International Journal of Foundations of Computer Science

AN INVERSE METHOD FOR PARAMETRIC TIMED AUTOMATA*

ÉTIENNE ANDRÉ, THOMAS CHATAIN, LAURENT FRIBOURG

LSV – ENS Cachan & CNRS
61 avenue du Président Wilson, 94230 Cachan, France
{*andre, chatain, fribourg*}*@lsv.ens-cachan.fr*

EMMANUELLE ENCRENAZ

LIP6 – Université Pierre et Marie Curie & CNRS
4 place Jussieu, 75005 Paris, France
emmanuelle.encrenaz@lip6.fr

Received (Day Month Year)
Accepted (Day Month Year)
Communicated by (xxxxxxxxxx)

We consider in this paper systems modeled by timed automata. The timing bounds involved in the action guards and location invariants of our timed automata are not constants, but *parameters*. Those *parametric timed automata* allow the modelling of various kinds of timed systems, e.g. communication protocols or asynchronous circuits. We will also assume that we are given an initial tuple π_0 of values for the parameters, which corresponds to values for which the system is known to behave properly. Our goal is to compute a constraint K_0 on the parameters, satisfied by π_0, guaranteeing that, under any parameter valuation satisfying K_0, the system behaves in the same manner : for any two parameter valuations satisfying K_0, the behaviors of the timed automata are *(time-abstract) equivalent*, i.e., the traces of execution viewed as alternating sequences of actions and locations are identical. We present an algorithm *InverseMethod* that terminates in the case of acyclic models, and discuss how to extend it in the cyclic case. We also explain how to combine our method with classical synthesis methods which are based on the avoidance of a given set of bad states. A prototype implementation has been done, and various experiments are described.

Keywords: Parameter Synthesis, Reachability Analysis, Time-Abstract Equivalence.

1. Introduction

Timed automata are finite control automata equipped with *clocks*, which are real-valued variables which increase uniformly. This model is useful for reasoning about real-time systems, because one can specify quantitatively the interval of time during which the transitions can occur, using the bounds involved in invariants and

*This work is partially supported by the Agence Nationale de la Recherche, grant ANR-06-ARFU-005, and by Institut Farman (ENS Cachan).

guards. However, the behavior is very sensitive to the values of these bounds, and it is rather difficult to find their correct values. It is therefore interesting to reason parametrically, by considering that these bounds are *unknown constants*, or *parameters* and try to synthesize a *constraint* on these parameters, in order to ensure a correct behavior. Such automata are called *parametric timed automata* (PTA).

Context. The synthesis of constraints for PTA (and more generally for the larger class of parametric hybrid automata) has been mainly done by supposing one is given a set of "bad states" (see, e.g., [14]). The goal is to find a set of parameters for which the considered timed (or hybrid) automaton does not reach a given set of bad states. The problem is known to be semi-solvable (if the algorithm terminates the result is correct) by introducing the parameters as state variables and computing the set of reachable states. We call such a method a *classical* (or "bad-state oriented") method.

The parameter design problem for parametric hybrid automata was formulated and solved by [16], but the proposed solution is tractable for only very simple systems with few parameters. If a counterexample is found, i.e., if there is a path reaching a bad state, then the current constraint on the parameters is refined in order to make the counterexample infeasible. If all counterexamples have been eliminated, the resulting constraint describes a set of parameters for which the system is safe, in the sense that no path reaches the set of bad states.

In order to increase the efficiency (and the termination) of the method, some approximations are sometimes used for implementing the operator on the constraint that makes the counterexample infeasible (e.g., [14]). This is in the style of CEGAR-based methods (counter-example guided abstraction refinement [10]).

The synthesis of constraints has been implemented in the context of PTA or hybrid systems, e.g. in [6] using tool TReX [11], or in [17] using an extension of Uppaal [18] for linear parametric model checking. Note that [6] is able to infer non-linear constraints. Another interesting related work on PTA is presented in [17], which gives decidability results for the verification of a special class, called "L/U automata". Two subclasses of L/U automata, called lower-bound and upper-bound PTA, are also considered in [24], with decidability results. The synthesis of constraints has been studied more specifically in the context of asynchronous circuits, mainly by Myers and co-workers (see, e.g., [26]), and by Clarisó and Cortadella (see, e.g., [9, 8]). They also proceed by analyzing failure traces and generating timing constraints that prevent the occurrence of such failures.

Contribution. In this paper, we propose an *inverse* (or "good-state oriented") method, which does not suppose given a set of bad states that should be avoided, but rather a "good instantiation" π_0 of the parameters that one wants to generalize. More precisely, we want to generate a constraint K_0 on the parameters that corresponds to a set such that, for all instantiation π of parameters in this set, the behavior of the timed automaton $\mathcal{A}$ is *(time-abstract) equivalent* to the behavior

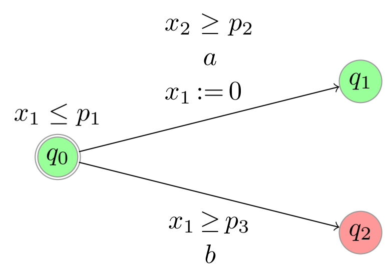

Fig. 1. A parametric timed automaton

of $\mathcal{A}$ under π_0, in a sense that will be defined later.

Our procedure consists of generating runs starting from the initial state, and removing states incompatible with the reference values by appropriately refining the current constraint K_0 on the parameters. The generation procedure is then restarted until a new incompatible state is produced, and so on, iteratively until no incompatible state is generated.

Toy example. Let us consider the parametric timed automaton (PTA) schematized on Fig. 1, following the formalism defined in Sect. 2.2. This PTA contains two clocks x_1 and x_2, three parameters p_1, p_2 and p_3, and three locations q_0, q_1 and q_2. The initial location q_0 has invariant $x_1 \leq p_1$. The transition from q_0 to q_1, labelled a, has guard $x_2 \geq p_2$, and resets x_1. The transition from q_0 to q_2, labelled b, has guard $x_1 \geq p_3$, and does not reset any clock.

Let us assume that q_2 corresponds to a "bad location". Classical methods, using this information, will generate the constraint $Z : p_1 < p_3$, which guarantees that the location is not reachable.

Suppose now that we are given the following "good" instantiation of the parameters $\pi_0 : p_1 = 4 \land p_2 = 2 \land p_3 = 6$, under which the PTA is assumed to have a "good" behavior. Then our inverse method will generate the constraint $K_0 : p_1 < p_3 \land p_2 \leq p_1$. For all instantiation π of the parameters satisfying this constraint, our method guarantees that the PTA will behave in the same manner as under π_0. We are thus ensured that the behavior of the PTA is correct. Note that K_0 is strictly smaller than Z. On the one hand, this may be viewed as a limitation of our method. On the other hand, this may indicate that there are incorrect behaviors other than those corresponding to the reaching of q_2. For example, there are some parameter instantiations satisfying Z, under which a deadlock of the PTA occurs at the initial location q_0. In contrast, our inverse method guarantees that such a deadlock is impossible under any instance satisfying K_0 (because the deadlock does not occur under π_0).

Overview of the paper. We first introduce the notion of Parametric Timed Automata in Sect. 2. Then we present our method of synthesis of constraints in Sect. 3, and show its correctness. We then present in Sect. 4 a variant of the method in order to improve termination. We then describe some experiments in Sect. 5. We then present in Sect. 6 an extension allowing to enlarge the synthesized constraint by exploiting some additional information, possibly issued from classical methods. We give some final remarks in Sect. 7.

2. Parametric Timed Automata

These preliminary definitions are mainly borrowed from [17].

2.1. *Constraints on the Clocks and the Parameters*

Throughout this paper, we assume a fixed set $X = \{x_1, \dots, x_H\}$ of *clock variables*. A *clock variable* is a variable x_i with value in $\mathbb{R}_+$, where $\mathbb{R}_+$ is the standard notation for the set of real numbers greater or equal to 0. All clocks evolve linearly at the same rate. We define a *clock valuation* as a function $w : X \to \mathbb{R}_+$ assigning a nonnegative real value to each clock variable. We will often identify a valuation w with the point $(w(x_1), \dots, w(x_H))$. Given a constant $d \in \mathbb{R}_+$, we use $w + d$ to denote $(w(x_1) + d, \dots, w(x_H) + d)$.

Throughout this paper, we assume a fixed set $P = \{p_1, \dots, p_M\}$ of *parameters*. A *parameter valuation* π is a function $\pi : P \to \mathbb{R}_+$ assigning a nonnegative real value to each parameter. There is a one-to-one correspondence between valuations and points in $(\mathbb{R}_+)^M$. We will often identify a valuation π with the point $(\pi(p_1), \dots, \pi(p_M))$.

