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

***Abstract*—Parameter synthesis for timed systems aims at deriving parameter valuations satisfying a given property. In this paper we target concurrent systems; it is well known that concurrency is a source of state-space explosion, and partial order techniques were defined to cope with this problem. Here we use partial order semantics for parametric time Petri nets as a way to significantly enhance the result of an existing synthesis algorithm. Given a reference parameter valuation, our approach synthesizes other valuations preserving, up to interleaving, the behavior of the reference parameter valuation. We show the applicability of our approach using acyclic asynchronous circuits.**

I. 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., expressed in some temporal logics). Popular formalisms to model and verify parametric concurrent timed systems include parametric timed automata (PTA) [AHV93] and parametric time Petri nets (PTPNs) [TLR09]. Parameter synthesis for PTA or PTPNs was tackled with respect to safety or unavoidability of some states [AHV93], [AS11], [JLR15], or the satisfiability of temporal logic formulas (e.g., [BR07]). All these problems are undecidable in general, with only one non-trivial exception: the emptiness of the set of parameter valuations for which a state is reachable, or for which there exists an infinite accepting run, is decidable for a subclass of PTA called L/U-PTA [BL09]. The same holds for L/U-PTPNs [TLR09]. Applications include the verification of asynchronous circuits with parametric propagation delays using octahedra [CC05] and PTA [CEFX09].

In [ACEF09], [APP13], we proposed the inverse method IM: given a PTPN and a reference parameter valuation v_0, IM synthesizes other parameter 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. Among several 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., [CEFX09], [AS11]).

This work is partially supported by the ANR national research program ANR-14-CE28-0002 PACS ("Parametric Analyses of Concurrent Systems"). This is the author version of the paper of the same name accepted for publication at ACSD 2015. The final publication is available at http://www.ieee.org.

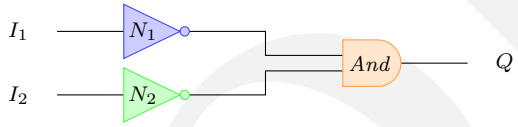

Figure 1: An asynchronous circuit

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 verification. A way to alleviate this problem is to consider *partial order semantics*, as opposed to a sequential view to the executions of the system. But in the context of real-time distributed systems, one cannot rely on a smooth concurrency theory like in the untimed, asynchronous case. In particular, the nice independency relations used in Mazurkiewicz traces do not work: actions performed on distinct machines may not commute freely because they are ordered by their occurrence time. This explains that little literature exists on partial-order reduction techniques for time Petri nets [PP01], [VP99], [YS97] and for networks of timed automata [BJLY98], [Min99], [LNZ05], [NQ06]. Concerning partial order semantics and unfoldings, we can cite [AL00], [CJ06], [TGJ+10] for time Petri nets and [CCJ06], [BHR06] for networks of timed automata.

Here we use partial order semantics to enhance the quality of the output of the inverse method IM, so that the algorithm outputs a larger set of parameter valuations. Consider the asynchronous circuit of Fig. 1, where 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 a precise scenario where signal I_1 falls and signal I_2 rises, which causes N_1 to rise ($N_1^{\nearrow}$) and N_2 to fall ($N_2^{\searrow}$). Depending on the timing delays of the circuit, Q may rise. Assume now a reference parameter valuation v_0 for which Q never rises, while forcing the following ordering: $N_1^{\nearrow}$ then $N_2^{\searrow}$. IM will output a constraint on the parameters which preserves the *sequential* behavior of the circuit, i.e. $N_1^{\nearrow}$ then $N_2^{\searrow}$. But, since gates N_1 and N_2 operate concurrently, the ordering of their propagation delays is irrelevant for the overall behavior of the circuit, seen as a partial order execution. In other words, IM preserves here the *temporal ordering* fixed by v_0, while preserving the *partial order behavior* would have been enough to prevent Q from rising.

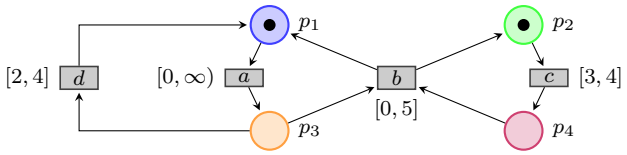

Figure 2: A safe time Petri net.

Contribution: In this paper, we propose an approach called IMPO 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. We show that IMPO significantly enhances the result of IM, 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 so far only deals with acyclic systems (or, in a variant of our approach, with a limited dose of cyclicity): 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., [CEFX09]).

Outline: In Section II, we define time Petri nets and their parametric extension; we then recall the inverse method in Section III. In Section IV, we introduce our partial order semantics for TPN. Then, we present our parameter synthesis method IMPO for preserving partial order runs in Section V; we also present a variant IMPO' of the method that addresses limited cyclic systems. We illustrate our method in Section VI by applying it to a scenario of the asynchronous circuit of Fig. 1, and we apply IMPO' to a circuit with a loop. We conclude in Section VII.

II. Parametric Time Petri Nets

In this section, we first define (non-parametric) time Petri nets and their semantics (Section II-A); then we give notations for parametric models (Section II-B).

