}$. Only the first one is feasible in $\llbracket\mathcal{N}\rrbracket_{v_0}$, i.e., $v_0 \not\models K_{\{e_2,e_3,e_0\}}^\lambda$. Then our procedure* $\mathsf{IM}^K\mathsf{PO}$ *outputs the constraint $K_0 \wedge a_2 + a_3 + a_0 > b_1$, which is the negation of $K_{\{e_2,e_3,e_0\}}^\lambda$. Remember that K_0 is assumed to specify at least that the firing intervals are nonempty. Notice that this constraint is much more permissive than the constraint $a_2 > b_1$ output by* IM^K. *While* IM^K *requires t_1 to fire strictly before t_2,* $\mathsf{IM}^K\mathsf{PO}$ *only requires that it fires before being disabled by t_0.*

Let us now show that the output of $\mathsf{IM}^K\mathsf{PO}$ is always equally or more permissive than the output of IM^K.

Theorem 4.5. *Let $\mathcal{N}$ be a PTPN with only finite executions, and let v_0 be a parameter valuation. Denote K_{IM^K} the constraint output by* IM^K *and $K_{\mathsf{IM}^K\mathsf{PO}}$ the constraint output by* $\mathsf{IM}^K\mathsf{PO}$. *Then $\{v_0\} \subseteq \{v \mid v \models K_{\mathsf{IM}^K}\} \subseteq \{v \mid v \models K_{\mathsf{IM}^K\mathsf{PO}}\}$.*

Proof. By Theorem 2.6, $\{v_0\} \subseteq \{v \mid v \models K_{\mathsf{IM}^K}\}$. Now, let v be a parameter valuation satisfying K_{IM^K}. Again by Theorem 2.6, $\mathit{MaxSequences}(\llbracket\mathcal{N}\rrbracket_v) \subseteq \mathit{MaxSequences}(\llbracket\mathcal{N}\rrbracket_{v_0})$. We show that $\mathit{MaxProcesses}(\llbracket\mathcal{N}\rrbracket_v) \subseteq \mathit{MaxProcesses}(\llbracket\mathcal{N}\rrbracket_{v_0})$: indeed, every maximal abstract process E feasible for $\llbracket\mathcal{N}\rrbracket_v$ is the image by Π of a maximal timed word $((t_1, \theta_1), \dots, (t_n, \theta_n))$ feasible for $\llbracket\mathcal{N}\rrbracket_v$, whose corresponding time-abstract word $(t_1, \dots, t_n)$ is in $\mathit{MaxSequences}(\llbracket\mathcal{N}\rrbracket_v)$. Because $\mathit{MaxSequences}(\llbracket\mathcal{N}\rrbracket_v) \subseteq \mathit{MaxSequences}(\llbracket\mathcal{N}\rrbracket_{v_0})$, we have that $(t_1, \dots, t_n)$ is also in $\mathit{MaxSequences}(\llbracket\mathcal{N}\rrbracket_{v_0})$, i.e. there exist dates $\theta'_1, \dots, \theta'_n$ (notice that they are not necessarily the same as the θ_i) such that $((t_1, \theta'_1), \dots, (t_n, \theta'_n))$ is feasible for $\llbracket\mathcal{N}\rrbracket_{v_0}$. The image by Π of this timed word is a process of $\llbracket\mathcal{N}\rrbracket_{v_0}$ whose set of events (determined only by the time-abstract word) is E. Then $E \in \mathit{MaxProcesses}(\llbracket\mathcal{N}\rrbracket_{v_0})$.

To conclude, $\mathit{MaxProcesses}(\llbracket\mathcal{N}\rrbracket_v) \subseteq \mathit{MaxProcesses}(\llbracket\mathcal{N}\rrbracket_{v_0})$, and by Theorem 4.3, $v \models K_{\mathsf{IM}^K\mathsf{PO}}$. $\square$

4.3. An Alternative Method for Restricted Cyclic Models

Our method $\mathsf{IM}^K\mathsf{PO}$ first constructs all maximal processes, and then infers parameter valuations to preserve partial orders. For cyclic systems, this method will not terminate. We address here the case of systems that may be cyclic for some parameter valuations (i.e. the Petri net is not structurally acyclic), but are acyclic for the reference valuation v_0.

We propose now an alternative method $\mathsf{IM}^K\mathsf{PO}'(\mathcal{N}, v_0)$, that avoids computing the entire unfoldings of the untimed Petri net, but explores only the processes that exist in $[\![\mathcal{N}]\!]_{v_0}$:

(1) Compute the (finite) set $\mathit{MaxProcesses}([\![\mathcal{N}]\!]_{v_0})$ of maximal abstract processes feasible for $[\![\mathcal{N}]\!]_{v_0}$; one way to do this is to compute the finite set $\mathit{MaxSequences}([\![\mathcal{N}]\!]_{v_0})$, and then represent every sequence as a process, as explained in Definition 3.2.
(2) For every $E \in \mathit{MaxProcesses}([\![\mathcal{N}]\!]_{v_0})$, for every causally closed subset E' of E (called a prefix of E), and for every event $e \in D_N$ which extends E' (i.e. ${}^\bullet e \subseteq \uparrow(E')$) such that the abstract process $E' \cup \{e\}$ is not the prefix of any abstract process of $[\![\mathcal{N}]\!]_{v_0}$, compute the constraint $K^\lambda_{E' \cup \{e\}}$.
(3) Return the conjunction of K_0 with the negation of all the constraints $K^\lambda_{E' \cup \{e\}}$.

Notice that not all prefixes E' of maximal abstract processes feasible for $[\![\mathcal{N}]\!]_{v_0}$ are feasible abstract processes for $[\![\mathcal{N}]\!]_{v_0}$: for the PTPN of Fig. 3, the abstract process containing only the occurrence of t_2 and the occurrence of t_3 is not feasible for $[\![\mathcal{N}]\!]_{v_0}$ because t_1 fires earlier than t_2. Still E' must be considered in order to prevent its extension by t_0 which is not a prefix of any feasible abstract process of $[\![\mathcal{N}]\!]_{v_0}$.

This alternative approach $\mathsf{IM}^K\mathsf{PO}'$ returns the negation of the parametric constraints associated to the extension by one event of any prefix of a process of $[\![\mathcal{N}]\!]_{v_0}$. As a consequence, it avoids the full exploration of the part of the state space that does not correspond to admissible behaviors in $[\![\mathcal{N}]\!]_{v_0}$. In fact, this alternative approach is closer to the spirit of the original inverse method, that also proceeds with a limited exploration of the state space.