Definition 1. *A* linear inequality on the parameters P *(resp.* linear inequality on the clock variables X and the parameters P*) is an inequality $e \prec e'$, where $\prec \in \{<, \leq\}$, and e, e' are two linear terms of the form*

$$\Sigma_i \alpha_i p_i + d, \qquad (resp.\ \ \Sigma_i \alpha_i p_i + \Sigma_j \beta_j x_j + d)$$

where $1 \leq i \leq M, 1 \leq j \leq H$ and $\alpha_i, \beta_j, d \in \mathbb{N}$.

A (convex) constraint on the parameters P *(resp.* (convex) constraint on the clock variables X and the parameters P*) is a conjunction of inequalities on P (resp. on X and P).*

In the sequel, J will denote a linear inequality on the parameters, and the letter K (resp. C) will denote a constraint on the parameters (resp. on the clocks and the parameters). The negation of a linear inequality J of the form $e < e'$ (resp. $e \leq e'$) is the linear inequality $e' \leq e$ (resp. $e' < e$), and will be denoted by $\neg J$.

Given a parameter valuation π and a constraint C, $C[\pi]$ denotes the constraint obtained by replacing each parameter p in C with $\pi(p)$. Likewise, given a clock valuation w, $C[\pi][w]$ denotes the expression obtained by replacing each clock variable x in $C[\pi]$ with $w(x)$. A clock valuation w *satisfies* constraint $C[\pi]$ (denoted by $w \models C[\pi]$) if $C[\pi][w]$ evaluates to true. We say that a parameter valuation π *satisfies* a constraint C, denoted by $\pi \models C$, if the set of clock valuations that satisfy $C[\pi]$ is nonempty. We use the notation $<w, \pi> \models C$ to indicate that $C[\pi][w]$ evaluates to true. We say that $C \subseteq D$ if $\forall w, \pi : <w, \pi> \models C \Rightarrow <w, \pi> \models D$. We say that $C = D$ if $C \subseteq D$ and $D \subseteq C$.

Given a constraint C, it is sometimes convenient to rename the set of variables $X = \{x_1, \ldots, x_H\}$ as $X' = \{x'_1, \ldots, x'_H\}$. We use the notation $C(X)$ (resp. $C(X')$) to indicate that X (resp. X') is the set of clock variables occurring in C.

Similarly to the semantics of constraints on the clocks and the parameters, we say that a parameter valuation π *satisfies* a constraint K on the parameters, denoted by $\pi \models K$, if the expression obtained by replacing each parameter p in K with $\pi(p)$ evaluates to true. We will consider *True* as a constraint on the parameters, corresponding to the set of all possible values for P.

Given a constraint C on the clocks and the parameters, we denote by $\exists X : C$ the constraint on the parameters obtained from C after elimination of the clock variables, i.e., $\{\pi \mid \exists w : <w, \pi> \models C\}$.

2.2. *Parametric Timed Automata*

We assume familiarity with timed automata (considered in, e.g., [20]), which are an extension of the class of standard automata [1]. All clock constraints of standard timed automata are boolean combinations of atomic conditions that compare values with natural numbered *constants*. With respect to the classical definition, this class is contrived by the fact that guards and invariants are necessarily in conjunctive form, but this is not restrictive in practice.

The following definition is an extension of the class of timed automata to the parametric case. Parametric timed automata allow within guards and invariants the use of parameters in place of constants (see [2]).

Definition 2. *Given a set of clocks X and a set of parameters P, a* parametric timed automaton (PTA) $\mathcal{A}$ *is a 6-tuple of the form* $\mathcal{A} = (\Sigma, Q, q_0, K, I, \rightarrow)$, *where:*

- Σ *is a finite set of actions,*
- Q *is a finite set of locations,*
- $q_0 \in Q$ *is the initial location,*
- K *is a constraint on the parameters* P,
- I *is the invariant, assigning to every* $q \in Q$ *a constraint* I_q *on the clocks and the parameters, and*
- $\rightarrow$ *is a step relation consisting of elements of the form* (q, g, a, ρ, q'), *also denoted by* $q \overset{g,a,\rho}{\rightarrow} q'$, *where* $q, q' \in Q$, $a \in \Sigma$, $\rho \subseteq X$ *is a set of clock variables to be reset by the step, and* g *(the step guard) is a constraint on the clocks and the parameters.*

In the sequel, we will consider the PTA $\mathcal{A} = (\Sigma, Q, q_0, K, I, \rightarrow)$. We will simply denote this automaton by $\mathcal{A}(K)$, in order to emphasize the fact that only K will change in $\mathcal{A}$.

We use $X' = \rho(X)$, where X' is a renaming of X, to denote the conjunction of equalities $x'_i = 0$ for all $x_i \in \rho$, and $x'_i = x_i$ for all the other variables x_i of X.

For every parameter valuation $\pi = (\pi_1, \ldots, \pi_M)$, $\mathcal{A}[\pi]$ denotes the PTA $\mathcal{A}(K)$, where K is $\bigwedge_{i=1}^{M} p_i = \pi_i$. This corresponds to the PTA obtained from $\mathcal{A}$ by substituting every occurrence of a parameter p_i by π_i in the guards and invariants. Note that $\mathcal{A}[\pi]$ is a standard timed automaton. (Strictly speaking, $\mathcal{A}[\pi]$ is only a timed automaton if π assigns an integer to each parameter.)

In the sequel, we suppose that one is given a valuation π of the parameters, and the PTA $\mathcal{A}[\pi]$.

Definition 3. *A* labeled transition system (LTS) *over a set of symbols* Σ *is a triple* $\mathcal{L} = (S, S_0, \Rightarrow)$*, with* S *a set of* states*,* $S_0 \subset S$ *a set of* initial states*, and* $\Rightarrow \in S \times \Sigma \times S$ *a* transition relation*. We write* $s \overset{a}{\Rightarrow} s'$ *for* $(s, a, s') \in \Rightarrow$*. A* run *(of length* m*) of* $\mathcal{L}$ *is a finite alternating sequence of states* $s_i \in S$ *and symbols* $a_i \in \Sigma$ *of the form* $s_0 \overset{a_0}{\Rightarrow} s_1 \overset{a_1}{\Rightarrow} \cdots \overset{a_{m-1}}{\Rightarrow} s_m$*, where* $s_0 \in S_0$*. A state* s_m *is* reachable *if it is the last state of some run* R*. We say that* R reaches s_m*.*

We now give the concrete semantics of a PTA.

Definition 4. *The* concrete semantics of $\mathcal{A}[\pi]$ *is the LTS* $(S, S_0, \Rightarrow)$ *over* Σ *where*

$$S = \{(q, w) \in Q \times (X \to \mathbb{R}_+) \mid \langle w, \pi \rangle \models I_q\},$$
$$S_0 = \{(q_0, w) \mid \langle w, \pi \rangle \models I_{q_0} \wedge w = (w_0, \ldots, w_0) \text{ for some } w_0\}$$

and the transition predicate $\Rightarrow$ *is specified by the following three rules. For all* $(q, w), (q', w') \in S, d \geq 0$ *and* $a \in \Sigma$*,*

- $(q, w) \overset{a}{\to} (q', w')$ *if* $\exists g, \rho : q \overset{g,a,\rho}{\to} q'$ *and* $\langle w, \pi \rangle \models g$ *and* $w' = \rho(w)$ *;*
- $(q, w) \overset{d}{\to} (q', w')$ *if* $q' = q$ *and* $w' = w + d$ *;*
- $(q, w) \overset{a}{\Rightarrow} (q', w')$ *if* $\exists d, w'' : (q, w) \overset{a}{\to} (q', w'') \overset{d}{\to} (q', w')$*.*

We consider with the definition of S_0 that all clocks are initially set to 0, or have evolved linearly in the bounds given by I_{q_0}. A state (resp. run) in the concrete semantics will be referred to as a *concrete state* (resp. *concrete run*).

We now define a *trace* as a time-abstract run, i.e., an alternating sequence of locations and actions.

Definition 5. *Given a PTA* $\mathcal{A}$ *and a concrete run* R *of* $\mathcal{A}[\pi]$ *of the form* $(q_0, w_0) \overset{a_0}{\Rightarrow} \cdots \overset{a_{m-1}}{\Rightarrow} (q_m, w_m)$*, the* trace associated to R *is the alternating sequence of locations and actions* $q_0 \overset{a_0}{\Rightarrow} \cdots \overset{a_{m-1}}{\Rightarrow} q_m$*.*

In the following, the traces of $\mathcal{A}[\pi]$ will refer to the traces associated to the runs of $\mathcal{A}[\pi]$.