A. 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 1 (Time Petri Net (TPN) [MF76])**.** *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 $lfd(t)$ *with each transition* t*;* $M_0 \subseteq P$ *is the* initial marking.

A time Petri net is represented as a graph with two types of nodes: places (circles) and transitions (rectangles). The interval $[efd(t), lfd(t)]$ is written near each 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 $En(M)$. 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, 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:

$$\forall t \in En(M) \quad \theta' \leq doe(t) + lfd(t) \ .$$

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 sequence $((t_1, \theta_1), \ldots, (t_n, \theta_n))$ 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 $((t_1, \theta_1), \ldots, (t_n, \theta_n))$, its associated *sequence* is the time-abstract word $(t_1, \ldots, t_n)$. Given a TPN N, we denote by $Sequences(N)$ the set of *maximal*[1] sequences associated with all timed words of N. In the following, we will refer to $Sequences(N)$ as the *set of sequences* of N.

B. Parametric Time Petri Nets

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 $v : \Lambda \to \mathbb{R}_+$ assigning with each parameter a value in $\mathbb{R}_+$. For technical convenience, we extend the function v to linear terms over $\mathbb{Q}_+ \cup \Lambda \cup \{\infty\}$.

A constraint over Λ is a Boolean combination (disjunctions and conjunctions) of linear inequalities over $\Lambda \cup \mathbb{Q}_+$ with integer coefficients.

A parameter valuation v *satisfies* a constraint K over the parameters, 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.

Parametric time Petri nets (PTPNs) are a parametric extension of TPNs, where the temporal bounds of the transitions can either be rational numbers or *parameters* [TLR09], [APP13].[2]

[1] A sequence is maximal if it is not the prefix on any other sequence.

[2] 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.

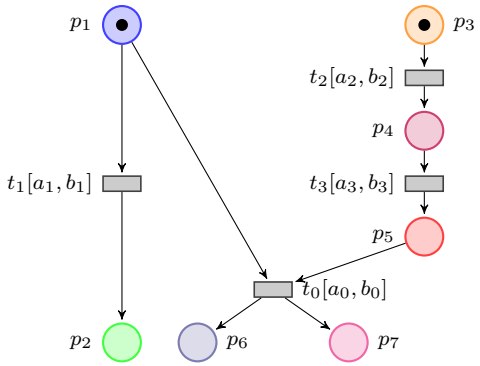

Figure 3: An example of a PTPN

Definition 2 (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 i)* P *and* T *are non-empty, disjoint sets of* places *and* transitions *respectively, ii)* $\Lambda \stackrel{\text{def}}{=} \{\lambda_1, \ldots, \lambda_j\}$ *is a finite set of* parameters*, iii)* 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* $t^\bullet \stackrel{\text{def}}{=} post(t) \subseteq P$*; iv) 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*, v)* $M_0 \subseteq P$ *is the* initial marking*, and vi)* K_0 *is the* initial constraint *over* Λ.

K_0 is a constraint over Λ giving the initial domain of the parameters, and must at least specify that the minimum bounds of the firing intervals are lower than or equal to the maximum bounds. Additional linear constraints 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 3 ($[\![\mathcal{N}]\!]_v$). *Given a PTPN* $\mathcal{N} \stackrel{\text{def}}{=} (P, T, \Lambda, pre, post, pefd, plfd, M_0, K_0)$ *and a valuation* $v : \Lambda \to \mathbb{R}_+$*, 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* $t \in T$.

III. Preserving Time-Abstract Runs Using IM

In [ACEF09], we proposed the *inverse method* IM, that considers a system modeled using a network of PTA, and synthesizes a constraint by taking advantage of a reference parameter valuation. IM was then extended to PTPNs [APP13]: Given a PTPN $\mathcal{N}$ and a reference parameter valuation v_0, IM 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 $[\![\mathcal{N}]\!]_{v_0}$. This result 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 gives a measure of the system robustness (see, e.g., [Mar11]), 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 [APP13].

Theorem 1 ([APP13]). *Let* $\mathcal{N}$ *be a PTPN and* v_0 *be a parameter valuation. Assume* IM *terminates with result* K*. Then i)* $v_0 \models K$*, and ii) for all* $v \models K$*,* $Sequences([\![\mathcal{N}]\!]_v) \subseteq Sequences([\![\mathcal{N}]\!]_{v_0})$.

IM explores a set of symbolic states of the input PTPN. This parametric semantics (not given here for sake of conciseness, but available in [TLR09], [APP13]) considers symbolic states made of a marking and a constraint over parameters and parametric firing times, i.e., variables similar to clocks in PTA [AHV93], with the exception that they decrease with time whereas PTA clocks increase. IM 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 projects this constraint onto Λ (i.e., eliminates the parametric firing times), 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 can be found in [APP13].