Theorem 4.6. *Let $\mathcal{N}$ be a PTPN, and let v_0 be a parameter valuation for which $[\![\mathcal{N}]\!]_{v_0}$ has only finite executions. Let $K = \mathsf{IM}^K\mathsf{PO}'(\mathcal{N}, v_0)$. Then for all valuation v of the parameters satisfying the initial constraint K_0 of the model,*

$$v \models K \iff \mathit{MaxProcesses}([\![\mathcal{N}]\!]_v) \subseteq \mathit{MaxProcesses}([\![\mathcal{N}]\!]_{v_0}) \,.$$

In particular $v_0 \models K$.

Proof. Let v be a parameter valuation such that $\mathit{MaxProcesses}([\![\mathcal{N}]\!]_v) \subseteq \mathit{MaxProcesses}([\![\mathcal{N}]\!]_{v_0})$. Let E' be a prefix of an abstract process $E \in \mathit{MaxProcesses}([\![\mathcal{N}]\!]_{v_0})$ and e a possible extension of E' such that $E' \cup \{e\}$ is not the prefix of any abstract process of $[\![\mathcal{N}]\!]_{v_0}$. Because $\mathit{MaxProcesses}([\![\mathcal{N}]\!]_v) \subseteq \mathit{MaxProcesses}([\![\mathcal{N}]\!]_{v_0})$, $E' \cup \{e\}$ is not the prefix of any abstract process of $[\![\mathcal{N}]\!]_v$ either. Then $v \models \neg K^\lambda_{E' \cup \{e\}}$.

Now, let v be a parameter valuation such that $\mathit{MaxProcesses}([\![\mathcal{N}]\!]_v) \not\subseteq \mathit{MaxProcesses}([\![\mathcal{N}]\!]_{v_0})$. Let E be an abstract process in $\mathit{MaxProcesses}([\![\mathcal{N}]\!]_v) \setminus \mathit{MaxProcesses}([\![\mathcal{N}]\!]_{v_0})$. Compute a prefix E' of E by removing events one by one (starting by those that are maximal w.r.t. $\to$ so that E' remains causally closed) until E' becomes a prefix of an abstract process feasible for $[\![\mathcal{N}]\!]_{v_0}$. (If needed, remove all the events: $E' = \emptyset$ is suitable.) Then extend E' with the last event e that was removed. We have $v_0 \not\models K^\lambda_{E' \cup \{e\}}$ and $v \models K^\lambda_{E' \cup \{e\}}$, Hence $v \not\models K$. $\square$

As a consequence, when $\mathsf{IM}^K\mathsf{PO}$ can be applied, $\mathsf{IM}^K\mathsf{PO}$ and $\mathsf{IM}^K\mathsf{PO}'$ return equivalent constraints.

4.4. Block-Oriented Method for Handling Cyclic Models

We present now a method for handling general cyclic models. This method $\mathsf{IM}^K\mathsf{PO}_n^{\text{blocks}}(\mathcal{N}, v_0)$ considers the processes using blocks of n events, for some n given by the user, and proceeds as follows: it first considers the abstract processes having n events (plus those who have less than n events but are maximal); then it proceeds like $\mathsf{IM}^K\mathsf{PO}$ on those processes and computes a first constraint on the parameters. All the parameter valuations satisfying this constraint behave like $[\![\mathcal{N}]\!]_{v_0}$ on these processes. It remains to check that they behave well also for longer processes.

Then, for each non-maximal process E considered in the previous step and feasible for $[\![\mathcal{N}]\!]_{v_0}$, we consider again the abstract processes which have n events and start from the marking M reached at the end of E. Simply, the tokens in M may not have the same date of birth. Hence, the computation of the constraint for the processes starting at M must be adapted in order to take into account the possible age of the tokens. Since the possible ages depend on the parameter valuation, we need a constraint K over the parameters and over variables $\alpha_p, p \in M$ representing these ages. The constraint K is obtained as follows: add to the contraint $K_E^{\theta\lambda}$ the equalities $\alpha_p = \theta_{\text{end}} - \theta({}^\bullet b)$, for $p \in P$ and $b \in \uparrow(E)$ the final condition of E such that $\pi(b) = p$; then eliminate the variables $\theta(e)$, $e \in E$.

A marking M equipped with such constraint K is called a *symbolic state*. Actually a set of explored symbolic states will be stored during the procedure. Initially, for M_0, we have the constraint $\bigwedge_{p \in M_0} \alpha_p = 0$ (where the parameters do not appear).

We can now describe the full procedure $\mathsf{IM}^K\mathsf{PO}_n^{\text{blocks}}(\mathcal{N}, v_0)$:

(1) Initialize
- the constraint K on the parameters to K_0 and
- the set of visited symbolic states to $\mathcal{S} := (M_0, \bigwedge_{p \in M_0} \alpha_p = 0)$.

(2) Compute the set $\mathit{Processes}_n([\![\mathcal{N}]\!]_{v_0})$ of abstract processes having n events, plus those having less than n events but maximal; then proceed like $\mathsf{IM}^K\mathsf{PO}$ on those processes. Let K' be the obtained constraint on the parameters. Set $K := K \wedge K'$.

(3) For each non-maximal process E considered in the previous step and feasible for $[\![\mathcal{N}]\!]_{v_0}$:
- Compute the constraint K_E which expresses the possible age for the tokens in the marking M reached after E, depending on the valuation of the parameters.
- Let $\mathcal{S} := \mathcal{S} \cup (M, K_E)$.

(4) Iterate the method starting from the new symbolic states (i.e. explore the processes of n events feasible from the new symbolic states), until no new symbolic state is discovered. In order to start form a symbolic state (M, K_M), the constraint $K_E^{\theta\lambda}$ needs to be replaced by $(\exists(\alpha_p)_{p\in M}\ (K_E^{\theta\lambda\alpha} \wedge K_M))$, where $K_E^{\theta\lambda\alpha}$ is defined like $K_E^{\theta\lambda}$ except that, for every initial condition b, $\theta({}^\bullet b)$ is replaced by $-\alpha_{\pi(b)}$ (i.e. the date of birth of the token in p assuming that its age is α_p at time 0 when the process starts).

(5) When a fixpoint is reached, i.e. when no new symbolic state is computed, return the obtained constraint K.