2.3. *Network of Parametric Timed Automata*

We now introduce the notion of network of parametric timed automata.

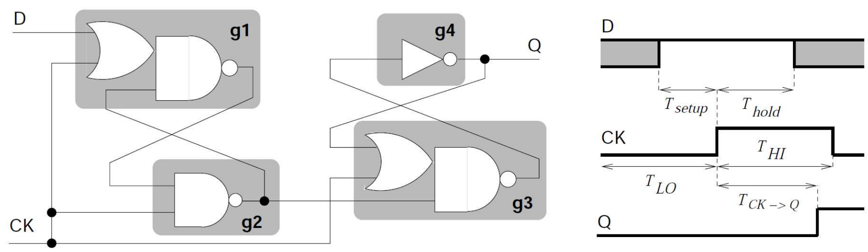

Fig. 2. Flip-flop circuit

Definition 6. *For all* $1 \leq i \leq N$, *let* $\mathcal{A}_i = (\Sigma_i, Q_i, (q_0)_i, K_i, I_i, \rightarrow_i)$ *be a PTA on a set of clocks* X_i *and a set of parameters* P_i. *The sets of locations* Q_i, *parameters* P_i, *and clocks* X_i *are mutually disjoint. A* network of parametric timed automata (NPTA) *is* $\mathcal{A} = \mathcal{A}_1 \| \ldots \| \mathcal{A}_N$, *where* $\|$ *is the standard operator for parallel composition.*

The global PTA over $X = \biguplus_{i=1}^{N} X_i$ and $P = \biguplus_{i=1}^{N} P_i$ obtained from the NPTA is $\mathcal{A} = (\Sigma, Q, q_0, K, I, \rightarrow)$, where $\Sigma = \bigcup_{i=1}^{N} \Sigma_i$, $Q = \Pi_{i=1}^{N} Q_i$, $q_0 = \langle (q_0)_1, \ldots, (q_0)_N \rangle$, $K = \bigwedge_{i=1}^{N} K_i$, $I_{\langle q_1, \ldots, q_N \rangle} = \bigwedge_{i=1}^{N} I_{q_i}$ for all $\langle q_1, \ldots, q_N \rangle \in Q$, and $\rightarrow$ is defined in the following. For all $a \in \Sigma$, let T_a be the subset of indices $i \in 1, \ldots, N$ such that $a \in \Sigma_i$. Then, for all $a \in \Sigma$, for all $\langle q_1, \ldots, q_N \rangle \in Q$, for all $\langle q'_1, \ldots, q'_N \rangle \in Q$, $(\langle q_1, \ldots, q_N \rangle, g, a, \rho, \langle q'_1, \ldots, q'_N \rangle) \in \rightarrow$ if :

- for all $i \in T_a$, there exist g_i, ρ_i s.t. $(q_i, g_i, a, \rho_i, q'_i) \in \rightarrow_i$,
- $g = \bigwedge_{i \in T_a} g_i$,
- $\rho = \bigcup_{i \in T_a} \rho_i$, and
- for all $i \notin T_a$, $q'_i = q_i$.

2.4. *The Inverse Problem*

In this paper, starting from an instantiation π_0 of the set P of parameters, we are interested in finding a constraint K_0 on the parameters, such that, for any valuation π of P satisfying K_0, the behaviors (in terms of sets of traces) of $\mathcal{A}[\pi]$ and $\mathcal{A}[\pi_0]$ will be the same. Formally, the *inverse problem* is stated as follows :

> Consider a PTA $\mathcal{A}$ and a valuation π_0 of the parameters. Find a constraint K_0 such that $\pi_0 \models K_0$ and, for all $\pi \models K_0$, the set of traces of $\mathcal{A}[\pi_0]$ and the set of traces of $\mathcal{A}[\pi]$ are equal.

Example 7. *Consider an asynchronous "D flip-flop" circuit described in* [9] *and depicted on Fig. 2. It is composed of 4 gates* (G_1, G_2, G_3 *and* G_4) *interconnected in a cyclic way, and an environment involving two input signals D and CK. The global output signal is Q. Each gate* G_i *has a delay in the parametric interval* $[\delta_i^-, \delta_i^+]$, *with* $\delta_i^- \leq \delta_i^+$. *There are 4 other parameters (viz.,* $T_{HI}, T_{LO}, T_{setup}$, *and* T_{hold}) *used to*

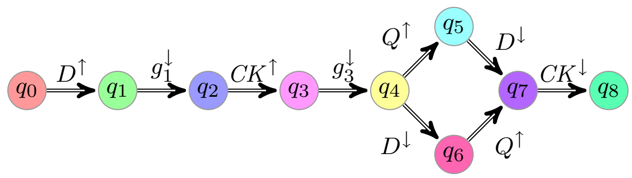

Fig. 3. Traces of the flip-flop circuit under π_0

model the environment. Each gate is modeled by a PTA, as well as the environment. We consider an inertial model for gates, where any change of the input may lead to a change of the output (after some delay). The (network of) PTA $\mathcal{A}$ modeling the system results from the composition of those 5 PTA. The output signal of a gate G_i is named g_i (note that $g_4 = Q$). The rising (resp. falling) edge of signal D is denoted by $D^\uparrow$ (resp. $D^\downarrow$) and similarly for signals $CK, Q, g_1, \ldots, g_4$. We consider the following instantiation π_0 of the parameters:

$$\begin{array}{llll} T_{HI} = 24 & T_{LO} = 15 & T_{setup} = 10 & T_{hold} = 17 \\ \delta_1^- = 7 & \delta_1^+ = 7 & \delta_2^- = 5 & \delta_2^+ = 6 \\ \delta_3^- = 8 & \delta_3^+ = 10 & \delta_4^- = 3 & \delta_4^+ = 7 \end{array}$$

We consider an environment starting from $D = CK = Q = 0$ and $g_1 = g_2 = g_3 = 1$, with the following ordered sequence of actions for inputs D and CK : $D^\uparrow$, $CK^\uparrow$, $D^\downarrow$, $CK^\downarrow$, as depicted on Fig. 2 right. Therefore, we have the implicit constraint $T_{setup} \leq T_{LO} \wedge T_{hold} \leq T_{HI}$. For this environment and the instantiation π_0, the set of traces of the system is depicted on Fig. 3 under the form of an oriented graph. We are now interested in finding other instantiations of the parameters yielding the same set of traces. We will therefore infer a constraint K_0 such that, for any instantiation $\pi \models K_0$, the set of traces under π is the same as under π_0.

3. The Inverse Method

3.1. *Symbolic Semantics of Parametric Timed Automata*

Definition 8. *A* symbolic state *s of $\mathcal{A}(K)$ is a couple (q, C) where q is a location, and C a constraint on the clocks and the parameters.*

For each valuation π of the parameters P, we may view a symbolic state s as the set of pairs (q, w) where w is a clock valuation such that $<w, \pi> \models C$.

We say that a state $s_1 = (q_1, C_1)$ is *included* in a state $s_2 = (q_2, C_2)$, denoted by $s_1 \subseteq s_2$, if $q_1 = q_2$ and $C_1 \subseteq C_2$. We say that two states $s_1 = (q_1, C_1)$ and $s_2 = (q_2, C_2)$ are *equal*, denoted by $s_1 = s_2$, if $q_1 = q_2$ and $C_1 = C_2$.

The *initial state* of $\mathcal{A}(K)$ is a symbolic state s_0 of the form (q_0, C_0), where $C_0 = K \wedge I_{q_0} \wedge \bigwedge_{i=1}^{H-1} x_i = x_{i+1}$. K is the initial constraint, I_{q_0} is the invariant of the initial state, and the rest of the expression lets clocks evolve from the same initial value.

The symbolic semantics of a PTA is given in the following. Given a symbolic state $s = (q, C)$, a symbolic step of the automaton from s is defined below. Given a constant $d \in \mathbb{R}_+$, we use $X + d$ to denote the set $\{x_1 + d, \dots, x_H + d\}$.

