Étienne André,
Emmanuelle Encrenaz,
Laurent Fribourg

Synthesizing Parametric Constraints on Various Case Studies Using IMITATOR

Research Report LSV-09-13

June 2009

Synthesizing Parametric Constraints on Various Case Studies Using IMITATOR

Étienne André, Laurent Fribourg[1]

LSV – ENS de Cachan & CNRS, France

Emmanuelle Encrenaz[2]

LIP6 – Université Pierre et Marie Curie & CNRS, France

Abstract

We present here applications of Imitator, a tool for synthesizing constraints on timing bounds (seen as parameters) in the framework of timed automata. Unlike classical synthesis methods, we take advantage of a given reference valuation of the parameters for which the system is known to behave properly. Our aim is to generate a constraint such that, under any valuation satisfying this constraint, the system is guaranteed to behave, in terms of alternating sequences of locations and actions, as under the reference valuation. This is useful for safely relaxing some values of the reference valuation, and optimizing timing bounds of the system. We have successfully applied our tool to various examples of asynchronous circuits and protocols, which are detailed in this report.

Keywords: Parametric timed automata, optimization of timing delays, asynchronous circuits, protocols.

1 Introduction

Timed automata [1] 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 timing bounds. However, the behavior of a system 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* (i.e., a conjunction of linear inequalities) on these parameters which will guarantee a correct behavior of the system. Such automata are called *parametric timed automata* (PTA) [2,14].

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

[1] Email: {andre,fribourg}@lsv.ens-cachan.fr

[2] Email: emmanuelle.encrenaz@lip6.fr

The synthesis of constraints for PTAs has been mainly done by supposing given a set of “bad states” (see, e.g., [11,12]). The goal is to find a set of parameters for which the considered timed automaton does not reach any of these bad states. We call such a method a *bad-state oriented* method. By contrast, we present in this paper a tool based on a *good-state oriented* method.

The tool Imitator [4], standing for *Inverse Method for Inferring Time AbstracT behaviOR*, implements the algorithm InverseMethod, described in [5]. We assume given a system modeled by a PTA $\mathcal{A}$. Whereas bad-state oriented methods consider a set of bad states, Imitator considers an initial tuple π_0 of values for the parameters, under which the system is known to behave properly. When the parameters are instantiated with π_0, the system is denoted by $\mathcal{A}[\pi_0]$. Under certain conditions, the algorithm InverseMethod generalizes this *good* behavior by computing a constraint K_0 which guarantees that, under any parameter valuation π satisfying K_0, the system behaves in the same manner: the behaviors of the timed automata $\mathcal{A}[\pi]$ and $\mathcal{A}[\pi_0]$ are *(time-abstract) equivalent*, i.e., the traces of execution viewed as alternating sequences of *locations* (or “control states”) and actions are identical. This is written $\mathcal{A}[\pi] \equiv_{TA} \mathcal{A}[\pi_0]$. More formally, the algorithm InverseMethod solves the following *inverse problem* [5] for *acyclic* systems (i.e., with only finite traces) by computing a constraint K_0 such that:

(i) $\pi_0 \models K_0$,

(ii) $\mathcal{A}[\pi] \equiv_{TA} \mathcal{A}[\pi_0]$, for any $\pi \models K_0$.

A trivial solution is $K_0 = \{\pi_0\}$. However, Imitator will always generate something more general than $K_0 = \{\pi_0\}$, under the form of a conjunction of inequalities on the parameters (without any constant, apart from 0).

A practical application is to *optimize* (either decrease or increase) the value of some element of π_0, as long as it still satisfies K_0. This is of particular interest in the framework of digital circuits, in order to safely minimize some stabilization timings (typically “setup” or “hold”). For example, one can minimize some local stabilization timings, without changing the global delay for writing an input signal in a memory circuit.

We shortly recall the algorithm InverseMethod and its implementation Imitator in Sect. 2. We then present various case studies from the literature: two circuits, i.e., a flip-flop circuit (Sect. 3), and an “And–Or” circuit (Sect. 4), and four protocols, i.e., the root contention protocol (Sect. 5), the CSMA/CD protocol (Sect. 6), the bounded retransmission protocol (Sect. 7), and the conformance protocol (Sect. 8). We then present two real case studies: a portion of the memory circuit SPSMALL designed by ST-Microelectronics (Sect. 9), and a distributed control system (Sect. 10). We finally summarize the experiments in Sect. 11 and give some final remarks in Sect. 12.