Theorem 4.7. *Let $\mathcal{N}$ be a PTPN, let v_0 be a parameter valuation and let $n \in \mathbb{N}$. If $\mathsf{IM}^K\mathsf{PO}_n^{\text{blocks}}(\mathcal{N}, v_0)$ terminates and returns a constraint K, then for all valuations v of the parameters satisfying the initial constraint K_0 of the model,*

$$\mathit{MaxSequences}([\![\mathcal{N}]\!]_v) \subseteq \mathit{MaxSequences}([\![\mathcal{N}]\!]_{v_0}) \implies v \models K, \textit{ and}$$

$$v \models K \implies \mathit{MaxProcesses}(\llbracket \mathcal{N} \rrbracket_v) \subseteq \mathit{MaxProcesses}(\llbracket \mathcal{N} \rrbracket_{v_0})\,.$$

In particular $v_0 \models K$.

Proof. Let v be a parameter valuation satisfying K_0 and such that, $\mathit{MaxSequences}(\llbracket \mathcal{N} \rrbracket_v) \subseteq \mathit{MaxSequences}(\llbracket \mathcal{N} \rrbracket_{v_0})$. We show that v satisfies all the successive constraints K' added by the method and that, for every explored symbolic state (M, K_M) and for every maximal sequence σ, if there exists a valuation α of the $\alpha_p, p \in M$ such that

(1) K_M is satisfied by the valuation given by α (for the age of the tokens) together with v (for the parameters), and
(2) the sequence σ is feasible by $\llbracket \mathcal{N} \rrbracket_v$ starting from marking M with the tokens aged according to α,

then there exists a valuation α_0 of the $\alpha_p, p \in M$ such that the same too items are true for α_0, v_0 and $\llbracket \mathcal{N} \rrbracket_{v_0}$.

The initial symbolic state $(M_0, \bigwedge_{p \in M_0} \alpha_p = 0)$ satisfies this property because $\mathit{MaxSequences}(\llbracket \mathcal{N} \rrbracket_v) \subseteq \mathit{MaxSequences}(\llbracket \mathcal{N} \rrbracket_{v_0})$. Now, take an abstract processes E considered by the method from a symbolic state (M, K_M). If E is feasible for by $\llbracket \mathcal{N} \rrbracket_v$ starting from marking M with the tokens aged according to a valuation α such that K_M is satisfied by the valuation given by α and v, then let $((t_1, \theta_1), \ldots, (t_m, \theta_m))$[3] be a timed word corresponding to this process, i.e. such that E is the abstraction of the process $\Pi\Big(((t_1, \theta_1), \ldots, (t_n, \theta_n))\Big)$. Then the corresponding sequence $(t_1, \ldots, t_n)$ is also feasible by $\llbracket \mathcal{N} \rrbracket_{v_0}$ (with other dates, in general), and so is the abstract process E. Hence the constraint on the parameters generated by $\mathsf{IM}^K\mathsf{PO}_n^{\text{blocks}}$ when considering E, is satisfied by v. Moreover, by construction, the symbolic state reached after E satisfies the property announced above.

It remains to show the second part of the theorem:

$$v \models K \implies \mathit{MaxProcesses}(\llbracket \mathcal{N} \rrbracket_v) \subseteq \mathit{MaxProcesses}(\llbracket \mathcal{N} \rrbracket_{v_0})\,.$$

By construction of K, if $v \models K$ then the processes of size n (or less than n and ending in a deadlock) of $\llbracket \mathcal{N} \rrbracket_v$ are also feasible by $\llbracket \mathcal{N} \rrbracket_{v_0}$ from the same symbolic states. By concatenation of the processes, we obtain $\mathit{MaxProcesses}(\llbracket \mathcal{N} \rrbracket_v) \subseteq \mathit{MaxProcesses}(\llbracket \mathcal{N} \rrbracket_{v_0})$. □

Theorem 4.8. *Let $\mathcal{N}$ be a PTPN and v_0 be a parameter valuation.* $\mathsf{IM}^K\mathsf{PO}_1^{\text{blocks}}(\mathcal{N}, v_0)$ *terminates iff* IM^K *terminates, and they return equivalent constraints.*

Proof. We show that for each valuation v of the parameters satisfying the initial constraint K_0 of the model, v satisfies the constraint K returned by $\mathsf{IM}^K\mathsf{PO}_1^{\text{blocks}}(\mathcal{N}, v_0)$ iff $\mathit{MaxSequences}(\llbracket \mathcal{N} \rrbracket_v) \subseteq \mathit{MaxSequences}(\llbracket \mathcal{N} \rrbracket_{v_0})$. Theorem 4.7 gives one direction, we prove now that the reciprocal also holds for process of size 1. Indeed, if from a symbolic state, the abstract processes of size 1 which are feasible by $\llbracket \mathcal{N} \rrbracket_v$ are also feasible by $\llbracket \mathcal{N} \rrbracket_{v_0}$, then so are the sequences of length 1 (because there is a bijection between the abstract processes of size 1 and the sequences of length 1). □

Theorem 4.9. *Let $\mathcal{N}$ be a PTPN, let v_0 be a parameter valuation and let $n \in \mathbb{N}$. If* $\mathsf{IM}^K\mathsf{PO}_n^{\text{blocks}}(\mathcal{N}, v_0)$ *and* $\mathsf{IM}^K\mathsf{PO}_{2n}^{\text{blocks}}(\mathcal{N}, v_0)$ *terminate and return constraints K_n and K_{2n}, then K_n implies K_{2n}, i.e. K_{2n} is weaker than K_n.*

[3]If E has n events, then $m = n$.