Remark 1. *In fact, the method we refer to as* IM *in this paper is not exactly the method presented in [ACEF09], [APP13], but rather a variant called "*IM^K*" introduced in [AS11]. The original* IM *[ACEF09], [APP13] does not return* K *but the intersection of the projection onto* Λ *of the constraints associated with all symbolic states. This latter operation is needed to ensure that i) deadlocks not possible in* $[\![\mathcal{N}]\!]_{v_0}$ *cannot occur for any* $v \models K$*, and ii) the sets of maximal sequences of* $[\![\mathcal{N}]\!]_{v_0}$ *and* $[\![\mathcal{N}]\!]_v$ *are the same. Different from TA, in time Petri nets there is no need to return the intersection of all states to avoid deadlocks: a deadlock for* v *implies that no transition is enabled at the end of the sequence, and then this deadlock exists also for* v_0*. Here, we focus on* IM^K *rather than the original* IM *for two reasons. First, preventing deadlock absent from* $[\![\mathcal{N}]\!]_{v_0}$ *is ensured using* IM^K *only (which is not the case in TA). Second, proving that the constraint returned by our approach (in Section V) is strictly better than the inverse method is easier to show using* IM^K *rather than* IM*. Note that, since the result of* IM^K *is always weaker (i.e., corresponding to a larger set of parameter valuations) than* IM*, showing that the constraint returned by our approach is weaker than the one output by* IM^K *also implies that it outperforms* IM*. In the following, we refer to* IM^K *as* IM.

Example 1. *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 1 time unit, whereas transition* t_2 *can only fire after at least 2 time units. More precisely, the only 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 *to* $\mathcal{N}$ *and* v_0 *gives (besides* $a_i \leq b_i$ *for* $0 \leq i \leq 3$*) the constraint* $b_1 < a_2$*. Intuitively, this requires* t_1 *to fire strictly before* t_2.

IV. Partial Order Semantics

The inverse method 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 forces this ordering and allows only valuations for which the only sequence (t_1, t_2, t_3) is possible, like in $[\![\mathcal{N}]\!]_{v_0}$.

With other parameter valuations (recall that we assume $a_i \leq b_i$ for $i \in \{1, 2, 3\}$, to avoid further deadlocks), 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 Fig. 1, a designer may want to replace a hardware gate with another one 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 V, we will propose an alternative to IM which relaxes the inverse method to output a weaker constraint, i.e., a set of parameter valuations larger than in the original IM. The new method does not guarantee the preservation of the sequential behavior (sequences) but only of the *partial order behavior* of the system.

A. Partial Order Representation of Runs: Processes

Processes are a way to represent an execution of a (time) Petri net so that the actions (called events) are not totally ordered like in timed words: in the context of 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.

Processes of a safe time Petri net will be defined as the image of a mapping Π from its timed words to their partial order representation as processes. These processes are those described in [AL00]. Fig. 4 shows a process of the TPN of Fig. 2, which is the image by Π of the timed word $((a, 3), (c, 3), (b, 5), (a, 9))$.

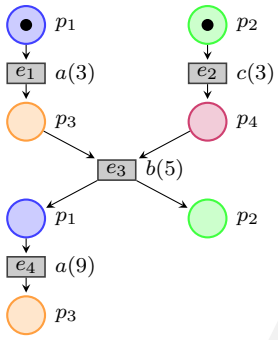

Figure 4: A representation of a process of the TPN of Fig. 2. The dates of the events are in brackets. Technically, the initial condition labeled p_1 is coded as $(\bot, p_1)$, the event e_1 (labeled a) is coded as $(\{(\bot, p_1)\}, a)$, its output condition is coded as (e_1, p_3), event e_2 as $(\{(\bot, p_2)\}, c)$, and e_3 as $(\{(e_1, p_3), (e_2, p_4)\}, b)$.

1) Coding of Events and Conditions: We use a canonical coding like in [Eng91]. This coding is illustrated in Fig. 4. 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 $E_{\mathcal{E}}$ (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 pairs $b \stackrel{\text{def}}{=} ({}^\bullet b, \pi(b))$. Such a pair is called *condition* and refers to the token that has been created by the event ${}^\bullet b$ in the place $\pi(b)$. We say that the event $e \stackrel{\text{def}}{=} ({}^\bullet e, \tau(e))$ *consumes* the conditions in ${}^\bullet e$. Symmetrically the set $\{(e, p) \mid p \in \tau(e)^\bullet\}$ of conditions that are *created* by e is denoted $e^\bullet$. A virtual initial event $\bot$ is used as preset for initial conditions. By convention $\bot^\bullet \stackrel{\text{def}}{=} \{\bot\} \times M_0$ and $\theta(\bot) \stackrel{\text{def}}{=} 0$.

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 4 (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 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 $\mathcal{E}$, $\uparrow(E_{\mathcal{E}})$ represents the set of conditions that remain at the end of the process.

Definition 5. *The function Π which assigns to each timed word $((t_1, \theta_1), \ldots, (t_n, \theta_n))$ of a safe TPN N its partial-order representation, is defined as follows:*

- $\Pi(\epsilon) \stackrel{\text{def}}{=} \emptyset$
- $\Pi\Big(\big((t_1, \theta_1), \ldots, (t_{n+1}, \theta_{n+1})\big)\Big) \stackrel{\text{def}}{=} \mathcal{E} \cup \{(e, \theta_{n+1})\}$,

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

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

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 all event e, we denote $\lceil e \rceil \stackrel{\text{def}}{=} \{f \in D_N \mid f \to^* e\}$, and for all set E of events, $\lceil E \rceil \stackrel{\text{def}}{=} \bigcup_{e\in E} \lceil e \rceil$.