2 The Algorithm InverseMethod

We use in this section the same formalism as in [5].

```
ALGORITHM InverseMethod($\mathcal{A}$, $\pi_0$)
Input      $\mathcal{A}$  : PTA
           $\pi_0$ : 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

 $i := 0$ ;  $K := True$ ;  $S := \{s_0\}$
 DO
     DO UNTIL $S$ is $\pi_0$-compatible
         Select a $\pi_0$-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
                     %% $S$ $\pi_0$-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. 1. Algorithm InverseMethod

2.1 Description

The algorithm InverseMethod on which Imitator relies can be summarized as follows. Starting with $K := True$, we iteratively compute a growing set of reachable symbolic states. A symbolic state of the system is a couple (q, C), where q is a location of the PTA, and C a constraint on the parameters[3]. When a π_0-*incompatible* state (q, C) is encountered (i.e., when $\pi_0 \not\models C$), K is refined as follows: a π_0-incompatible inequality J (i.e., such that $\pi_0 \not\models J$) is selected within C, and $\neg J$ is added to K. The procedure is then started again with this new K, and so on, until the whole set of reachable states ($Post^*$) is computed.

The algorithm InverseMethod is given in Fig. 1, where the clock variables have been disregarded for the sake of simplicity. We denote by $Post^i_{\mathcal{A}(K)}(S)$ the set of symbolic states reachable from S in exactly i steps of $\mathcal{A}(K)$, and $\exists X : C$ denotes the elimination of clock variables in constraint C.

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 $\pi_0 \models \neg J$).

2.2 Implementation

Our algorithm InverseMethod has been implemented under the form of a program named Imitator [4] (standing for *Inverse Method for Inferring Time AbstracT*

[3] Strictly speaking, C is a constraint on the clock variables and the parameters, but the clock variables are omitted here for the sake of simplicity. See [5] for more details.

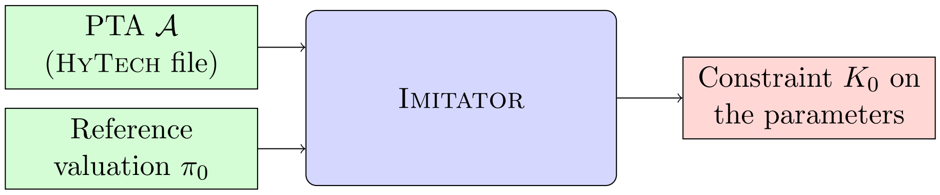
PTA $\mathcal{A}$ (HyTech file)
Reference valuation π_0
Imitator
Constraint K_0 on the parameters

Fig. 2. Imitator inputs and output

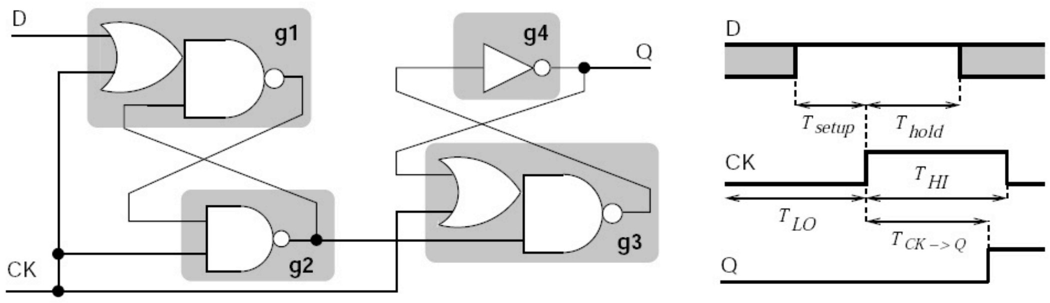

Fig. 3. Flip-flop circuit

behaviOR). As depicted on Fig. 2, Imitator takes as inputs a PTA described in HyTech syntax, and a reference valuation π_0. The aim of the program is to output a constraint K_0 on the parameters solving the inverse problem. This program, containing about 1500 lines of code, is written in Python and calls parametric model checker HyTech [13] 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 [8] 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.

All experiments were conduced on an Intel Quad Core 3 GHz with 3.2 Gb. The source code of all the examples is available on Imitator's webpage[4].

3 Flip-flop Circuit

3.1 Description

We consider here an asynchronous "D flip-flop" circuit described in [10] and depicted on Fig. 3. 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 model the environment. The output signal of a gate G_i is named g_i (note that

[4] http://www.lsv.ens-cachan.fr/~andre/IMITATOR/

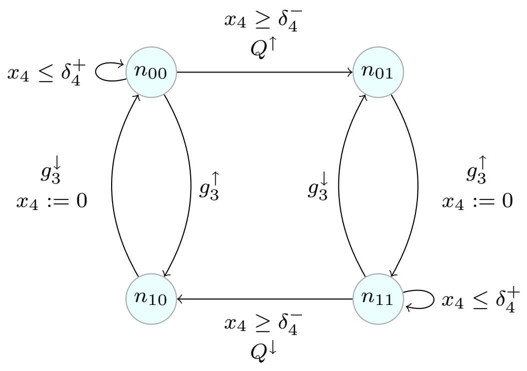

Fig. 4. Parametric Timed Automaton modeling gate G_4

$Q = g_4$). 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 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. 3. Therefore, we have the implicit constraint $T_{setup} \leq T_{LO} \wedge T_{hold} \leq T_{HI}$.