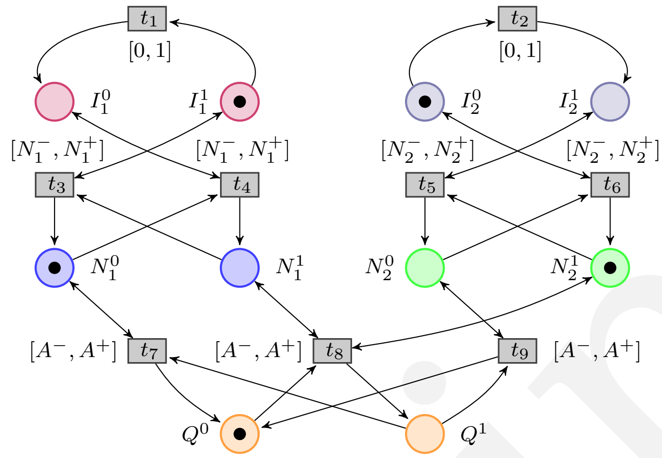

Fig. 5: A PTPN model for the circuit of Fig. 1

Proof. One has to remark that for every symbolic state M, K_M explored by $\mathsf{IM}^K\mathsf{PO}_{2n}^{\text{blocks}}(\mathcal{N}, v_0)$, a symbolic state M, K'_M with K_M weaker than K'_M will all be explored by $\mathsf{IM}^K\mathsf{PO}_n^{\text{blocks}}(\mathcal{N}, v_0)$ The reason is that every sequence $(t_1, \ldots, t_{2n})$ (giving raise to a process with $2n$ events considered by $\mathsf{IM}^K\mathsf{PO}_{2n}^{\text{blocks}}(\mathcal{N}, v_0)$) also gives raise to two processes with n events each (corresponding to sequences $(t_1, \ldots, t_n)$ and $(t_{n+1}, \ldots, t_{2n})$), which will be explored successively by $\mathsf{IM}^K\mathsf{PO}_n^{\text{blocks}}(\mathcal{N}, v_0)$. Hence, both $\mathsf{IM}^K\mathsf{PO}_n^{\text{blocks}}(\mathcal{N}, v_0)$ and $\mathsf{IM}^K\mathsf{PO}_{2n}^{\text{blocks}}(\mathcal{N}, v_0)$ will generate a constraint for this scenario and explore a symbolic state for the final marking, but since $\mathsf{IM}^K\mathsf{PO}_n^{\text{blocks}}(\mathcal{N}, v_0)$ allows only interleavings among $t_1, \ldots, t_n$ and among $t_{n+1}, \ldots, t_{2n}$, the constraints generated by $\mathsf{IM}^K\mathsf{PO}_n^{\text{blocks}}(\mathcal{N}, v_0)$ are stronger than those generated by $\mathsf{IM}^K\mathsf{PO}_{2n}^{\text{blocks}}(\mathcal{N}, v_0)$. □

Finally, we note that selecting a value for n involves deciding a trade-off between termination and quality of the constraints. For small values of n, this method synchronizes concurrent events more frequently, resulting in a generated constraint closer to that synthesized by IM^K. In the extreme case of $n = 1$, this method and IM^K return the equivalent constraints. For large values of n, the step (3) of our algorithm adds symbolic states less frequently to the set $\mathcal{S}$, thus reducing the chances of reaching a fixpoint, and potentially pushing the approach to perform a larger number of iterations.

5. APPLICATION TO ASYNCHRONOUS CIRCUITS

5.1. Improving Latencies in Asynchronous Circuit Design

In this section we apply $\mathsf{IM}^K\mathsf{PO}$ to the asynchronous circuit of Example 1.1 and shown in Fig. 1. Asynchronous circuits are an important application of parameter synthesis techniques: whereas engineers may be able to find one correct valuation of the gate traversal and environment delays using empirical methods, changing these values usually requires the design to restart from zero. Generalizing one correct valuation using synthesis techniques helps designers to find dense sets of parameter valuations preserving the system behavior [Chevallier et al. 2009].

The PTPN $\mathcal{N}$ modeling the circuit in Fig. 1 is shown in Fig. 5. Every signal (e.g. I_2) is encoded by two places representing a low (I_2^0) and high (I_2^1) state of the signal. Every gate (e.g. N_1) is encoded by a number of transitions simulating the raising (t_4)

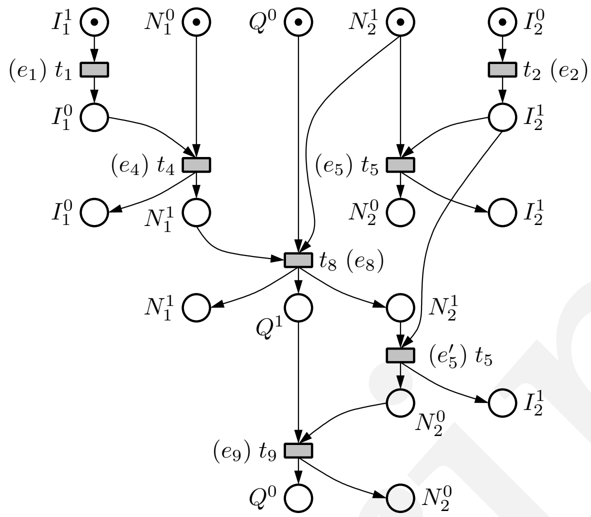

Fig. 6: Untimed unfolding of Fig. 5

and falling (t_3) edges that the gate triggers in its output (N_1^0, N_1^1). The output signal of each gate takes the name of the gate itself. All transitions encoding one gate (e.g. N_1) have the same time interval (e.g. $[N_1^-, N_1^+]$, where N_1^- and N_1^+ are parameters representing the lower and upper propagation delay of the gate).

Transitions t_1 and t_2 simulate the environment. They excite the circuit from its initial state $\langle I_1, I_2, N_1, N_2, Q\rangle \stackrel{\text{def}}{=} \langle 1, 0, 0, 1, 0\rangle$ with a falling edge of signal I_1 and a rising edge of signal I_2 at any moment between 0 and 1 time unit.

We consider a reference parameter valuation v_0 assigning propagation delays to the gates in such a way that *signal Q never rises under the environment attached to $\mathcal{N}$*:

$$N_1^- = 6 \qquad N_1^+ = 7 \qquad N_2^- = A^- = 1 \qquad N_2^+ = A^+ = 2$$

Under v_0, the propagation delay of N_1 is so slow that N_2 always falls before N_1 rises. Specifically, t_4 always fires in the absolute time interval $[6, 8]$, while t_5 is forced to do it in $[1, 3]$. As a result, t_8 (the only transition that raises signal Q) is not firable in $[\![\mathcal{N}]\!]_{v_0}$. Indeed, initially t_8 is disabled. In order to enable it, we need to put a token in N_1^1 before the token in N_2^1 is consumed, which can only be done by firing t_4 before t_5. So, although there is no structural synchronization between the *Not* gates, $\mathcal{N}$ behaves under v_0 as if such synchronization existed. As a result, the original IM^K produces a constraint disallowing to fire t_5 before t_4. We will see that this is not the case for the constraint produced by $\mathsf{IM}^K\mathsf{PO}$.

Let us now apply $\mathsf{IM}^K\mathsf{PO}$ to $\mathcal{N}$ and v_0. First, $\mathsf{IM}^K\mathsf{PO}$ initializes K to $\bigwedge_{g \in \{N_1, N_2, A\}} g^- \leq g^+$. Next, it enumerates the maximal processes of the untimed Petri net underlying $\mathcal{N}$. There are two maximal untimed processes (see Fig. 6):

$$E_1 \stackrel{\text{def}}{=} \{e_1, e_2, e_4, e_5\} \quad \text{and} \quad E_2 \stackrel{\text{def}}{=} \{e_1, e_2, e_4, e_8, e_5', e_9\}.$$

For each of them, our method $\mathsf{IM}^K\mathsf{PO}$ generates an associated $K^{\theta\lambda}$-constraint, composed of three sub-constraints asking that (1) firing dates are met, (2) events enabled and later disabled by the process did not overtake their latest firing delay, and (3) events enabled at the end of the process have enough time to fire. Observe that E_1 and E_2 are maximal processes, so there is no event enabled at the end and (3) simplifies to true. For every event $e_i \in E_1 \cup E_2$, with $i \in \{1, \dots, 9\}$, we denote by $\theta_i \stackrel{\text{def}}{=} \theta(e_i)$ the rational variable associated to e_i.