- $(q, C) \xrightarrow{a} (q', C')$ if $(q, g, a, \rho, q') \in \rightarrow$, and C' is a constraint on the clocks and the parameters defined, using the set of (renamed) clock variables X', by:
  $C'(X') = (\exists X :\ (C(X) \wedge g(X) \wedge X' = \rho(X) \wedge I_{q'}(X')))$.
- $(q, C) \xrightarrow{d} (q, C')$, where d is a new parameter with values in $\mathbb{R}_+$, which means that C' is given by:
  $C'(X') = (\exists X :\ (C(X) \wedge X' = X + d \wedge I_q(X')))$.
- $(q, C) \stackrel{a}{\Rightarrow} (q', C')$ if $\exists C''$ such that $(q, C) \xrightarrow{a} (q', C'')$ and $(q', C'') \xrightarrow{d} (q', C')$, i.e., C' is a constraint on the clocks and the parameters obtained by removing X and d from the following expression[a]:
  $C'(X') = (\exists X, d :\ (C(X) \wedge g(X) \wedge X' = \rho(X) \wedge I_{q'}(X') \wedge I_{q'}(X' + d)))$.
  It can be shown that C' can be put under the form of a (convex) constraint on the clocks and the parameters, using, e.g., Fourier-Motzkin elimination (see [21]) of X and d.

Definition 9. *A* symbolic run *of $\mathcal{A}(K)$ (of length m) is a finite alternating sequence of symbolic states and actions of the form:*

$$s_0 \stackrel{a_0}{\Rightarrow} s_1 \stackrel{a_1}{\Rightarrow} \cdots \stackrel{a_{m-1}}{\Rightarrow} s_m$$

such that for all $i = 0, \dots, m-1$, $a_i \in \Sigma$ and $s_i \stackrel{a_i}{\Rightarrow} s_{i+1}$ is a symbolic step of $\mathcal{A}(K)$.

Example 10. *Consider again the PTA $\mathcal{A}(True)$ described on Fig. 1. Starting from q_0, we consider an a-transition. Thus, we have $(q_0, C_0) \stackrel{a}{\Rightarrow} (q_1, C_1)$. We have $C_0 : x_1 \leq p_1 \wedge x_1 = x_2$. By eliminating X and d in $\exists X, d : x_1 \leq p_1 \wedge x_1 = x_2 \wedge x_2 \geq p_2 \wedge x_1' = 0 \wedge x_2' = x_2$, and renaming X' back to X, we get $C_1 : x_1 = 0 \wedge x_2 \leq p_1 \wedge x_2 \geq p_2$.*

One defines $Post^i_{\mathcal{A}(K)}(S)$ as the set of states reachable from S in exactly i steps, and $Post^*_{\mathcal{A}(K)}(S)$ as the set of all states reachable from S in $\mathcal{A}(K)$ (i.e., $Post^*_{\mathcal{A}(K)}(S) = \bigcup_{i \geq 0} Post^i_{\mathcal{A}(K)}(S)$).

In the sequel, we will be interested in computing the set $Post^*_{\mathcal{A}(K)}(\{s_0\})$, where s_0 is the initial state of $\mathcal{A}(K)$. Note that if $Post^{i+1}_{\mathcal{A}(K)}(\{s_0\}) = \emptyset$ (or, more generally, if $Post^{i+1}_{\mathcal{A}(K)}(\{s_0\}) \subseteq \bigcup_{j=0}^{i} Post^j_{\mathcal{A}(K)}(\{s_0\})$), then $Post^*_{\mathcal{A}(K)}(\{s_0\}) = \bigcup_{j=0}^{i} Post^j_{\mathcal{A}(K)}(\{s_0\})$.

We now define the notion of trace associated to a symbolic run.

Definition 11. *Given a PTA $\mathcal{A}$ and a symbolic run R of $\mathcal{A}$ of the form $(q_0, C_0) \stackrel{a_0}{\Rightarrow} \cdots \stackrel{a_{m-1}}{\Rightarrow} (q_m, C_m)$, the* trace associated to R *is the alternating sequence of locations*

[a]In $C'(X')$, we use the expression $I_{q'}(X') \wedge I_{q'}(X' + d)$ instead of $\forall 0 \leq e \leq d : I_{q'}(X' + e)$, using the fact that $I_{q'}$ is convex.

and actions $q_0 \stackrel{a_0}{\Rightarrow} \cdots \stackrel{a_{m-1}}{\Rightarrow} q_m$.

We define below the classical notion of time-abstract equivalence between any two runs (see, e.g., [1, 23, 20]).

Definition 12. *Let R_1 (resp. R_2) be a concrete run of $\mathcal{A}[\pi]$ or a symbolic run of $\mathcal{A}(K)$. R_1 and R_2 are* time-abstract equivalent *(or more simply* equivalent*) if the trace associated to R_1 is identical to the trace associated to R_2.*

Similarly, let $\mathcal{R}_1$ (resp. $\mathcal{R}_2$) be a set of concrete runs of $\mathcal{A}[\pi]$ or a set of symbolic runs of $\mathcal{A}(K)$. $\mathcal{R}_1$ and $\mathcal{R}_2$ are equivalent *if any run of $\mathcal{R}_1$ is equivalent to a run in $\mathcal{R}_2$, and conversely.*

3.2. *The Algorithm InverseMethod*

The algorithm *InverseMethod* makes use of the following notion of incompatible state.

Definition 13. *Given a valuation π_0 of the parameters, a symbolic state (q, C) is said to be* compatible with respect to π_0 *(or more simply* π_0-compatible*) if $\pi_0 \models \exists X : C$, i.e., if $\exists w : <w, \pi_0> \models C$.*

A state is said to be π_0-incompatible *if it is not π_0-compatible.*

We say that a set of states is π_0-compatible *if all its elements are π_0-compatible.*

The test of π_0-compatibility of a state (q, C) is done by eliminating the clock variables from $\exists X : C$ (e.g., using Fourier-Motzkin algorithm).

We now present the algorithm *InverseMethod* in Fig. 4. The inner *DO* loop removes all the π_0-incompatible states. The outer *DO* loop computes the set of all reachable states, and returns the intersection K_0 of all the constraints on the parameters associated to the states of S. Note that there are two possible sources of nondeterminism in the algorithm:

- when one selects a π_0-incompatible state (q, C) (i.e, $\pi_0 \not\models \exists X : C$), and
- when one selects an inequality J among the conjunction of inequalities $\exists X : C$, that is "responsible" for this π_0-incompatibility (i.e., such that $\pi_0 \not\models J$, hence $\pi_0 \models \neg J$).

Remark. Note that, at the last iteration of Algorithm *InverseMethod*, we have: $S = Post^*_{\mathcal{A}(K)}(\{s_0\})$.

Example 14. *Let us apply InverseMethod on our toy example of PTA depicted on Fig. 1, with the following $\pi_0 : p_1 = 4 \wedge p_2 = 2 \wedge p_3 = 6$. We start with $K = True$ and $S = \{(q_0, C_0)\}$ where C_0 is $x_1 \leq p_1 \wedge x_1 = x_2$. We test whether S is π_0-compatible: we compute $\exists X : C_0$, which gives the trivially π_0-compatible constraint True. Thus, S contains no π_0-incompatible state, and we compute $Post_{\mathcal{A}(K)}(S)$.*

ALGORITHM ***InverseMethod***$(\mathcal{A},\, \pi_0)$

Input $\mathcal{A}$: PTA
π_0 : Reference valuation of P
Output K_0 : Constraint on the parameters
Variables i : Current iteration
S : Current set of reachable states ($S = \bigcup_{j=0}^{i} Post^j_{\mathcal{A}(K)}(\{s_0\})$)
K : Current constraint on the parameters

$i := 0\,;\quad K := True\,;\quad S := \{s_0\}$
DO
 DO UNTIL S is π_0-compatible
  Select a π_0-incompatible state (q, C) of S
  Select an inequality J of $(\exists X : C)$ such that $\pi_0 \models \neg J$
  $K := K \wedge \neg J$
  $S := \bigcup_{j=0}^{i} Post^j_{\mathcal{A}(K)}(\{s_0\})$
 OD
  %% S π_0-compatible
 IF $Post_{\mathcal{A}(K)}(S) = \emptyset$
  THEN RETURN $K_0 := \bigcap_{(q,C) \in S} (\exists X : C)$
 FI
 $i := i + 1$
 $S := S \cup Post_{\mathcal{A}(K)}(S)$ *%% $S = \bigcup_{j=0}^{i} Post^j_{\mathcal{A}(K)}(\{s_0\})$*
OD

Fig. 4. Algorithm *InverseMethod*

We now have $S = \{(q_0, C_0), (q_1, C_1), (q_2, C_2)\}$ with $C_1 : x_1 = 0 \wedge x_2 \leq p_1 \wedge x_2 \geq p_2$ and $C_2 : x_1 = x_2 \wedge x_1 \leq p_1 \wedge x_1 \geq p_3$. We test whether S is π_0-compatible: we compute $\exists X : C_1$, which gives the π_0-compatible constraint $p_2 \leq p_1$. We compute $\exists X : C_2$, and get $p_3 \leq p_1$, which is π_0-incompatible. The only inequality J to be negated is $p_3 \leq p_1$. Thus: $K = p_1 < p_3$.