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 PTA $\mathcal{A}$ modeling the system results from the composition[5] of those 5 PTA. As for an example, we give on Fig. 4 the PTA modeling the gate G_4 (which is actually a "Not" gate). This PTA contains four locations $n_{00}, n_{01}, n_{10}, n_{11}$, where n_{ij} stands for a state of the gate where the input g_3 is equal to i and the output Q is equal to j. This PTA contains one clock x_4 and, as explained before, contains two parameters δ_4^- and δ_4^+ representing the interval of time between a change of the input and the change of the output.

3.2 Clarisó and Cortadella's Constraints

In [10], 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:

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

3.3 First Instantiation Point

We first consider the following instantiation π_0 of the parameters, which was chosen so that $\pi_0 \models Z$:

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

[5] It can be shown that the standard parallel composition of several PTAs is a PTA.

$$\begin{tikzcd}
 & & & & & q_5 \arrow[dr, "D^{\downarrow}"] & & \\
q_0 \arrow[r, "D^{\uparrow}"] & q_1 \arrow[r, "g_1^{\downarrow}"] & q_2 \arrow[r, "CK^{\uparrow}"] & q_3 \arrow[r, "g_3^{\downarrow}"] & q_4 \arrow[ur, "Q^{\uparrow}"] \arrow[dr, "D^{\downarrow}"'] & & q_7 \arrow[r, "CK^{\downarrow}"] & q_8 \\
 & & & & & q_6 \arrow[ur, "Q^{\uparrow}"'] & &
\end{tikzcd}$$

Fig. 5. Traces of the flip-flop circuit under π_0

Location	D	CK	g_1	g_2	g_3	Q
q_0	0	0	1	1	1	0
q_1	1	0	1	1	1	0
q_2	1	0	0	1	1	0
q_3	1	1	0	1	1	0
q_4	1	1	0	1	0	0
q_5	1	1	0	1	0	1
q_6	0	1	0	1	0	0
q_7	0	1	0	1	0	1
q_8	0	0	0	1	0	1

Fig. 6. Locations of the flip-flop circuit

For the given environment and the instantiation π_0, the set of traces of the system is depicted on Fig. 5 under the form of an oriented graph, where q_i, $0 \leq i \leq 8$, are locations of $\mathcal{A}$. The value of the signals of the system for each location q_i is given on Fig. 6. 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.

Let us apply our program IMITATOR to the PTA modeling the flip-flop circuit and to the instantiation π_0 of the parameters. The program generates the following constraint K_0[6]:

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

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

One can check that, for any instantiation π such that $\pi \models K_0$, the set of traces under π coincides with the set of traces under π_0 depicted on Fig. 5.

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. 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 $\pi_1 \not\models K_0$.

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

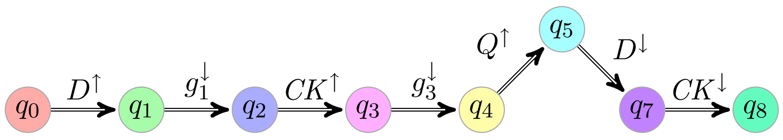

Fig. 7. Traces of the flip-flop circuit under π_1

3.4 Second Instantiation Point

We now aim at widening the constraint K_0 found 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):

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

The set of traces of the system under π_1 is depicted on Fig. 7, under the form of an oriented graph (actually reduced to a single path). The value of the signals of the system for each location q_i is given on Fig. 6. Note that this graph is a subgraph of the set of traces of the system under π_0 (Fig. 5).

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

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

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). Note also that, under our environment, the system under the constraint Z of [10] yields the same set of traces as under π_0 (depicted on Fig. 5).