Consider process E_1. The *firing constraints* due to (1) for e_1, e_2, and e_4 are:

$$0 \leq \theta_1 \leq 1 \qquad 0 \leq \theta_2 \leq 1 \qquad N_1^- \leq \theta_4 - \theta_1 \leq N_1^+ \tag{4}$$

For event e_5, the firing constraint is

$$N_2^- \leq \theta_5 - \theta_2 \leq N_2^+ \tag{5}$$

The *disabling constraints* due to (2) apply to a single event, e_8, enabled after firing e_4 and disabled by e_5. So we have $doe(e_8) = \theta_4$ and $dod(e_8) = \theta_5$. Then constraint (2) for E_1 becomes

$$\theta_5 \leq \theta_4 + A^+ \tag{6}$$

Putting all together, the constraint $K_{E_1}^{\theta\lambda}$ associated to E_1 is the conjunction of (4), (5), and (6).

Analogously, for process E_2, the firing constraints for e_1, e_2, e_4 are the same as for E_1. For e_8, e_5' and e_9 we get

$$\begin{array}{ll} A^- \leq \theta_8 - \theta_4 \leq A^+ & N_2^- \leq \theta_5' - \max\{\theta_8, \theta_2\} \leq N_2^+ \\ A^- \leq \theta_9 - \theta_5' \leq A^+ & \end{array} \tag{7}$$

As for the disabling constraint, we only need to consider e_5, with $doe(e_5) = \theta_2$ and $dod(e_5) = \theta_8$, so we get

$$\theta_8 \leq \theta_2 + N_2^+ \tag{8}$$

and the final constraint $K_{E_2}^{\theta\lambda}$ associated with E_2 is the conjunction of (4), (7), and (8).

After building $K_{E_1}^{\theta\lambda}$ and $K_{E_2}^{\theta\lambda}$, IM^{K}PO eliminates all variables θ_i, resulting in constraints $K_{E_1}^{\lambda}$ and $K_{E_2}^{\lambda}$ over Λ, and checks which of them are satisfied by v_0.

We have $v_0 \models K_{E_1}^{\lambda}$, as clearly $((t_1, 0), (t_2, 0), (t_5, 1), (t_4, 6))$ is a timed word of $[\![\mathcal{N}]\!]_{v_0}$ and E_1 is the set of events of the corresponding process. As for $K_{E_2}^{\lambda}$, observe that it fires e_8, labeled by t_8. We argued earlier that t_8 is not firable in $[\![\mathcal{N}]\!]_{v_0}$. So $v_0 \not\models K_{E_2}^{\lambda}$, and IM^{K}PO adds the negation of $K_{E_2}^{\lambda}$ to K. In the end, IM^{K}PO returns the constraint

$$K \stackrel{\text{def}}{=} \left(\bigwedge_{g \in \{N_1, N_2, A\}} g^- \leq g^+ \right) \wedge \left(N_1^- + A^- > N_2^+ + 1 \right).$$

First remark that v_0 is indeed a model of K. The first part was expected. The inequality $N_1^- + A^- > N_2^+ + 1$ indicates how to generalize the parameters around v_0 while ensuring that t_8 never fires. Indeed, its earliest possible firing time, $N_1^- + A^-$, is required to be larger than the latest possible time when the latest firing delay of t_5 expires: $N_2^+ + 1$.

Now, observe that this constraint allows for a sequential execution where t_4 fires before t_5, which the original IM^K would have forbidden. Indeed any valuation setting the lower and upper propagation delays for N_1, and N_2 to 0 and A^- to a high enough value, would be a model of K, allowing to fire t_1, t_2, t_4, t_5 at time 0, but preventing the firing of t_8.

5.2. Application to a Circuit with a Loop

Let us now consider a variant of this circuit, given in Fig. 7. Initially, we have $\langle I, N_1, N_2, N_3, Q \rangle \stackrel{\text{def}}{=} \langle 1, 1, 0, 0, 1 \rangle$. Observe that this is an unstable configuration for two reasons: the output of the *And* gate is 1, although its two inputs are 0; and both the input and the output of N_1 are 1. The input signal I will eventually fall within a parametric delay $[I^-, I^+]$ (in contrast to the circuit in Fig. 1 where the interval was constant).

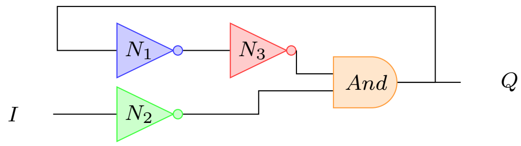

Fig. 7: An asynchronous circuit (looping variant)

Again, all gates have a bounded traversal delay (e.g. $[N_1^-, N_1^+]$ for N_1, and similarly for the other gates). Now, depending on the falling delay of I and on the traversal delays of the gates, two situations may occur: either the system will eventually reach a stable configuration, or signals will rise and fall forever in a cyclic manner through gates N_1, N_3, and And.

Consider the following reference parameter valuation v_0:

$$\begin{array}{lllll} N_1^- = 8 & N_1^+ = 10 & N_2^- = 4 & N_2^+ = 5 & I^- = 1 \\ N_3^- = 2 & N_3^+ = 8 & A^- = 3 & A^+ = 4 & I^+ = 2 \end{array}$$

For v_0, the only possible (maximal) sequence of signals is $I^{\searrow}$, $Q^{\searrow}$, $N_2^{\nearrow}$. Since Q falls before N_1 rises, there is no risk of having both N_2 and N_3 equal to 1, and hence Q will not rise again, preventing an infinite loop.

The PTPN $\mathcal{N}$ of this variant is not given for sake of conciseness; it is similar to the one in Fig. 3 with additional places to model N_3, and specific arcs to model the loop outgoing from the And gate towards the input of N_1.