We now perform $S := \bigcup_{j=0}^{i} Post^j_{\mathcal{A}(K)}(\{(q_0, C_0)\})$ with $i = 1$, which gives $S = \{(q_0, C'_0), (q_1, C'_1)\}$, with $C'_0 : x_1 \leq p_1 \wedge x_1 = x_2 \wedge p_1 < p_3$ and $C'_1 : x_1 = 0 \wedge x_2 \leq p_1 \wedge x_2 \geq p_2 \wedge p_1 < p_3$. We test whether S is π_0-compatible: we compute $\exists X : C'_0$, which gives the π_0-compatible constraint $p_1 < p_3$; we compute $\exists X : C'_1$, which gives the π_0-compatible constraint $p_2 \leq p_1 \wedge p_1 < p_3$. As we have no π_0-incompatible state, we leave the inner DO *loop. We then perform $S := Post_{\mathcal{A}(K)}(S)$, which gives the empty set. The algorithm thus terminates returning $K_0 = (\exists X : C'_0) \wedge (\exists X : C'_1)$, i.e.: $p_2 \leq p_1 \wedge p_1 < p_3$.*

3.3. *Correctness*

We now formally establish the correctness of Algorithm *InverseMethod*.

We suppose in this subsection that *InverseMethod*$(\mathcal{A}, \pi_0)$ terminates with output K_0. Let K (resp. S) be the current constraint on the parameters (resp. the current set of reachable states) when the algorithm terminates. We have $S = Post^*_{A(K)}(\{s_0\})$ and $K_0 = \bigcap_{(q,C)\in S}(\exists X : C)$.

Proposition 15. *We have $K_0 \subseteq K$.*

Proof. From the semantics of PTA, for all state $(q, C) \in S$, we have $(\exists X : C) \subseteq K$, since $S = Post^*_{A(K)}(\{s_0\})$. As $K_0 = \bigcap_{(q,C)\in S}(\exists X : C)$, then $K_0 \subseteq K$. □

Let us now show that $\pi_0 \models K_0$.

Proposition 16. *Given π_0, let K_0 =InverseMethod$(\mathcal{A}, \pi_0)$. We have : $\pi_0 \models K_0$.*

Proof. When Algorithm *InverseMethod* terminates, the set S is π_0-compatible (i.e., $\pi_0 \models (\exists X : C)$, for all $(q, C) \in S$). Thus, the intersection K_0 of the constraints associated to the states of S, i.e., $\bigcap_{(q,C)\in S}(\exists X : C)$, is satisfied by π_0. □

Let us now show that the set of traces in the concrete semantics and the set of traces in the symbolic semantics are equal. This will lead to Theorem 22, stating the correctness of Algorithm *InverseMethod*. First of all, we state that, for each symbolic run of $\mathcal{A}(K)$, we can find an equivalent concrete run of $\mathcal{A}[\pi]$.

Proposition 17. *For all π be such that $\pi \models K_0$, for all symbolic run of $\mathcal{A}(K)$ reaching (q, C), there exists a clock valuation w such that $<w, \pi> \models C$.*

Proof. For all symbolic run of $\mathcal{A}(K)$ reaching (q, C), we have $(q, C) \in S$ since $S = Post^*_{A(K)}(\{s_0\})$. Moreover, we have $K_0 = \bigcap_{(q,C)\in S}(\exists X : C)$. Thus, for all $\pi \models K_0$, for all $(q, C) \in S$, we have $\pi \models (\exists X : C)$. Hence, there exists a clock valuation w such that $<w, \pi> \models C$. □

Proposition 18. *For all symbolic run of $\mathcal{A}(K)$ reaching (q, C), for all parameter valuation π and clock valuation w such that $<w, \pi> \models C$, there exists an equivalent concrete run of $\mathcal{A}[\pi]$ reaching (q, w).*

Proof. The proof of Proposition 3.17 in [17] can be adapted in a straightforward manner. □

Proposition 19. *For all $\pi \models K_0$, for all symbolic run of $\mathcal{A}(K)$, there exists an equivalent concrete run of $\mathcal{A}[\pi]$.*

Proof. From Prop. 17 and Prop. 18. □

Conversely, we now state that, for each concrete run of $\mathcal{A}[\pi]$, we can find an equivalent symbolic run of $\mathcal{A}(K)$.

Proposition 20. *For all $\pi \models K_0$, for all concrete run of $\mathcal{A}[\pi]$, there exists an equivalent symbolic run of $\mathcal{A}(K)$.*

Proof. The proof of Proposition 3.18 in [17] can be adapted in a straightforward manner to show that, for all $\pi \models K$, for all concrete run of $\mathcal{A}[\pi]$, there exists an equivalent symbolic run of $\mathcal{A}(K)$. The results follows from the fact that $\pi \models K_0$ implies $\pi \models K$ (by Prop. 15). □

Proposition 21. *For all $\pi \models K_0$, the sets of runs of $\mathcal{A}(K)$ and $\mathcal{A}[\pi]$ are equivalent, i.e., the sets of traces are equal.*

Proof. From Prop. 19 and 20. □

The following theorem states that *InverseMethod* solves our inverse problem as defined in Sect. 2.4.

Theorem 22. *Suppose that InverseMethod$(\mathcal{A}, \pi_0)$ terminates with output K_0. Then:*

(1) $\pi_0 \models K_0$, and
(2) for all $\pi \models K_0$, the sets of concrete runs of $\mathcal{A}[\pi_0]$ and $\mathcal{A}[\pi]$ are equivalent, i.e., the sets of traces are equal.

Proof. From Prop. 16 and Prop. 21. □

3.4. *Termination*

Reachability analysis is known to be undecidable in the framework of PTAs [2, 12], and computations performed with tools on PTAs (such as HyTech [15]) do not always terminate. However, we give a sufficient condition for ensuring termination of our method.

Proposition 23. *Let $\mathcal{A}$ be a PTA and π_0 be a valuation of P. If there exists $n \in \mathbb{N}$ s.t. $Post^n_{\mathcal{A}[\pi_0]}(\{s_0\}) = \emptyset$, then algorithm InverseMethod terminates.*

Proof. Let us first consider the inner *DO* loop for a given i. At each iteration, we select a state $s = (q, C)$ of $Post^i_{\mathcal{A}(K)}(\{s_0\})$. We select an inequality J in $\exists X : C$, negate J, and add it to K. Hence, s does not belong to $Post^i_{\mathcal{A}(K \wedge \neg J)}(\{s_0\})$. The traces of $\mathcal{A}(K)$ of length $j \leq i$ can be organized under the form a finite tree, say T. Likewise, the traces of $\mathcal{A}(K \wedge \neg J)$ of length $j \leq i$ can be organized under the form a finite tree, say T'. It is easy to show that T' is a subtree of T, i.e., each branch starting from the root of T' is a (sub)branch starting from the (same) root in T. Thus no new state can be reached in $\mathcal{A}(K \wedge \neg J)$. Moreover, the branch of T reaching the location corresponding to s does not belong to T'. So, the number of nodes of T' is less than the number of nodes of T. Hence, the number of states of

$Post^i_{\mathcal{A}(K \wedge \neg J)}(\{s_0\})$ is less than the number of states of $Post^i_{\mathcal{A}(K)}(\{s_0\})$. Thus, the inner DO loop terminates.

Let us now consider the outer DO loop. Since $Post^n_{\mathcal{A}[\pi_0]}(\{s_0\}) = \emptyset$, the symbolic runs of $\mathcal{A}[\pi_0]$ have at most length $n-1$. Let us show by *reductio ad absurdum* that the outer DO loop terminates at iteration $i \leq n-1$. Suppose that we are still in the outer DO loop at $i = n$. Thus, there exists a symbolic run of $\mathcal{A}(K)$ of length n of the form $(q_0, C_0) \overset{a_0}{\Rightarrow} \cdots \overset{a_{n-2}}{\Rightarrow} (q_{n-1}, C_{n-1}) \overset{a_{n-1}}{\Rightarrow} (q_n, C_n)$. Moreover, since all the states in $S = Post^i_{\mathcal{A}(K)}(\{s_0\})$ are π_0-compatible, we have $\pi_0 \models C_i$, for $0 \leq i \leq n$. Hence, there exists w s.t. $<w, \pi_0> \models C_n$. Therefore, by Prop. 18, there exists an equivalent concrete run of length n of $\mathcal{A}[\pi_0]$ reaching (q_n, w). It follows from Prop. 3.18 of [17] that there exists an equivalent symbolic run of length n of $\mathcal{A}[\pi_0]$, which contradicts the assumption $Post^n_{\mathcal{A}[\pi_0]}(\{s_0\}) = \emptyset$. □

A sufficient condition so that there exists $n \in \mathbb{N}$ s.t. $Post^n_{\mathcal{A}[\pi_0]}(\{s_0\}) = \emptyset$ is that the oriented graph depicting the traces associated the symbolic runs of $\mathcal{A}[\pi_0]$ is acyclic, i.e., traces never pass twice by the same location. This is for example the case of the traces depicted on Fig. 3.