B. Characterization of Processes

Since timed processes are defined as sets of dated events (the events being taken from the set D_N defined above), a natural problem is to decide whether a set of dated events $\mathcal{E} \subseteq D_N \times \mathbb{R}_+$ is a process. The answer is nontrivial and was treated in [AL00]. We give a summary here.

The following lemma concerns only the structural relations between the events in $\mathcal{E}$.

Lemma 1. *Let* N *be a safe extended 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 5), 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 6 (Abstract process, $Processes(N)$)**.** *In the following, a causally closed, conflict free set* E *of events is called an* abstract process*. Notice that the time constraints of the TPN do not play any role here. Therefore we speak of the abstract processes of a PTPN* $\mathcal{N}$ *as well.*

An abstract process E *is* feasible *for an instantiated TPN* N *if there exists a process* $\mathcal{E}$ *of* N *such that* $E = E_{\mathcal{E}}$*. We denote by* $Processes(N)$ *the set of maximal (w.r.t. set inclusion) abstract processes which are feasible for* N.

The following lemma characterizes the possible firing dates for the events of an abstract process under the time constraints of a TPN N.

Lemma 2 (Possible dates for an abstract process)**.** *Let* N *be a safe extended TPN. Let* $\mathcal{E} \subseteq D_N \times \mathbb{R}_+$ *be a 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:* $\forall e \in E$ $efd(\tau(e)) \leq \theta(e) - doe(e) \leq lfd(\tau(e))$ *where* $doe(e) \stackrel{\text{def}}{=} \max_{b\in{}^\bullet e} \theta({}^\bullet b)$ *is the date when the event* e *was enabled;*
- *Denote* $EnabledEvents(E)$ *the set of events* $e \in D_N \setminus E$ *that were eventually enabled during the process (*${}^\bullet e \subseteq \bigcup_{f\in E\cup\{\bot\}} f^\bullet$*) but did not fire because they were disabled by an event* $f \in E$ *such that* ${}^\bullet e \cap {}^\bullet f \neq \emptyset$*. It is required that these events did not overtake their latest firing delay (notice that this concerns events which are* not *in* E*):* $\forall e \in EnabledEvents(E) \quad dod(e) \leq doe(e) + lfd(\tau(e))$ *where* $dod(e) \stackrel{\text{def}}{=} \min\{\theta(f) \mid f \in E \wedge {}^\bullet f \cap {}^\bullet e \neq \emptyset\}$ *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:* $\forall e \in D_N \quad {}^\bullet e \subseteq \uparrow(E) \implies \theta_{end} \leq doe(e) + lfd(\tau(e))$ *where* $\theta_{end} \stackrel{\text{def}}{=} \max_{f\in E\cup\{\bot\}} \theta(f)$ *is the date that is reached at the end of the process.*

The proof can be found in [AL00].

Let $E = \{e_1, \ldots, e_n\} \subseteq D_N$ be a causally closed, conflict free set of events of a TPN N. The conditions in Lemma 2 can be summarized in the following constraint K_E^θ 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 7 (K_E^θ)**.** *We denote* K_E^θ *the constraint on the* $\theta(e)$*,* $e \in E$*, defined as the conjunction of the following:*

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

Notice that the notations $doe(e)$, $dod(e)$ and θ_{end} hide terms of the form $\max\{\ldots\}$ and $\min\{\ldots\}$. Inequalities containing such terms can be expanded to Boolean combinations (disjunctions and conjunctions) of linear inequalities over $\Lambda \cup \mathbb{Q}_+$ with integer coefficients. For instance the inequality $\theta_{\text{end}} \leq doe(e) + lfd\big(\tau(e)\big)$ becomes

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

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

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

Example 2. *Consider the process of Fig. 4 and replace the firing dates by variables* $\theta(e_1)$*,* $\theta(e_2)$*,* $\theta(e_3)$*,* $\theta(e_4)$*. The values of the variables that correspond to processes accepted by the TPN of Fig. 2 are those satisfying the following constraint, given by earliest and latest firing delays of the events (notice that only one event in the process is maximal w.r.t.* $\to$*, hence*

$\theta_{end} = \theta(e_4)$):

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

One sees clearly on this example that the constraints on the firing delays of the events of E do not suffice: for instance, the constraint $\theta(e_3) \le \theta(e_1)+4$ (occurrence of d enabled after e_1 and disabled by e_3) restricts the range of possible firing times for event e_3.

V. Preserving Partial Order Runs

A. Constraint on Parameters for an Abstract Process

We can now come back to our parameter synthesis problem. For this we consider a *parametric* TPN $\mathcal{N}$. The first step is, given an abstract process E of $\mathcal{N}$, to find a constraint K on the parameters such that for every valuation v of the parameters it holds that $E \in Processes([\![\mathcal{N}]\!]_v)$ iff $v \models K$.