We notice that, whatever the algorithm randomly chooses at each iteration for the inequality $\neg J$, the constraint finally computed always remains equivalently the same. Hence, for this particular example, the behavior of Imitator seems to be *confluent* (see final remarks in Sect. 12).

4 And–Or Circuit

This example deals with an "And–Or" circuit described in [9] and depicted on Fig. 8. It is composed of 2 gates (one "And" gate and one "Or" gate) which are interconnected in a cyclic way, and the environment which corresponds to 2 input signals a and b, with cyclic alternating rising edges and falling edges. 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" gate takes also a delay in $[\delta_{Or}^-, \delta_{Or}^+]$, and likewise for the "And" gate.

Both gates are 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

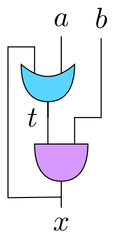

Fig. 8. And–Or Component

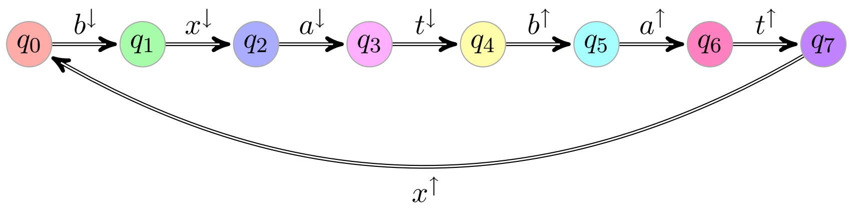

Fig. 9. Traces of the And–Or circuit under π_0

the output (after some delay). The PTA $\mathcal{A}$ modeling the system results from the composition of those 3 PTA. There are 12 timing parameters.

A bad state expresses the fact that the rising edge of output signal x occurs before the rising edge of a within the same cycle. We set the parameters to the following values, ensuring that the bad state is not reachable:

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

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: $b^\downarrow, a^\downarrow, b^\uparrow, a^\uparrow$. For the given environment and the instantiation π_0, the set of traces of the system is depicted on Fig. 9 under the form of an oriented graph, where q_i, $0 \le i \le 7$, are locations of $\mathcal{A}$. The value of the signals of the system for each location q_i is given on Fig. 10. We can check that, in this graph, the bad state is not reached, i.e., the rising edges and falling edges of a, b, x alternate properly.

Using our program Imitator applied to the PTA $\mathcal{A}$ modeling the system and the reference instantiation π_0, the following constraint is computed:

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

Under any instantiation of the parameters $\pi \models K_0$, the set of traces under π is guaranteed to be identical to the set of traces under π_0 given on Fig. 9 and, therefore, does not reach any bad state. As in the case of the flip-flop example, we notice that, whatever the algorithm randomly chooses at each iteration for the inequality $\neg J$, the constraint finally computed always remains equivalently the same (see final remarks in Sect. 12). In [9], the generated constraint is not given.

Location	a	b	t	x
q_0	1	1	1	1
q_1	1	0	1	1
q_2	1	0	1	0
q_3	0	0	1	0
q_4	0	0	0	0
q_5	0	1	0	0
q_6	1	1	0	0
q_7	1	1	1	0
q_8	1	1	1	1

Fig. 10. Locations of the And–Or circuit

5 Root Contention Protocol

We now aim at generating sufficient constraint for a good behavior of the Root Contention Protocol, to elect a leader in the Firewire protocol. As described in [18], the IEEE 1394-1995 standard for a high performance serial bus specifies a bus that supports isochronous and asynchronous data transfers, peer-to-peer, among up to 64 devices. Also, the bus is hot-plug-and-play, which makes it suitable for interconnecting digital multimedia and consumer electronics.

The model we here study is the same as in [18]. It is made of two nodes, two wires, and a safety observer. Two values for *delay* are considered in [18]: $30\,ns$ or $360\,ns$.

5.1 First Case: $delay = 30$

We consider the following instantiation π_{30} of the parameters:

$$\begin{array}{ccc} rc\_fast\_max = 85 & rc\_fast\_min = 76 & delay = 30 \\ rc\_slow\_max = 167 & rc\_slow\_min = 159 & \end{array}$$

Using Imitator, we get the following constraint K_{30}:

$$\begin{array}{rr} & rc\_fast\_max + 2delay < rc\_slow\_min \\ \wedge & 2delay < rc\_fast\_min \end{array}$$

5.2 Second Case: $delay = 360$

We consider the following instantiation π_{360} of the parameters:

$$\begin{array}{ccc} rc\_fast\_max = 85 & rc\_fast\_min = 76 & delay = 360 \\ rc\_slow\_max = 167 & rc\_slow\_min = 159 & \end{array}$$

Using Imitator, we get the following constraint K_{360}:

$$\begin{array}{rr} & rc\_fast\_min > 0 \\ \wedge & 2rc\_slow\_min \leq delay \\ \wedge & rc\_fast\_max < rc\_slow\_min \end{array}$$

6 CSMA/CD

Carrier Sense Multiple Access With Collision Detection (CSMA/CD) is a network control protocol in which a carrier sensing scheme is used, and a transmitting data station that detects another signal while transmitting a frame, stops transmitting that frame, transmits a jam signal, and then waits for a random time interval before trying to send that frame again. Methods for collision detection are media dependent, but on an electrical bus such as Ethernet, collisions can be detected by comparing transmitted data with received data. If they differ, another transmitter is overlaying the first transmitter's signal (a collision), and transmission terminates immediately. A jam signal is sent which will cause all transmitters to back off by random intervals, reducing the probability of a collision when the first retry is attempted.

6.1 First model

We first study the model of this protocol described in [16]. This models contains two senders and a bus, modeled by three Parametric Timed Automata. We consider three parameters: σ is the time for a signal to propagate between the two farthest station, and λ is the time to send a message. We set those two parameters to typical values given in [16], as follows:

$$\lambda = 780 \qquad c = 52 \qquad \sigma = 26$$

We then get the following constraint:

$$\begin{array}{rr} & 0 < \sigma \\ \wedge & c < \lambda \\ \wedge & 2\sigma \leq c \end{array}$$

6.2 Prism model

We now consider the model of this protocol described in [15]. We consider three parameters: λ, σ and *slot*. The following parameters are taken from the IEEE standard 802.3 for 10 Mbps Ethernet. It constitutes the reference valuation for the following three parameters:

$$\lambda = 808 \qquad \mathit{slot} = 52 \qquad \sigma = 26$$

We then get the following constraint:

$$\begin{array}{rr} & \sigma < \mathit{slot} \\ \wedge & \lambda < 16\mathit{slot} \\ \wedge & 15\mathit{slot} < \lambda \end{array}$$

7 Bounded Retransmission Protocol

We here study a simplified version of the Bounded Retransmission Protocol described and modeled by timed automata in [17].

The instantiation π_0 of the parameters of the system is as follows:

$$\begin{array}{ccc} \mathit{MAX} = 2 & N = 2 & \mathit{TD} = 1 \\ \mathit{T1} = 3 & \mathit{TR} = 16 & \mathit{SYNC} = 17 \end{array}$$

We then get the following constraint:

$$\begin{array}{rrcl} & 0 < \mathit{TD} & \wedge & N = 2 \\ \wedge & \mathit{MAX} = 2 & \wedge & \mathit{TR} \leq \mathit{SYNC} + \mathit{TD} \\ \wedge & 2\mathit{TD} < \mathit{T1} & \wedge & \mathit{TR} < 5\mathit{T1} + 2\mathit{TD} \\ \wedge & 4\mathit{T1} + 3\mathit{TD} \leq \mathit{TR} & \wedge & 5\mathit{T1} < \mathit{TR} + \mathit{TD} \end{array}$$

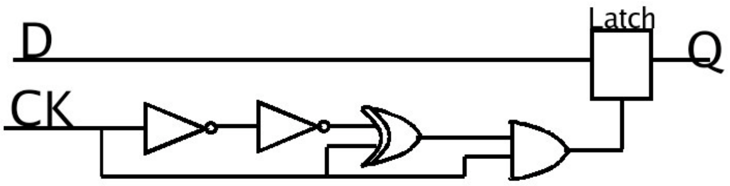

Fig. 11. A latch circuit studied in the case of ANR project Valmem

8 ABR Conformance Protocol