A sufficient condition for the acyclicity of the graph of traces is that the oriented graph depicting the PTA $\mathcal{A}$ be itself acyclic. This is generally the case for synchronous circuits analyzed over a fixed number (typically, 1 or 2) of clock cycles.

3.5. *Application to the Flip-flop Example*

Let us apply our algorithm *InverseMethod* (implemented in the program IMITATOR, see Sect. 5) to the NPTA modeling the flip-flop circuit and to the instantiation π_0 of the parameters given in Sect. 2.4. The program generates the following constraint K_0 in 2 seconds[b]:

$$\begin{array}{lccccc} & T_{setup} < T_{LO} & \wedge & \delta_3^+ + \delta_4^+ < T_{HI} & \wedge & \delta_1^+ < T_{setup} \\ \wedge & T_{hold} \leq \delta_3^+ + \delta_4^+ & \wedge & \delta_3^- + \delta_4^- \leq T_{hold} & \wedge & \delta_3^+ < T_{hold} \\ \wedge & \delta_1^- > 0 & & & & \end{array}$$

Besides, by construction on the environment (signals D, CK and Q), recall that we have the implicit additional constraint: $T_{hold} \leq T_{HI}$.

As formally stated in Theorem 22, one can check that the set of traces coincides with the one depicted on Fig. 3.

In [9], a constraint Z is generated in order to prevent bad system behaviors. The bad state is defined as the case where $CK^{\downarrow}$ occurs before $Q^{\uparrow}$. This constraint Z is the following:

$$\begin{array}{lccc} & T_{setup} > \delta_1^+ + \delta_2^+ - \delta_2^- & \wedge & T_{hold} > \delta_2^+ + \delta_3^+ \\ \wedge & T_{HI} > \delta_2^+ + \delta_3^+ + \delta_4^+ & \wedge & T_{HI} > T_{hold} \\ \wedge & T_{LO} > T_{setup} & \wedge & \delta_1^- > \delta_2^+ \end{array}$$

[b]It can be surprising that neither δ_2^- nor δ_2^+ appear in K_0. This constraint K_0 actually prevents G_2 from any change, as g_1 and CK are never both set to 1; therefore, g_2 always remains set to 1, and the delay of G_2 does not have any influence on the system for the considered environment.

Note that, as we have $\pi_0 \models Z$, the set of traces under π_0 (and by construction under K_0) also prevents bad system behaviors[c]. It is easy to check that our constraint K_0 is *uncomparable* with Z, i.e., we can find instantiations satisfying K_0 and not Z, and vice versa. This suggests to extend K_0 by applying *InverseMethod* to a new instantiation π_1 such that $\pi_1 \models Z$ and $\pi_1 \not\models K_0$. This will be the subject of Sect. 6.

4. Extension to the Cyclic Case

We have seen in Section 3.4 that the algorithm *InverseMethod* terminates in the "acyclic" case. The algorithm does not terminate in the cyclic case in general. We first present a slightly modified version of our algorithm, which terminates in some special cases of the cyclic class while still preserving Theorem 22. Then we present a further extension which terminates more often for cyclic cases, at the price of weakening the equivalence result of Theorem 22.

4.1. *First Extension*

We state here that a slight generalization of the algorithm *InverseMethod* is also correct. One slightly modifies the algorithm by replacing the termination test

$$\textbf{IF } Post_{\mathcal{A}(K)}(S) = \emptyset$$

with the expression

$$\textbf{IF } \forall s \in Post_{\mathcal{A}(K)}(S), \exists s' \in S : s = s'.$$

Thus, the algorithm now terminates as soon as every new reachable state has already been (exactly) produced before. We denote by *InverseMethod*$'$ this modified algorithm.

Remark. Note that our modified **IF** condition is still *more restricted* than the subsumption test performed by HyTech. Indeed, HyTech stops computing reachable states when $\forall s \in Post_{\mathcal{A}(K)}(S), \exists s' \in S : s \subseteq s'$.

The following theorem states the correction of this modified algorithm.

Theorem 24. *Suppose that InverseMethod*$'(\mathcal{A}, \pi_0)$ *terminates with output* K_0'. *Then :*

(1) $\pi_0 \models K_0'$, *and*
(2) for all $\pi \models K_0'$, *the sets of concrete runs of* $\mathcal{A}[\pi_0]$ *and* $\mathcal{A}[\pi]$ *are equivalent, i.e., the sets of traces are equal.*

[c]Our instantiation π_0 was actually chosen in order to satisfy Z.

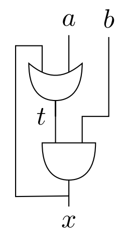

Fig. 5. And–Or Component

Proof. Suppose that $\mathit{InverseMethod}'(\mathcal{A}, \pi_0)$ terminates with output K'_0. Let K' (resp. S') be the current constraint on the parameters (resp. the current set of reachable states) when the algorithm $\mathit{InverseMethod}'$ terminates. We have $S' = \mathit{Post}^*_{A(K')}(\{s_0\})$ and $K'_0 = \bigcap_{(q,C)\in S'}(\exists X : C)$. Furthermore any state (q, C) reached by a symbolic run of $\mathcal{A}(K')$ is such that $\pi_0 \models (\exists X : C)$, because this state (or one identical previously generated) has passed the π_0-compatibility test. It follows that all the properties and Theorem 22 of Sect. 3.3 still hold, and can be proved similarly (replacing S, K, K_0 with S', K', K'_0 respectively). □

And–Or Example. We consider an "And–Or" circuit described in [8] and depicted on Fig. 5. It is composed of 2 gates (one "And" gate and one "Or" gate) which are interconnected in a cyclic way. Each rising (resp. falling) edge of signal a, is denoted by $a^\uparrow$ (resp. $a^\downarrow$), and similarly for b, t, x. The delay between the rising and the falling edge of $a^\uparrow$ (resp. $a^\downarrow$) and $a^\downarrow$ (resp. $a^\uparrow$) is in $[\delta^-_{a^\uparrow}, \delta^+_{a^\uparrow}]$ (resp. $[\delta^-_{a^\downarrow}, \delta^+_{a^\downarrow}]$), and similarly for b. The traversal of the gate Or takes also a delay in $[\delta^-_{Or}, \delta^+_{Or}]$, and likewise for gate And. There are 12 timing parameters. We consider the following instantiation π_0 of the parameters:

$$\begin{array}{llll}
\delta^-_{a^\uparrow} = 13 & \delta^+_{a^\uparrow} = 14 & \delta^-_{a^\downarrow} = 16 & \delta^+_{a^\downarrow} = 18 \\
\delta^-_{b^\uparrow} = 7 & \delta^+_{b^\uparrow} = 8 & \delta^-_{b^\downarrow} = 19 & \delta^+_{b^\downarrow} = 20 \\
\delta^-_{And} = 3 & \delta^+_{And} = 4 & \delta^-_{Or} = 1 & \delta^+_{Or} = 2
\end{array}$$

We consider an environment starting at location q_0 with $a = b = x = t = 1$, and the following repeated cycle of alternating rising and falling edges of a and b: $b^\downarrow, a^\downarrow, b^\uparrow, a^\uparrow$. Under this environment and the reference valuation π_0, the set of traces, expressed under the compact form of a reachability graph, is given on Fig. 6.

Applying algorithm $\mathit{InverseMethod}'$ (implemented in the program IMITATOR), the following constraint is computed[d]:

$$\begin{array}{rrrr}
 & 0 < \delta^-_{a^\downarrow} & \wedge & 0 < \delta^-_{And} \\
\wedge & 0 < \delta^-_{Or} & \wedge & \delta^+_{And} + \delta^+_{b^\uparrow} < \delta^-_{a^\uparrow} \\
\wedge & \delta^+_{a^\uparrow} + \delta^+_{Or} < \delta^-_{b^\downarrow} + \delta^-_{b^\uparrow} & \wedge & \delta^-_{b^\downarrow} + \delta^-_{b^\uparrow} \leq \delta^+_{a^\uparrow} + \delta^+_{a^\downarrow} \\
\wedge & \delta^+_{Or} + \delta^+_{And} < \delta^-_{b^\uparrow} & &
\end{array}$$

[d]In [8], the generated constraint is not explicitly given.