Applying IM^K to $\mathcal{N}$ and v_0 gives the following result:

$$A^- > I^+ \quad \wedge \quad I^- + N_2^- > A^+ \quad \wedge \quad N_1^- > A^+$$

As expected, this constraint requires the same sequence as for $[\![\mathcal{N}]\!]_{v_0}$, i.e. $I^{\searrow}$, $Q^{\searrow}$, $N_2^{\nearrow}$. Intuitively, the first inequality requires I to fall before Q falls; the second inequality requires that Q falls before N_2 rises; the third one prevents N_1 from falling before Q falls, which hence prevents N_1 from falling at all, since by then N_1 becomes stable.

The application of $\mathsf{IM}^K\mathsf{PO}$ to $\mathcal{N}$ does not terminate: indeed, $\mathsf{IM}^K\mathsf{PO}$ requires to compute all maximal (parametric) processes, and at least one of these is infinite (the one that encodes the infinite loop through the N_1, N_2, and And gates).

However, $\mathsf{IM}^K\mathsf{PO}'$ does terminate and outputs the constraint:

$$I^- + N_2^- > A^+ \quad \wedge \quad N_1^- > A^+$$

In contrast to IM^K, $\mathsf{IM}^K\mathsf{PO}'$ does not impose any order between $I^{\searrow}$ and $Q^{\searrow}$, hence the constraint $A^- > I^+$ does not appear. The constraint $I^- + N_2^- > A^+$ (constraining the order between $Q^{\searrow}$ and $N_2^{\nearrow}$) is preserved because the corresponding transitions in the model share the input place N_2^0, which forces to sequentialize them. The second constraint $N_1^- > A^+$ again prevents the rise of N_1 (as in IM^K).

6. EXPERIMENTAL EVALUATION

As a proof of concept, we implemented both IM^K and $\mathsf{IM}^K\mathsf{PO}$ in a tool non-surprisingly baptized `impo`.[4] Our tool relies on the CUNF Petri net unfolder [Rodríguez and Schwoon 2013] to build the (untimed) unfolding of the input net, and PolyOp[5] to handle polyhedra operations on the generated constraints. PolyOp itself is a wrapper around the Parma Polyhedra Library (PPL) [Bagnara et al. 2008]. The use of external tools and libraries rather than an ad-hoc implementation of them entails a small performance

[4]Tool and experiments are publicly available from https://lipn.fr/~rodriguez/exp/tecs16/.
[5]Available at https://github.com/etienneandre/PolyOp.

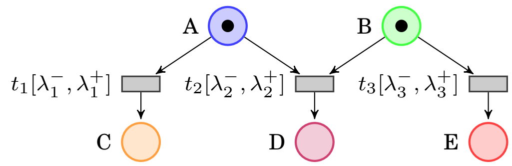

Fig. 8: A simple net when $\mathsf{IM}^K\mathsf{PO}$ outperforms IM^K

penalty. We find this justified for the goals of this section. All aforementioned programs and libraries are released under open-source licenses.

Our implementation first enumerates all maximal (untimed) configurations, by means of an ad-hoc concretization of the Optimal Dynamic Partial-Order Reduction (ODPOR) algorithm presented in [Rodríguez et al. 2015]. To the best of our knowledge, this is the first implementation of this algorithm for Petri nets.

In the sequel we investigate the practicality of $\mathsf{IM}^K\mathsf{PO}$ over various examples.

Case Study 1: Fig. 5. We first deal with the circuit in Fig. 5. As expected, the constraint computed by impo is the one given in Section 5.1, i.e.:

$$K \stackrel{\text{def}}{=} \left(\bigwedge_{g\in\{N_1,N_2,A\}} g^- \leq g^+\right) \wedge \left(N_1^- + A^- > N_2^+ + 1\right).$$

Case Study 2: Fig. 8. In this simple net (originally taken from [André et al. 2013, Fig.3b]), the reference parameter valuation v_0 is $\{\lambda_1^- \to 5, \lambda_1^+ \to 6, \lambda_2^- \to 1, \lambda_2^+ \to 3, \lambda_3^- \to 2, \lambda_3^+ \to 4\}$. For this valuation, t_1 cannot fire first as its firing interval is greater than that of the other transitions. Hence, IM^K outputs the following constraint:

$$K \stackrel{\text{def}}{=} \left(\bigwedge_{i\in\{1,2,3\}} \lambda_i^- \leq \lambda_i^+\right) \wedge \left(\lambda_1^- > \lambda_2^+ \vee (\lambda_2^+ \geq \lambda_1^- > \lambda_3^+)\right).$$

This constraint ensures that either t_3 or t_2 fires before t_1, but t_1 cannot fire first.

However, $\mathsf{IM}^K\mathsf{PO}$ only outputs $\bigwedge_{i\in\{1,2,3\}} \lambda_i^- \leq \lambda_i^+$ since only the partial orders need to be preserved, hence allowing t_1 to fire first. Without surprise, the constraint output by IM^K is strictly included in that output by $\mathsf{IM}^K\mathsf{PO}$.

Case Study 3. We consider Fischer's mutual exclusion protocol (we use a version adapted from that encoded in timed CSP in PAT's benchmarks library [Sun et al. 2009]). Before trying to enter the critical section, a process checks the state of a global variable containing a process id; if the variable is unset, it waits at most δ time units, and then sets the variable to its own id. After waiting again at least ϵ time units, it checks that the variable is still set to its own id and, if so, finally enters the critical session. If any of the variable checks fails, the process starts again from the beginning.

We synthesized constraints for variants with two processes. Both IM^K and $\mathsf{IM}^K\mathsf{PO}$ synthesize the well-known constraint ensuring mutual exclusion, i.e. $\epsilon > \delta$. However, $\mathsf{IM}^K\mathsf{PO}$ enumerates 30 maximal configurations whereas IM^K requires examining the 78 linearizations of those 30 configurations.

Lessons learned. For each maximal configuration, impo generates the constraint $K^{\theta\lambda}$ to subsequently perform existential quantification of non-parametric variables, yielding the constraint K^{λ}. Although we foresee important algorithmic improvements for this task[6], in our experiments this seems to be one of the most expensive steps in

[6]For instance, existential quantification for multiple maximal configurations can be factorized by existentially quantifying first their common parts and the reusing partial results.

the $\mathsf{IM}^K\mathsf{PO}$ loop (together with the final conjunction of negated v_0-incompatible constraints).

An advantage of $\mathsf{IM}^K\mathsf{PO}$ over IM^K is the fact that $\mathsf{IM}^K\mathsf{PO}$ examines up to exponentially less executions than IM^K. We experimentally observed that often this comes at the cost of generating harder constraints: while $K^{\theta\lambda}$ is convex for a sequential execution (as in IM^K), it is in general not for a partial order ($\mathsf{IM}^K\mathsf{PO}$), due to the $\min$ and $\max$ expressions. This observation suggests interest in future work aimed at exploring a reduced fragment of the *sequential* execution tree of the system (for instance, by combining ODPOR [Rodríguez et al. 2015] with the results in this paper).