The ABR conformance protocol is a real-time program developed at France Telecom, that control dataflow rates on ATM networks. A crucial part of this protocol is the dynamical computation of the expected rate of data cell emission. We consider here the model of the corresponding program described in [6].

Note that the version of the parametric timed automata considered there is an extension of standard PTA, allowing to reset clocks to a non-null value, and called Updatable Timed Automata [7]. Thus, we notice that Imitator is here successfully applied to another class of systems than PTA.

We choose the following instantiation π of the three parameters of the system:

$$a = 3 \qquad b = 7 \qquad t = 20$$

Using Imitator, we get the following constraint K_0:

$$\begin{array}{cc} & 0 < a \\ \wedge & 2a \leq b \\ \wedge & 2b \leq t \end{array}$$

9 Valmem Project

The examples described in this section have been studied in the case of French ANR project Valmem. This project aims at synthesizing timing constraints of memory circuits designed by chips manufacturer ST-Microelectronics with methods developed by research teams from LSV and LIP6.

9.1 Latch Circuit

This circuit, depicted on Fig. 11, contains 4 gates and one "latch". A bad state corresponds to the fact that the value output signal Q has not changed before the end of the cycle of signal CK.

The system contains 13 parameters. The following instantiation π_0 of these parameters (in ps) were extracted from the circuit description by simulation computed in project Valmem:

$$\begin{array}{rlrlrlrl} T_{HI} & = 1000 & T_{LO} & = 1000 & T_{Hold} & = 350 & T_{Setup} & = 0 \\ \delta_{Not1}{\uparrow} & = 219 & \delta_{Not1}{\downarrow} & = 147 & \delta_{Not2}{\uparrow} & = 155 & \delta_{Not2}{\downarrow} & = 163 \\ \delta_{Xor}{\uparrow} & = 147 & \delta_{Xor}{\downarrow} & = 416 & \delta_{And}{\uparrow} & = 80 & \delta_{And}{\downarrow} & = 155 \\ \delta_{Latch}{\uparrow} & = 240 & & & & & & \end{array}$$

Under this instantiation, the system does not reach the bad state. Using our program Imitator, the following constraint K_0 is computed in 20 seconds:

$$
\begin{array}{lrl}
 & 0 & < \delta_{And\downarrow} \\
\wedge & \delta_{Xor\uparrow} & = \delta_{Not1\downarrow} \\
\wedge & \delta_{Latch\uparrow} + \delta_{And\uparrow} & < T_{Hold} \\
\wedge & T_{Setup} & < T_{LO} \\
\wedge & \delta_{And\uparrow} & < \delta_{Xor\uparrow} \\
\wedge & T_{Hold} & < \delta_{Xor\downarrow} + \delta_{Not2\uparrow} + \delta_{Xor\uparrow} \\
\wedge & \delta_{Xor\uparrow} + \delta_{Not2\uparrow} & < \delta_{Latch\uparrow} + \delta_{And\uparrow} \\
\wedge & \delta_{Xor\uparrow} + \delta_{Xor\downarrow} + \delta_{And\downarrow} + \delta_{Not2\uparrow} & \leq T_{HI}
\end{array}
$$

Under this constraint, the system has the same behavior as under the instantiation, and therefore guarantees a good behavior of the system. Moreover, we are interested in minimizing the T_{Hold} value, provided the system keeps its good behavior. By instantiating all the parameters in K_0 with their value in π_0, except T_{Hold}, we get the following constraint:

$$320 < T_{Hold} < 718$$

So we can minimize the value of T_{Hold} to 321, and we guarantee that the system will have exactly the same behavior as before.

9.2 SPSMALL Memory

We studied a portion of the SPSMALL memory designed and sold by ST-Microelectronics. This memory is described in [8].

The following instantiation of the parameters was automatically extracted from the circuit description by simulation:

$$
\begin{array}{rlrlrl}