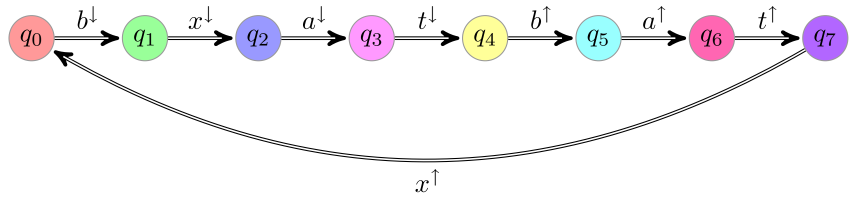

Fig. 6. Traces of the and–or circuit under π_0

The set of traces is guaranteed to be the same as the one depicted on Fig. 6.

4.2. *Second Extension*

In order to make the method terminate more often, we relax the termination condition in Algorithm *InverseMethod*$'$ by replacing the expression

$$\textbf{IF } \forall s \in Post_{\mathcal{A}(K)}(S), \exists s' \in S : s = s'$$

with the more general expression

$$\textbf{IF } \forall s \in Post_{\mathcal{A}(K)}(S), \exists s' \in S : s \subseteq s'.$$

Note that this termination test actually corresponds to the subsumption test used in HyTech. Let *InverseMethod*$''$ denote this modified algorithm.

Suppose that *InverseMethod*$''(\mathcal{A}, \pi_0)$ terminates with output K''_0. Let K'' (resp. S'') be the current constraint on the parameters (resp. the current set of reachable states) when the algorithm *InverseMethod*$''$ terminates. We have $S'' = Post^*_{\mathcal{A}(K'')}(\{s_0\})$ and $K''_0 = \bigcap_{(q,C) \in S''}(\exists X : C)$. However, a state (q, C) reached by a symbolic run of $\mathcal{A}(K'')$ is now in general *strictly* included into some state previously generated. Therefore it may lose the π_0-compatibility property. Thus, the counterparts of Prop. 17 and 19 do not hold any longer. Nevertheless, it is easy to see that the sets of *locations* reached by symbolic runs are identical to those reached by concrete runs.

Let $Loc^*(\mathcal{A}[\pi])$ denote the set of reachable locations of $\mathcal{A}[\pi]$, i.e., $\{q \mid \exists C : (q, C) \in Post^*_{\mathcal{A}[\pi]}(\{s_0\})\}$. We have:

Theorem 25. *Suppose that InverseMethod*$''(\mathcal{A}, \pi_0)$ *terminates with output* K''_0*. Then*

(1) $\pi_0 \models K''_0$*, and*
(2) for all $\pi \models K''_0$, $Loc^*(\mathcal{A}[\pi]) = Loc^*(\mathcal{A}[\pi_0])$*,*

Note that Theorem 25 still holds with K'' instead of K''_0, i.e., $\pi_0 \models K''$, and $Loc^*(\mathcal{A}[\pi]) = Loc^*(\mathcal{A}[\pi_0])$, for any $\pi \models K''$.

Example	# of PTAs	loc. per PTA	# of clocks	# of param.	# of iter.	$\|Post^*\|$	$\|K_0\|$	CPU time
Flip-flop [9]	5	$[4,16]$	5	12	8	11	6	2 s
And–or [8]	3	$[4,8]$	4	14	13	14	13	3 s
RCP [22]	5	$[6,11]$	6	5	18	154	2	70 s
CSMA/CD [19, 25]	3	$[6,7]$	4	3	21	294	3	108 s
SPSMALL [7]	10	$[3,8]$	10	22	31	31	23	78 min
SIMOP [3]	5	$[6,16]$	9	16	51	848	7	419 min

Fig. 7. Case studies using IMITATOR

Remark. The second item of Theorem 25 is interesting when $\mathcal{A}[\pi_0]$ is known to avoid a given *bad* location because, in this case, $\mathcal{A}[\pi]$ is also guaranteed to avoid this bad location, for any $\pi \models K_0''$. This is the case of the SIMOP case study (see Sect. 5 below).

5. Experiments

The algorithm *InverseMethod*, as well as its two variants, has been implemented under the form of a program named IMITATOR [5] (standing for *Inverse Method for Inferring Time AbstracT behaviOR*). This program, containing about 1500 lines of code, is written in Python and calls the parametric model checker HYTECH [15] in order to compute the *Post* operation. The selection of a π_0-compatible state and a π_0-incompatible inequality J is done in a random manner.

Our tool IMITATOR was experimented in the framework of French ANR project VALMEM, for synthesizing timing constraints on memory circuits designed by ST-Microelectronics. It successfully treated in 78 minutes a portion of the asynchronous circuit SPSMALL [7] containing 9 gates, modeled by 10 PTA using 10 clocks and 22 parameters. We also successfully applied IMITATOR to various case studies to infer constraints on parameters. One reason for which it behaves well in practice is that the procedure quickly reduces the number of reachable states, by drastically restraining the current constraint K.

A table of results is presented in Fig. 7. We give from left to right the name of the example, the number of PTAs composing the global system $\mathcal{A}$, the lower and upper bounds on the number of locations per PTA, the number of clocks and parameters of $\mathcal{A}$, the number of iterations of the algorithm, the number of reachable symbolic states, the number of inequalities of K_0 (put under a simplified form via HYTECH), and the computation time on an Intel Quad Core 3 GHz with 3.2 Gb. The fully detailed experiments are available in [4].

We used Algorithm *InverseMethod* for the flip-flop, CSMA/CD, RCP[e], and SPSMALL examples. We used the first extension *InverseMethod*$'$ for the And–or exam-

[e] We considered an acyclic model for CSMA/CD and RCP by bounding the maximal number of collisions of messages.

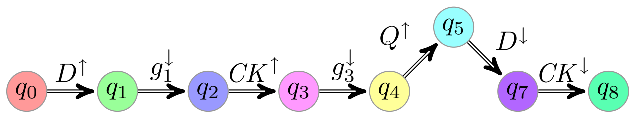

Fig. 8. Traces of the flip-flop circuit under π_1

ple, and the second extension *InverseMethod''* for the SIMOP case study.

The instantiation π_0 we use for each example is either an instance satisfying the constraint Z generated by classical constraint synthesis methods when such a Z is available (e.g., in the flip-flop, RCP examples), or corresponds to typical data given with the case study (e.g., in the CSMA/CD, SIMOP, VALMEM examples). In the first case, the constraint generated by our method is often the same as Z, but not always : for example, in the flip-flop example, the inferred constraint K_0 is different from the constraint Z originating from [9]. This suggests to combine both information (see Sect. 6). In the second case, the constraint allows us to safely decrease (or increase) some components of the typical data π_0, as long as they still satisfy K_0. This is useful, for example in order to relax some requirements on the environment of asynchronous circuits (see, e.g., [7]).

6. Extension : Incremental Method

We presented in Sect. 3 how to synthesize a constraint from a given instantiation of the parameters. We now aim at widening this constraint in order to cover a predefined zone Z.

The method we now present can be summarized as follows : we pick up an instantiation π_1 in Z, we find a constraint with *InverseMethod*, and we iterate this operation until Z is completely covered. Let us explain this on the flip-flop example.

Considering again our flip-flop example described in Sect. 2.4, we aim at widening the constraint K_0 found in Sect. 3.5 from the instantiation π_0. Recall that π_0 satisfies the constraint Z. Since $Z \not\subseteq K_0$, there exists a different instantiation of the parameters satisfying constraint Z, and not K_0. For example, consider the instantiation π_1 defined as follows (the differences with π_0 are in bold) :

$$\begin{array}{cccc}
T_{HI} = \mathbf{23} & T_{LO} = 15 & T_{setup} = 10 & T_{hold} = 17 \\
\delta_1^- = 7 & \delta_1^+ = 7 & \delta_2^- = 5 & \delta_2^+ = 6 \\
\delta_3^- = 8 & \delta_3^+ = 10 & \delta_4^- = 3 & \delta_4^+ = \mathbf{5}
\end{array}$$

The set of traces of the system under π_1 and the environment considered in Sect. 2.4 is depicted in Fig. 8, under the form of an oriented graph (actually reduced to a single path). Note that this graph is a subgraph of the set of traces of the system under π_0 (Fig. 3).