7. FINAL REMARKS

In this paper, we proposed a parametric analysis of concurrent timed systems based on a partial-order semantics. Our approach looks for parameters that preserve only the partial-order semantics of the system. Hence, the constraint output by our method enhances (i.e. weakens) the constraint output when looking for other parameter valuations with a similar set of sequences. We showed the interest of our approach on acyclic, or restricted cyclic asynchronous circuits.

The constraints manipulated and output by $\mathsf{IM}^K\mathsf{PO}$ (and its variants $\mathsf{IM}^K\mathsf{PO}'$ and $\mathsf{IM}^K\mathsf{PO}_n^{\text{blocks}}$) do not fall in general in the nice class of convex constraints. We have to deal in general with non-convex constraints, which can be represented as disjunctions of convex constraints or as unions of polyhedra. We argue that this is not a serious limitation of the method: First, the method may generate few disjunctions in practice. For the examples in Section 5, the disjunctions appear under the form of $\max$ and $\min$ in the inequalities of the $K_E^{\theta\lambda}$, but then completely disappear in the final result of $\mathsf{IM}^K\mathsf{PO}$ and $\mathsf{IM}^K\mathsf{PO}'$. Second, $\mathsf{IM}^K\mathsf{PO}$ outputs the weakest constraint that guarantees preservation of the partial-order semantics. This constraint is in general non-convex. But it is also possible to output only a convex constraint (or a union of few convex constraints) which is not the most permissive but is satisfied by v_0 and guarantees preservation of partial-order semantics. This is actually what the current implementation IMITATOR [André et al. 2012] (as of version 2.8) of IM^K does for the preservation of the sequential semantics. An alternative is to handle non-convex polyhedra using a dedicated library: several verification tools for timed systems, such as IMITATOR (for other algorithms than IM^K), ROMÉO [Lime et al. 2009] and UPPAAL [Larsen et al. 1997] and its extensions, deal with non-convex constraints; they use efficient representations and achieve very satisfactory performances in practice. This is also the case of our implementation `impo`. The Parma polyhedra library [Bagnara et al. 2008] (used in ROMÉO, IMITATOR and `impo`) offers such a representation.

The main focus in this paper was on acyclic models, for which we have a nice characterization of the result (Theorem 4.3); nevertheless we proposed two pragmatic approaches $\mathsf{IM}^K\mathsf{PO}'$ and $\mathsf{IM}^K\mathsf{PO}^{\text{blocks}}$ to handle cyclic models.

We prototyped a first version of our implementation. We aim at implementing variants of $\mathsf{IM}^K\mathsf{PO}$ (i.e. $\mathsf{IM}^K\mathsf{PO}'$ and $\mathsf{IM}^K\mathsf{PO}^{\text{blocks}}$). An improved implementation will especially be useful to experimentally compare the respective efficiency of $\mathsf{IM}^K\mathsf{PO}$ and its variants; whereas the latter have better termination, they may explore more states, since they explore all prefixes of maximal processes.

Among the future works, proving the undecidability of the underlying decision problem (given a valuation v, does there exist a valuation $v' \neq v$ for which the processes of v' are the same as that of v?) would be of interest. On the one hand, we believe this problem shall be undecidable: a very close problem (given a valuation v, does there exist a valuation $v' \neq v$ for which the untimed language (resp. the untimed traces) of v'

are the same as that of v?) was shown to be undecidable in the setting of PTAs [André and Markey 2015]. The proof relies on the encoding of a 2-counter machine (the halting of which is undecidable [Minsky 1967]); this encoding is completely sequential, and therefore words and sequences are equivalent in this setting. On the other hand, translating this result to PTPNs is not trivial: the translation from PTAs to PTPNs proposed in [Bérard et al. 2005] relies on languages with accepting locations (or markings), and therefore might add extra places, making the behavior not sequential anymore in PTPNs. We believe an interesting direction is to build an ad-hoc sequential encoding of a 2-counter machine in PTPNs (which, to the best of our knowledge, was never done), which could then be used to prove the undecidability of this problem.

ACKNOWLEDGMENTS

The authors would like to thank anonymous reviewers of both the ACSD version and of this manuscript for useful comments.

References

Rajeev Alur and David L. Dill. 1994. A Theory of Timed Automata. *Theoretical Computer Science* 126, 2 (1994), 183–235.

Rajeev Alur, Thomas A. Henzinger, and Moshe Y. Vardi. 1993. Parametric real-time reasoning. In *STOC*. ACM, 592–601.

Étienne André, Thomas Chatain, Emmanuelle Encrenaz, and Laurent Fribourg. 2009. An Inverse Method for Parametric Timed Automata. *IJFCS* 20, 5 (2009), 819–836.

Étienne André, Laurent Fribourg, Ulrich Kühne, and Romain Soulat. 2012. IMITATOR 2.5: A Tool for Analyzing Robustness in Scheduling Problems. In *FM (LNCS)*, Vol. 7436. Springer, 33–36.

Étienne André and Nicolas Markey. 2015. Language Preservation Problems in Parametric Timed Automata. In *FORMATS (LNCS)*, Vol. 9268. Springer, 27–43.

Étienne André, Laure Petrucci, and Giuseppe Pellegrino. 2013. Precise Robustness Analysis of Time Petri Nets with Inhibitor Arcs. In *FORMATS (LNCS)*, Vol. 8053. Springer, 1–15.

Étienne André and Romain Soulat. 2011. Synthesis of Timing Parameters Satisfying Safety Properties. In *RP (LNCS)*, Vol. 6945. Springer, 31–44.

Tuomas Aura and Johan Lilius. 2000. A causal semantics for time Petri nets. *Theoretical Computer Science* 243, 1-2 (2000), 409–447.

Roberto Bagnara, Patricia M. Hill, and Enea Zaffanella. 2008. The Parma Polyhedra Library: Toward a Complete Set of Numerical Abstractions for the Analysis and Verification of Hardware and Software Systems. *Science of Computer Programming* 72, 1–2 (2008), 3–21.

Johan Bengtsson, Bengt Jonsson, Johan Lilius, and Wang Yi. 1998. Partial Order Reductions for Timed Systems. In *CONCUR (LNCS)*, Vol. 1466. Springer, 485–500.

Béatrice Bérard, Franck Cassez, Serge Haddad, Didier Lime, and Olivier H. Roux. 2005. Comparison of the Expressiveness of Timed Automata and Time Petri Nets. In *FORMATS (LNCS)*, Vol. 3829. Springer, 211–225.