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

Definition 8. *For an abstract process E, we define $K_E^{\theta\lambda}$ as:*

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

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

and $K^{\theta\lambda}_{\{e_2,e_3,e_0\}} \stackrel{\text{def}}{=}$

$$\begin{cases} a_2 \le \theta(e_2) \le b_2 \quad \text{(firing delay of } e_2\text{)} \\ a_3 \le \theta(e_3) - \theta(e_2) \le b_3 \quad \text{(firing delay of } e_3\text{)} \\ a_0 \le \theta(e_0) - \theta(e_3) \le b_0 \quad \text{(firing delay of } e_0\text{)} \\ \theta(e_0) \le b_1 \quad \text{(occurrence of } t_1 \text{ disabled by } e_0\text{)} \end{cases}$$

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 $a_2+a_3+a_0 \le \theta(e_0) \le b_1$). This confirms that $\{e_1, e_2, e_3\}$ is the only maximal abstract process feasible in $[\![\mathcal{N}]\!]_{v_0}$.

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 9. *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 the values of the parameters for which the instantiated model $[\![\mathcal{N}]\!]_v$ can execute the abstract process E.

Coming back to the example of Fig. 3, we get, for the abstract process $\{e_1, e_2, e_3\}$, the constraint

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

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 $a_2+a_3+a_0 \le b_1$ (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.

B. Parameter Synthesis Preserving Partial Order Semantics

We now have all the necessary bricks to define our procedure IMPO (standing for "inverse method based on partial orders") for synthesizing parameters in a PTPN $\mathcal{N}$ that guarantee the preservation of its partial order semantics. More precisely, we are looking for a constraint on the parameters Λ of $\mathcal{N}$ guaranteeing that all maximal processes of the instantiated models are already maximal processes of the model instantiated with the reference valuation v_0. Notice 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 sequence feasible with the initial valuation v_0 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 IMPO 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 [Kho], [Sch]. The procedure $\mathsf{IMPO}(\mathcal{N}, v_0)$ is the following:

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[3]; they are the abstract processes of our PTPN $\mathcal{N}$.
3) For every $E \in MP$, construct the constraint K_E^λ 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:

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

Theorem 2. *Let $\mathcal{N}$ be a PTPN, let v_0 be a parameter valuation. Assume $\mathsf{IMPO}(\mathcal{N}, v_0)$ terminates with result K. Then for all valuation v of the parameters satisfying the initial constraint K_0 of the model,*

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

In particular $v_0 \models K$.

Proof: Let v be a parameter valuation such that $\mathit{Processes}([\![\mathcal{N}]\!]_v) \subseteq \mathit{Processes}([\![\mathcal{N}]\!]_{v_0})$. For every maximal process $E = \{e_1, \ldots, e_n\} \in MP$, $v_0 \not\models K_E^\lambda$ implies that there exists no valuation $\theta_1, \ldots, \theta_n \in \mathbb{R}_+$ of the variables $\theta(e_1), \ldots, \theta(e_n)$ such that $\{(e_1, \theta_1), \ldots, (e_n, \theta_n)\}$ is a process of $[\![\mathcal{N}]\!]_{v_0}$. Then $E \notin \mathit{Processes}([\![\mathcal{N}]\!]_{v_0})$. We deduce that E is not a maximal abstract process of $[\![\mathcal{N}]\!]_v$; 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 $[\![\mathcal{N}]\!]_{v_0}$ 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 parameter valuation such that $\mathit{Processes}([\![\mathcal{N}]\!]_v) \not\subseteq \mathit{Processes}([\![\mathcal{N}]\!]_{v_0})$. Let E be an abstract process in $\mathit{Processes}([\![\mathcal{N}]\!]_v) \setminus \mathit{Processes}([\![\mathcal{N}]\!]_{v_0})$. 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 $v \not\models K$. ■

Example 3. *For the PTPN of Fig. 3, we have explained 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 $[\![\mathcal{N}]\!]_{v_0}$, i.e., $v_0 \not\models K^\lambda_{\{e_2,e_3,e_0\}}$. Then our procedure IMPO outputs the constraint*

$$K_0 \wedge a_2 + a_3 + a_0 > b_1 \,,$$

which is the negation of $K^\lambda_{\{e_2,e_3,e_0\}}$. 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. While IM requires t_1 to fire strictly before t_2, IMPO only requires that it fires before being disabled by t_0.

[3]The maximal processes can be extracted for instance by a SAT solver using an appropriate SAT encoding.

Let us now show that the output of IMPO is always more (or equally) permissive than the output of IM.

Theorem 3. *Let $\mathcal{N}$ be a PTPN with only finite executions, and let v_0 be a parameter valuation. Denote K_{IM} the constraint output by IM and K_{IMPO} the constraint output by IMPO. Then*

$$\{v_0\} \subseteq \{v \mid v \models K_{\mathsf{IM}}\} \subseteq \{v \mid v \models K_{\mathsf{IMPO}}\} \,.$$