Applied to the NPTA modeling the flip-flop circuit, our program generates the following constraint K_1 :

$$\begin{array}{rcl}
 & T_{setup} < T_{LO} & \wedge \quad T_{hold} \leq T_{HI} \\
\wedge & \delta_3^+ + \delta_4^+ < T_{hold} & \wedge \quad \delta_1^+ < T_{setup}
\end{array}$$

ALGORITHM ***IIM*** $(\mathcal{A}, Z)$

Input $\mathcal{A}$: PTA
Z : Constraint on the parameters
Output $\mathcal{D}$: Disjunction of constraints on the parameters
Variable π : Valuation of the parameters

DO UNTIL $\mathcal{D} \supseteq Z$
Select π with $\pi \models Z \wedge \pi \not\models \mathcal{D}$
$\mathcal{D} \ := \ \mathcal{D} \cup \mathit{InverseMethod}(\mathcal{A}, \pi)$
OD
RETURN $\mathcal{D}$

Fig. 9. Algorithm *IIM*

It is easy to see now that $Z \subsetneq K_0 \cup K_1$. In other terms, our union of constraints $K_0 \cup K_1$ represents a strictly bigger set of parameter valuations than Z. Note that, although $Z \subsetneq K_0 \cup K_1$, the sets of time-abstract behaviors under Z and $K_0 \cup K_1$ are the same ($K_0 \cup K_1$ does not add any trace to Z).

We now present this method more formally under an algorithmic form. Algorithm *IIM* (standing for *Incremental Inverse Method*), described on Fig. 9, outputs a disjunction $\mathcal{D}$ of constraints K_0, K_1, etc. The notation $\pi \not\models \mathcal{D}$ means $\pi \not\models K_i$, for all $i = 0, 1, \dots$

7. Final Remarks

We presented an algorithm *InverseMethod* which synthesizes a constraint K_0 under which the studied PTA yields the same set of traces as under a reference parameter valuation π_0. Note that, although the final constraint K_0 induces a behavioral property of the system related to traces, only states (and not traces) are manipulated by the algorithm. The method is guaranteed to terminate in the acyclic case. For the cyclic case, we adapt the method in order to improve termination at the price of getting an overapproximation of the set of traces.

Our method is complementary to the CEGAR-based methods that assume given a set of bad states to be avoided. We showed how these two kinds of methods can be combined together, using algorithm *IIM*. Note that the principles of our method can be applied to other models than PTA. A preliminary study was already done in the framework of Timed Graphs of Events [13].

The algorithm *InverseMethod* has been presented in a naive way for clarity reasons, and could be optimized in several ways. For example, the statement $S := \bigcup_{j=0}^{i} \mathit{Post}^{j}_{\mathcal{A}(K)}(\{s_0\})$ in the inner *DO* loop can be removed as follows. For a given iteration i, this statement can be repeated in the worst case (considering that all states are π_0-incompatible) as many times as the number of states in $\mathit{Post}^{i}_{\mathcal{A}(K)}$. Actually, we can simply update all the states $s = (q, C)$ of S by adding $\neg J$ to C, which has a much smaller complexity. One could also redefine the *Post* operator so that it only computes the π_0-compatible states. In that case, it is enough to compute the $Post^*$ and return the intersection of the constraints associated to all the reachable states.

It can be shown that *InverseMethod* is (in general) non-confluent, i.e., several applications of *InverseMethod* to the same instance π_0 may lead to a different K_0. It follows from this remark that the generated constraint K_0 is not *maximal*, i.e., there may exist $\pi \not\models K_0$ such that the traces of $\mathcal{A}[\pi_0]$ and the traces of $\mathcal{A}[\pi]$ are identical. In practice, we observe on all the experiments of Sect. 5 a confluent behavior of the algorithm: applications of *InverseMethod* to the same instance π_0 generally lead to the same constraint K_0, whatever the random selections are. However, the constraint generated is not maximal. It would be interesting to evaluate how large the constraint generated by *InverseMethod* is.

Acknowledgment.

We are grateful to Laurent Doyen for a helpful discussion, and we thank Jeremy Sproston and the anonymous referees for helpful comments.

References

[1] R. Alur and D. L. Dill. A theory of timed automata. *TCS*, 126(2):183–235, 1994.
[2] R. Alur, T. A. Henzinger, and M. Y. Vardi. Parametric real-time reasoning. In *STOC '93*, pages 592–601, New York, USA, 1993. ACM.
[3] S. Amari, É. André, T. Chatain, O. De Smet, B. Denis, E. Encrenaz, L. Fribourg, and S. Ruel. Timed analysis of distributed control systems combining simulation and parametric model checking. Research report, Laboratoire Spécification et Vérification, ENS Cachan, France, June 2009.
[4] É. André, E. Encrenaz, and L. Fribourg. Synthesizing parametric constraints on various case studies using Imitator. Research report, Laboratoire Spécification et Vérification, ENS Cachan, France, June 2009.
[5] Étienne André. Imitator: a tool for synthesizing constraints on timing bounds of timed automata. In *ICTAC'09*, LNCS. Springer, August 2009. To appear.
[6] A. Annichini, E. Asarin, and A. Bouajjani. Symbolic techniques for parametric reasoning about counter and clock systems. In *CAV '00*, pages 419–434. Springer-Verlag, 2000.
[7] R. Chevallier, E. Encrenaz, L. Fribourg, and W. Xu. Timed verification of the generic architecture of a memory circuit using parametric timed automata. *Formal Methods in System Design*, 34(1):59–81, February 2009.
[8] R. Clarisó and J. Cortadella. Verification of concurrent systems with parametric delays using octahedra. In *ACSD '05*. IEEE Computer Society, 2005.
[9] R. Clarisó and J. Cortadella. The octahedron abstract domain. *Sci. Comput. Program.*, 64(1):115–139, 2007.
[10] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *CAV '00*, pages 154–169. Springer-Verlag, 2000.
[11] A. Collomb–Annichini and M. Sighireanu. Parameterized reachability analysis of the IEEE 1394 Root Contention Protocol using TReX. In *RT-TOOLS '01*, 2001.

[12] Laurent Doyen. Robust parametric reachability for timed automata. *Information Processing Letters*, 102(5):208–213, 2007.

[13] E. Encrenaz and L. Fribourg. Time separation of events: An inverse method. In *Proceedings of the LIX Colloquium '06*, volume 209 of *ENTCS*, Palaiseau, France, 2008. Elsevier Science Publishers.

[14] G. Frehse, S.K. Jha, and B.H. Krogh. A counterexample-guided approach to parameter synthesis for linear hybrid automata. In *HSCC '08*, volume 4981 of *LNCS*, pages 187–200. Springer, 2008.

[15] T. A. Henzinger, P. Ho, and H. Wong-Toi. A user guide to HyTech. In *TACAS*, pages 41–71, 1995.

[16] T.A. Henzinger and H. Wong-Toi. Using HyTech to synthesize control parameters for a steam boiler. In *Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control, LNCS 1165*, pages 265–282, London, UK, 1996. Springer-Verlag.

[17] T.S. Hune, J.M.T. Romijn, M.I.A. Stoelinga, and F.W. Vaandrager. Linear parametric model checking of timed automata. *Journal of Logic and Algebraic Programming*, 2002.

[18] K. G. Larsen, P. Pettersson, and W. Yi. Uppaal in a nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1-2):134–152, 1997.

[19] X. Nicollin, J. Sifakis, and S. Yovine. Compiling real-time specifications into extended automata. *IEEE Trans. on Software Engineering*, 18:794–804, 1992.

[20] R. Ben Salah, M. Bozga, and O. Maler. On interleaving in timed automata. In *CONCUR '06*, volume 4137 of *LNCS*, pages 465–476. Springer, 2006.

[21] Alexander Schrijver. *Theory of linear and integer programming.* John Wiley & Sons, Inc., New York, NY, USA, 1986.

[22] D. Simons and M. Stoelinga. Mechanical verification of the IEEE 1394a Root Contention Protocol using Uppaal2k. *International Journal on Software Tools for Technology Transfer*, 3(4):469–485, 2001.

[23] S. Tripakis and S. Yovine. Analysis of timed systems using time-abstracting bisimulations. *Form. Methods Syst. Des.*, 18(1):25–68, 2001.

[24] F. Wang and H.C. Yen. Timing parameter characterization of real-time systems. In *CIAA '03*, volume 2759 of *LNCS*, pages 23–34, 2003.

[25] Farn Wang. Symbolic parametric safety analysis of linear hybrid systems with BDD-like data-structures. *IEEE Trans. Softw. Eng.*, 31(1):38–51, 2005.

[26] T. Yoneda, T. Kitai, and C. J. Myers. Automatic derivation of timing constraints by failure analysis. In *CAV '02*, pages 195–208. Springer-Verlag, 2002.