Patricia Bouyer, Serge Haddad, and Pierre-Alain Reynier. 2006. Timed Unfoldings for Networks of Timed Automata. In *ATVA (LNCS)*, Vol. 4218. Springer, 292–306.

Laura Bozzelli and Salvatore La Torre. 2009. Decision problems for lower/upper bound parametric timed automata. *Formal Methods in System Design* 35, 2 (2009), 121–151.

Véronique Bruyère and Jean-François Raskin. 2007. Real-Time Model-Checking: Parameters everywhere. *Logical Methods in Computer Science* 3, 1:7 (2007).

Franck Cassez, Thomas Chatain, and Claude Jard. 2006. Symbolic Unfoldings For Networks of Timed Automata. In *ATVA (LNCS)*, Vol. 4218. Springer, 307–321.

Thomas Chatain and Claude Jard. 2006. Complete Finite Prefixes of Symbolic Unfoldings of Safe Time Petri Nets. In *ICATPN (LNCS)*, Vol. 4024. 125–145.

Rémy Chevallier, Emmanuelle Encrenaz-Tiphène, Laurent Fribourg, and Weiwen Xu. 2009. Timed Verification of the Generic Architecture of a Memory Circuit Using Parametric Timed Automata. *Formal Methods in System Design* 34, 1 (2009), 59–81.

Robert Clarisó and Jordi Cortadella. 2005. Verification of Concurrent Systems with Parametric Delays Using Octahedra. In *ACSD*. IEEE Computer Society, 122–131.

Joost Engelfriet. 1991. Branching Processes of Petri Nets. *Acta Informatica* 28, 6 (1991), 575–591.

Thomas Hune, Judi Romijn, Mariëlle Stoelinga, and Frits W. Vaandrager. 2002. Linear parametric model checking of timed automata. *Journal of Logic and Algebraic Programming* 52-53 (2002), 183–220.

Aleksandra Jovanović, Didier Lime, and Olivier H. Roux. 2015. Integer Parameter Synthesis for Timed Automata. *IEEE Transactions on Software Engineering* 41, 5 (2015), 445–461.

Victor Khomenko. 2012. Punf. (2012). http://homepages.cs.ncl.ac.uk/victor.khomenko/tools/punf/.

Michał Knapik and Wojciech Penczek. 2012. Bounded Model Checking for Parametric Timed Automata. *Transactions on Petri Nets and Other Models of Concurrency* 5 (2012), 141–159.

Kim G. Larsen, Paul Pettersson, and Wang Yi. 1997. UPPAAL in a Nutshell. *International Journal on Software Tools for Technology Transfer* 1, 1-2 (1997), 134–152.

Didier Lime, Olivier H. Roux, Charlotte Seidner, and Louis-Marie Traonouez. 2009. Romeo: A Parametric Model-Checker for Petri Nets with Stopwatches. In *TACAS (LNCS)*, Vol. 5505. Springer, 54–57.

Denis Lugiez, Peter Niebert, and Sarah Zennou. 2005. A partial order semantics approach to the clock explosion problem of timed automata. *Theoretical Computer Science* 345, 1 (2005), 27–59.

Nicolas Markey. 2011. Robustness in Real-time Systems. In *SIES*. IEEE Computer Society Press, 28–34.

Eric Mercer, Chris J. Myers, and Tomohiro Yoneda. 2002. Modular Synthesis of Timed Circuits using Partial Order Reduction. *Electr. Notes Theor. Comput. Sci.* 65, 6 (2002), 180–201.

Philip M. Merlin and David J. Farber. 1976. Recoverability of Communication Protocols – Implications of a Theoretical Study. *IEEE Transactions on Communications* 24 (1976).

Marius Minea. 1999. Partial Order Reduction for Model Checking of Timed Automata. In *CONCUR (LNCS)*, Vol. 1664. Springer, 431–446.

Marvin L. Minsky. 1967. *Computation: finite and infinite machines*. Prentice-Hall, Inc., NJ, USA.

Peter Niebert and Hongyang Qu. 2006. Adding Invariants to Event Zone Automata. In *FORMATS (LNCS)*, Vol. 4202. Springer, 290–305.

Wojciech Penczek and Agata Pólrola. 2001. Abstractions and Partial Order Reductions for Checking Branching Properties of Time Petri Nets. In *ICATPN (LNCS)*, Vol. 2075. 323–342.

Florent Peres, Bernard Berthomieu, and François Vernadat. 2011. On the composition of time Petri nets. *Discrete Event Dynamic Systems* 21, 3 (2011), 395–424.

César Rodríguez, Marcelo Sousa, Subodh Sharma, and Daniel Kroening. 2015. Unfolding-based Partial Order Reduction. In *CONCUR*. 456–469.

César Rodríguez and Stefan Schwoon. 2013. Cunf: A Tool for Unfolding and Verifying Petri Nets with Read Arcs. In *ATVA (LNCS)*, Vol. 8172. Springer, 492–495.

Alexander Schrijver. 1986. *Theory of linear and integer programming*. John Wiley & Sons, Inc., NY, USA.

Stefan Schwoon. 2014. Mole. (2014). http://lsv.ens-cachan.fr/~schwoon/tools/mole/.

Jun Sun, Yang Liu, Jin Song Dong, and Jun Pang. 2009. PAT: Towards Flexible Verification under Fairness. In *CAV (LNCS)*, Vol. 5643. Springer, 709–714.

Louis-Marie Traonouez, Bartosz Grabiec, Claude Jard, Didier Lime, and Olivier H. Roux. 2010. Symbolic Unfolding of Parametric Stopwatch Petri Nets. In *ATVA (LNCS)*, Vol. 6252. Springer, 291–305.

Louis-Marie Traonouez, Didier Lime, and Olivier H. Roux. 2009. Parametric Model-Checking of Stopwatch Petri Nets. *Journal of Universal Computer Science* 15, 17 (2009), 3273–3304.

Irina Virbitskaite and E. Pokozy. 1999. A Partial Order Method for the Verification of Time Petri Nets. In *FCT (LNCS)*, Vol. 1684. Springer, 547–558.

Tomohiro Yoneda and Bernd-Holger Schlingloff. 1997. Efficient Verification of Parallel Real-Time Systems. *Formal Methods in System Design* 11, 2 (1997), 187–215.

Hao Zheng, Eric Mercer, and Chris J. Myers. 2001. Automatic Abstraction for Verification of Timed Circuits and Systems. In *CAV (LNCS)*, Vol. 2102. Springer, 182–193.

Received March 2016; revised September 2016; accepted October 2016