Proof: By Theorem 1, $\{v_0\} \subseteq \{v \mid v \models K_{\mathsf{IM}}\}$. Now, let v be a parameter valuation satisfying K_{IM}. Again by Theorem 1, $\mathit{Sequences}([\![\mathcal{N}]\!]_v) \subseteq \mathit{Sequences}([\![\mathcal{N}]\!]_{v_0})$. We show that $\mathit{Processes}([\![\mathcal{N}]\!]_v) \subseteq \mathit{Processes}([\![\mathcal{N}]\!]_{v_0})$: indeed, every maximal abstract process E feasible for $[\![\mathcal{N}]\!]_v$ is the image by Π of a maximal timed word $((t_1, \theta_1), \ldots, (t_n, \theta_n))$ feasible for $[\![\mathcal{N}]\!]_v$, whose corresponding time-abstract word $(t_1, \ldots, t_n)$ is in $\mathit{Sequences}([\![\mathcal{N}]\!]_v)$. Because $\mathit{Sequences}([\![\mathcal{N}]\!]_v) \subseteq \mathit{Sequences}([\![\mathcal{N}]\!]_{v_0})$, we have that $(t_1, \ldots, t_n)$ is also in $\mathit{Sequences}([\![\mathcal{N}]\!]_{v_0})$, i.e., there exist dates $\theta'_1, \ldots, \theta'_n$ (notice that they are not necessarily the same as the θ_i) such that $((t_1, \theta'_1), \ldots, (t_n, \theta'_n))$ is feasible for $[\![\mathcal{N}]\!]_{v_0}$. The image by Π of this timed word is a process of $[\![\mathcal{N}]\!]_{v_0}$ whose set of events (determined only by the time-abstract word) is E. Then $E \in \mathit{Processes}([\![\mathcal{N}]\!]_{v_0})$.

To conclude, $\mathit{Processes}([\![\mathcal{N}]\!]_v) \subseteq \mathit{Processes}([\![\mathcal{N}]\!]_{v_0})$, and by Theorem 2, $v \models K_{\mathsf{IMPO}}$. ■

C. An Alternative Method for Restricted Cyclic Models

Our method IMPO first constructs all maximal processes, and then infers parameter valuations to preserve partial orders. For cyclic systems, this method will not terminate. We leave the cyclic case as future works (see Section VII); however, we would like to at least 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 $\mathsf{IMPO'}(\mathcal{N}, v_0)$, 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 $\mathit{Processes}([\![\mathcal{N}]\!]_{v_0})$ of maximal abstract processes feasible for $[\![\mathcal{N}]\!]_{v_0}$; one way to do this is to compute the finite set $\mathit{Sequences}([\![\mathcal{N}]\!]_{v_0})$, and then represent every sequence as a process, as explained in Definition 5.
2) For every $E \in \mathit{Processes}([\![\mathcal{N}]\!]_{v_0})$, 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 IMPO' 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. *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 $K = \mathsf{IMPO'}(\mathcal{N}, v_0)$. Then for all valuation v of the parameters satisfying the initial constraint K_0 of the model,*

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

In particular $v_0 \models K$.

Proof: Let v be a parameter valuation such that $\mathit{Processes}([\![\mathcal{N}]\!]_v) \subseteq \mathit{Processes}([\![\mathcal{N}]\!]_{v_0})$. Let E' be a prefix of an abstract process $E \in \mathit{Processes}([\![\mathcal{N}]\!]_{v_0})$ 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 $\mathit{Processes}([\![\mathcal{N}]\!]_v) \subseteq \mathit{Processes}([\![\mathcal{N}]\!]_{v_0})$, $E' \cup \{e\}$ 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 $\mathit{Processes}([\![\mathcal{N}]\!]_v) \not\subseteq \mathit{Processes}([\![\mathcal{N}]\!]_{v_0})$. Let E be an abstract process in $\mathit{Processes}([\![\mathcal{N}]\!]_v) \setminus \mathit{Processes}([\![\mathcal{N}]\!]_{v_0})$. Compute a prefix E' of E by removing events one by one (starting by those that are maximal w.r.t. $\rightarrow$ 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 $v \not\models K$. ∎

As a consequence, when IMPO can be applied, IMPO and IMPO' return equivalent constraints.

VI. Application to Asynchronous Circuits

A. Improving Latencies in Asynchronous Circuit Design

In this section we apply IMPO to the asynchronous circuit mentioned in the introduction (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 [CEFX09].

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) 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

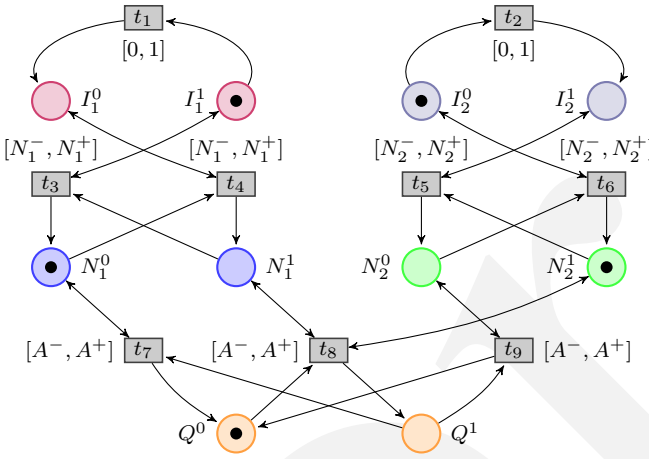

Figure 5: A PTPN model for the circuit of Fig. 1

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 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 IMPO.

Let us now apply IMPO to $\mathcal{N}$ and v_0. First, IMPO 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 IMPO 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 $i \in \{1, \ldots, 9\}$, we denote by $\theta_i \stackrel{\text{def}}{=} \theta(e_i)$ the rational variable associated to e_i.

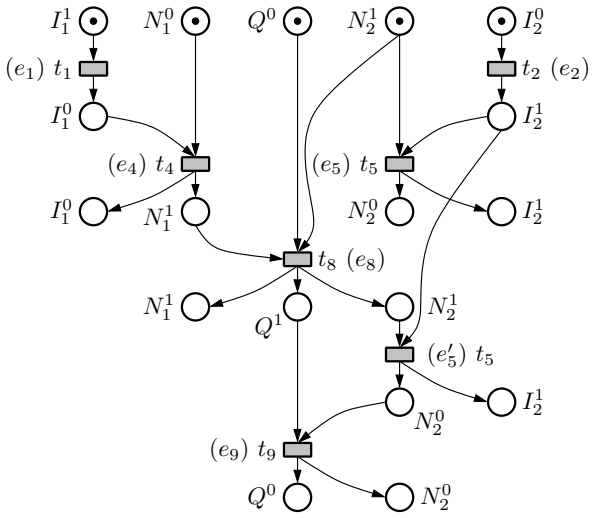

Figure 6: Untimed unfolding of Fig. 5

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

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

For event e_5, the firing constraint is

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

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

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

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

$$\begin{aligned} &A^- \le \theta_8 - \theta_4 \le A^+ \qquad N_2^- \le \theta'_5 - \max\{\theta_8, \theta_2\} \le N_2^+ \\ &A^- \le \theta_9 - \theta'_5 \le A^+ \end{aligned} \tag{7}$$

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

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

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 $K_{E_2}^{\theta\lambda}$, IMPO eliminates all variables θ_i from each constraint, 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 IMPO adds the negation of $K_{E_2}^{\lambda}$ to K. In the end, IMPO returns the constraint

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

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: $N_2^+ + 1$.

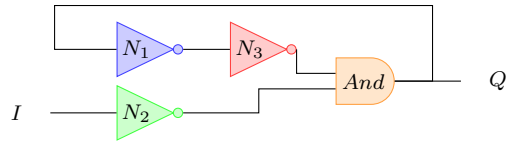

Figure 7: An asynchronous circuit (looping variant)

Now, observe that this constraint allows for a sequential execution where t_4 fires before t_5, which the original IM 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.

B. 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). 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 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 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 inequality 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 IMPO to $\mathcal{N}$ does not terminate: indeed, IMPO 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, IMPO' does terminate and outputs the constraint:

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

In contrast to IM, IMPO' 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).

VII. 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 IMPO (and its variant IMPO') 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 usually generates very few disjunctions on typical examples. For the examples presented in this paper, 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 the method. Second, the method 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 of IM does (for the preservation of sequential semantics). Finally, several mature verification tools (such as UPPAAL [LPY97] or ROMÉO [LRST09]) deal with such non-convex constraints; they use efficient representations and achieve very satisfactory performances in practice. The Parma polyhedra library [BHZ08] (used in ROMÉO and IMITATOR) also offers such a representation.

As future work, we would like to generalize our technique to cyclic models, but there are several difficulties. First, the classical way to deal with partial-order techniques for cyclic models is to define cut-off events; but no efficient cut-off criterion exists for timed models. Second, we have explained in Section V-B that our method focuses on *maximal* abstract processes in order to output the most permissive constraint. For cyclic nets, of course, some maximal abstract processes are infinite, which makes IMPO inapplicable. A pragmatic approach that we would like to investigate in the future is to consider partial orders by blocks. That is, we could explore the symbolic tree by steps of n transitions (e.g., $n = 10$), and derive constraints allowing the interleaving within each block. The constraint output would be weaker than IM (equivalent for $n = 1$) but most probably stricter than IMPO; however, the method could terminate also for cyclic models.

We aim at implementing our approach in a near future; The implementation IMITATOR of the inverse method IM, together with a prototype that computes the constraints K^θ, will be good candidates for our implementation. An implementation will especially be useful to experimentally compare the respective efficiency of IMPO and IMPO'; whereas the latter has better termination, it may explore more states, since it explores all prefixes of maximal processes.

References

[ACEF09] Étienne André, Thomas Chatain, Emmanuelle Encrenaz, and Laurent Fribourg. An inverse method for parametric timed automata. *IJFCS*, 20(5):819–836, 2009. 1, 3

[AHV93] Rajeev Alur, Thomas A. Henzinger, and Moshe Y. Vardi. Parametric real-time reasoning. In *STOC*, pages 592–601. ACM, 1993. 1, 3

[AL00] Tuomas Aura and Johan Lilius. A causal semantics for time Petri nets. *Theoretical Computer Science*, 243(1-2):409–447, 2000. 1, 4, 5

[APP13] Étienne André, Laure Petrucci, and Giuseppe Pellegrino. Precise robustness analysis of time Petri nets with inhibitor arcs. In *FORMATS*, volume 8053 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2013. 1, 2, 3

[AS11] Étienne André and Romain Soulat. Synthesis of timing parameters satisfying safety properties. In *RP*, volume 6945 of *Lecture Notes in Computer Science*, pages 31–44. Springer, 2011. 1, 3

[BHR06] P. Bouyer, S. Haddad, and P.-A. Reynier. Timed unfoldings for networks of timed automata. In *ATVA*, volume 4218 of *Lecture Notes in Computer Science*, pages 292–306. Springer, 2006. 1

[BHZ08] Roberto Bagnara, Patricia M. Hill, and Enea Zaffanella. 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):3–21, 2008. 10

[BJLY98] J. Bengtsson, B. Jonsson, J. Lilius, and W. Yi. Partial order reductions for timed systems. In *CONCUR*, volume 1466 of *Lecture Notes in Computer Science*, pages 485–500. Springer, 1998. 1

[BL09] Laura Bozzelli and Salvatore La Torre. Decision problems for lower/upper bound parametric timed automata. *Formal Methods in System Design*, 35(2):121–151, 2009. 1

[BR07] Véronique Bruyère and Jean-François Raskin. Real-time model-checking: Parameters everywhere. *Logical Methods in Computer Science*, 3(1:7), 2007. 1

[CC05] Robert Clarisó and Jordi Cortadella. Verification of concurrent systems with parametric delays using octahedra. In *ACSD*, pages 122–131. IEEE Computer Society, 2005. 1

[CCJ06] F. Cassez, T. Chatain, and C. Jard. Symbolic unfoldings for networks of timed automata. In *ATVA*, volume 4218 of *Lecture Notes in Computer Science*, pages 307–321. Springer, 2006. 1

[CEFX09] Rémy Chevallier, Emmanuelle Encrenaz-Tiphène, Laurent Fribourg, and Weiwen Xu. Timed verification of the generic architecture of a memory circuit using parametric timed automata. *FMSD*, 34(1):59–81, 2009. 1, 2, 8

[CJ06] Thomas Chatain and Claude Jard. Complete finite prefixes of symbolic unfoldings of safe time Petri nets. In *ICATPN*, volume 4024 of *Lecture Notes in Computer Science*, pages 125–145, 2006. 1

[Eng91] Joost Engelfriet. Branching processes of Petri nets. *Acta Informatica*, 28(6):575–591, 1991. 4

[JLR15] Aleksandra Jovanović, Didier Lime, and Olivier H. Roux. Integer parameter synthesis for timed automata. *IEEE Transactions on Software Engineering*, 2015. To appear. 1

[Kho] V. Khomenko. Punf. http://homepages.cs.ncl.ac.uk/victor.khomenko/tools/punf/. 7

[LNZ05] Denis Lugiez, Peter Niebert, and Sarah Zennou. A partial order semantics approach to the clock explosion problem of timed automata. *Theoretical Computer Science*, 345(1):27–59, 2005. 1

[LPY97] Kim G. Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1-2):134–152, 1997. 10

[LRST09] Didier Lime, Olivier H. Roux, Charlotte Seidner, and Louis-Marie Traonouez. Romeo: A parametric model-checker for Petri nets with stopwatches. In *TACAS*, volume 5505 of *Lecture Notes in Computer Science*, pages 54–57. Springer, 2009. 10

[Mar11] Nicolas Markey. Robustness in real-time systems. In *SIES*, pages 28–34. IEEE Computer Society Press, 2011. 3

[MF76] Philip M. Merlin and David J. Farber. Recoverability of communication protocols – implications of a theoretical study. *IEEE Transactions on Communications*, 24, 1976. 2

[Min99] Marius Minea. Partial order reduction for model checking of timed automata. In *CONCUR*, volume 1664 of *Lecture Notes in Computer Science*, pages 431–446. Springer, 1999. 1

[NQ06] Peter Niebert and Hongyang Qu. Adding invariants to event zone automata. In *FORMATS*, volume 4202 of *Lecture Notes in Computer Science*, pages 290–305. Springer, 2006. 1

[PP01] W. Penczek and A. Pólrola. Abstractions and partial order reductions for checking branching properties of time Petri nets. In *ICATPN*, volume 2075 of *Lecture Notes in Computer Science*, pages 323–342, 2001. 1

[Sch] S. Schwoon. Mole. http://lsv.ens-cachan.fr/~schwoon/tools/mole/. 7

[TGJ+10] Louis-Marie Traonouez, Bartosz Grabiec, Claude Jard, Didier Lime, and Olivier H. Roux. Symbolic unfolding of parametric stopwatch Petri nets. In *ATVA*, volume 6252 of *Lecture Notes in Computer Science*, pages 291–305. Springer, 2010. 1

[TLR09] Louis-Marie Traonouez, Didier Lime, and Olivier H. Roux. Parametric model-checking of stopwatch Petri nets. *Journal of Universal Computer Science*, 15(17):3273–3304, 2009. 1, 2, 3

[VP99] Irina Virbitskaite and E. Pokozy. A partial order method for the verification of time Petri nets. In *FCT*, volume 1684 of *Lecture Notes in Computer Science*, pages 547–558. Springer, 1999. 1

[YS97] Tomohiro Yoneda and Bernd-Holger Schlingloff. Efficient verification of parallel real-time systems. *Formal Methods in System Design*, 11(2):187–215, 1997. 1

author