\mathit{d\_up\_q\_0} & = 21 & \mathit{d\_dn\_q\_0} & = 20 & \mathit{d\_up\_net27} & = 0 \\
\mathit{d\_dn\_net27} & = 0 & \mathit{d\_up\_d\_inta} & = 22 & \mathit{d\_dn\_d\_inta} & = 45 \\
\mathit{d\_up\_wela} & = 0 & \mathit{d\_dn\_wela} & = 22 & \mathit{d\_up\_net45a} & = 5 \\
\mathit{d\_dn\_net45a} & = 4 & \mathit{d\_up\_net13a} & = 19 & \mathit{d\_dn\_net13a} & = 13 \\
\mathit{d\_up\_net45} & = 21 & \mathit{d\_dn\_net45} & = 22 & \mathit{d\_up\_d\_int} & = 14 \\
\mathit{d\_dn\_d\_int} & = 18 & \mathit{d\_up\_en\_latchd} & = 28 & \mathit{d\_dn\_en\_latchd} & = 32 \\
\mathit{d\_up\_en\_latchwen} & = 5 & \mathit{d\_dn\_en\_latchwen} & = 4 & \mathit{d\_up\_wen\_h} & = 11 \\
\mathit{d\_dn\_wen\_h} & = 8 & \mathit{d\_up\_d\_h} & = 95 & \mathit{d\_dn\_d\_h} & = 66 \\
T_{HI} & = 45 & T_{LO} & = 65 & T_{setupd} & = 108 \\
T_{setupwen} & = 48 & & & &
\end{array}
$$

Under this instantiation, the system has a good behavior. We then get the following constraint K_0:

$$\begin{array}{ll}
\wedge & T_{LO} < \mathit{d\_dn\_d\_inta} + \mathit{d\_dn\_d\_int} + \mathit{d\_up\_en\_latchd} \\
\wedge & \mathit{d\_up\_en\_latchwen} \geq 0 \\
\wedge & \mathit{d\_up\_d\_inta} + \mathit{d\_up\_d\_int} + \mathit{d\_up\_d\_h} < \mathit{d\_dn\_en\_latchd} + T_{setupd} \\
\wedge & \mathit{d\_dn\_wela} + \mathit{d\_dn\_net13a} < T_{HI} \\
\wedge & \mathit{d\_dn\_en\_latchd} < \mathit{d\_dn\_wela} + \mathit{d\_dn\_net13a} \\
\wedge & \mathit{d\_dn\_wen\_h} + T_{LO} < \mathit{d\_up\_en\_latchd} + T_{setupwen} \\
\wedge & T_{LO} < \mathit{d\_up\_net13a} + T_{setupwen} \\
\wedge & \mathit{d\_up\_wela} \geq 0 \\
\wedge & \mathit{d\_up\_wela} + \mathit{d\_up\_net13a} + T_{setupwen} < \mathit{d\_dn\_wen\_h} + T_{LO} \\
\wedge & \mathit{d\_up\_wela} + \mathit{d\_dn\_net45a} + \mathit{d\_dn\_net45} + \mathit{d\_dn\_wen\_h} + T_{setupd} < \mathit{d\_up\_d\_h} + T_{setupwen} \\
\wedge & \mathit{d\_up\_q\_0} + \mathit{d\_up\_net27} + \mathit{d\_dn\_wela} + \mathit{d\_dn\_net13a} < \mathit{d\_up\_net13a} + T_{HI} \\
\wedge & \mathit{d\_up\_en\_latchwen} + T_{setupwen} < T_{LO} \\
\wedge & \mathit{d\_up\_en\_latchwen} + T_{HI} < \mathit{d\_up\_q\_0} + \mathit{d\_up\_net27} + \mathit{d\_dn\_wela} + \mathit{d\_dn\_net13a} \\
\wedge & \mathit{d\_up\_d\_h} < T_{setupd} \\
\wedge & \mathit{d\_dn\_en\_latchwen} \geq 0 \\
\wedge & \mathit{d\_dn\_net13a} + T_{setupd} < \mathit{d\_up\_d\_inta} + \mathit{d\_up\_d\_int} + \mathit{d\_up\_d\_h} \\
\wedge & T_{LO} \leq T_{setupd} \\
\wedge & \mathit{d\_up\_en\_latchd} < \mathit{d\_dn\_net45a} + \mathit{d\_dn\_net45} + \mathit{d\_up\_en\_latchwen} \\
\wedge & \mathit{d\_dn\_en\_latchwen} < \mathit{d\_dn\_net13a} \\
\wedge & \mathit{d\_dn\_wen\_h} + T_{LO} < \mathit{d\_up\_net45a} + \mathit{d\_up\_net45} + \mathit{d\_up\_en\_latchwen} + T_{setupwen} \\
\wedge & T_{setupd} < T_{HI} + T_{LO} \\
\wedge & \mathit{d\_up\_d\_inta} + \mathit{d\_up\_d\_int} + \mathit{d\_up\_en\_latchd} < T_{LO} \\
\wedge & \mathit{d\_dn\_net45a} + \mathit{d\_dn\_net45} + \mathit{d\_up\_en\_latchwen} < \mathit{d\_up\_d\_inta} + \mathit{d\_up\_d\_int} + \mathit{d\_up\_en\_latchd} \\
\wedge & \mathit{d\_up\_net45a} + \mathit{d\_up\_net45} + \mathit{d\_up\_en\_latchwen} < T_{LO}
\end{array}$$

Under this constraint, the system has the same behavior as under the instantiation, and therefore guarantees a good behavior of the system. Moreover, we are interested in minimizing the values of $T_{setupwen}$ and T_{setupd}, provided the system keeps its good behavior. By instantiating all the parameters in K_0 with their value in π_0, except $T_{setupwen}$ and T_{setupd}, we get the following constraint:

$$\begin{array}{lrl}
 & 46 < T_{setupwen} & < 54 \\
\wedge & 99 < T_{setupd} & < 110 \\
\wedge & T_{setupd} < T_{setupwen} & + 61
\end{array}$$

Which allows us to minimize $T_{setupwen}$ and T_{setupd} to 47 and 100 respectively. This corresponds for T_{setupd} to an optimization of 7,4 %. And we guarantee that the system will have exactly the same behavior as before.

10 SIMOP

SIMOP project is a joint work with LSV and LURPA, ENS de Cachan, aiming at defining several good behavior zones for a distributed control system.

10.1 Description

We here consider a distributed control system. Focus is put on architectures where logic controllers and remote input-output modules (RIOMs) communicate to carry out automation functions. With the selected protocol, controllers are clients and RIOMs are data servers. The main features of the physical components of these architectures are:

- Controllers (Programmable Logical Controllers (PLCs) or industrial computers) are modular. Within each controller, a calculus processor runs a program cyclically, while a communication processor performs a periodic scanning of some RIOMs, termed I/O scanning. It matters to underline that the cycles of these two processors are asynchronous, data exchanges being made by means of a shared memory.

- The network includes Ethernet switches and Ethernet links and is dedicated only to communications between the PLCs and RIOMs; there is no other additional traffic.
- Inputs and outputs from/to the plant are gathered in RIOMs which are directly connected to the network. One RIOM may be shared by several PLCs.

Full description is available in [3].

Using our program Imitator and good functioning points, we can infer constraints starting from those points, and therefore define good functioning zones.

10.2 First Instantiation Point

10.2.1 Valuation of the parameters

$$\begin{aligned}
PLCct &= 300\\
COMct &= 1000\\
COMct &= 2071\\
PLCmtt &= 100\\
RIOd &= 70\\
COMd &= 25\\
NETd &= 10
\end{aligned}$$

10.2.2 Inequalities found

$$\begin{aligned}
2COMct + RIOd &< SIGmrt\\
7PLCct &< COMd + 2COMct + NETd + RIOd\\
COMd + 2COMct + NETd &< 7PLCct\\
3COMct &< 10PLCct + NETd\\
2COMct + 2NETd + RIOd &< 7PLCct\\
COMct &< 3PLCct + COMd + NETd + RIOd\\
COMd + NETd &< PLCmtt\\
10PLCct &< 3COMct + NETd\\
3PLCct + 2NETd + RIOd &< COMct\\
COMct &< PLCmtt + 3PLCct + NETd\\
3PLCct + COMd + NETd &< COMct\\
2NETd + RIOd &< PLCmtt\\
2COMct &< PLCmtt + 6PLCct + COMd + NETd + RIOd
\end{aligned}$$

10.3 Second Instantiation Point

10.3.1 Valuation of the parameters

$$\begin{aligned}
PLCct &= 600\\
COMct &= 500\\
COMct &= 2071\\
PLCmtt &= 100\\
RIOd &= 70\\
COMd &= 25\\
NETd &= 10
\end{aligned}$$

10.3.2 Inequalities found

$$\begin{aligned}
4COMct + RIOd &< SIGmrt\\
PLCct &< COMd + COMct + NETd + RIOd\\
0 &< NETd\\
COMct + 2NETd + RIOd &< PLCct\\
COMd + COMct + NETd &< PLCct\\
PLCct &< PLCmtt + COMct + NETd\\
2NETd + RIOd &< PLCmtt\\
COMd + NETd &< PLCmtt\\
4COMct &< PLCmtt + 3PLCct + COMd + NETd + RIOd\\
3PLCct + COMd + 2NETd + RIOd &< 4COMct
\end{aligned}$$

10.4 Third Instantiation Point

10.4.1 Valuation of the parameters

$$\begin{aligned}
PLCct &= 525\\
COMct &= 436\\
COMct &= 2071\\
PLCmtt &= 100\\
RIOd &= 70\\
COMd &= 25\\
NETd &= 10
\end{aligned}$$

10.4.2 Inequalities found (K_3)

$$\begin{aligned}
& & 4COMct + RIOd &< SIGmrt\\
&\wedge & 4COMct &< PLCmtt + 3PLCct + RIOd\\
&\wedge & PLCmtt + 3PLCct + COMd + NETd &\leq 4COMct\\
&\wedge & PLCct + NETd &< PLCmtt + COMct\\
&\wedge & PLCmtt &\leq COMd + NETd + RIOd\\
&\wedge & PLCct &< COMct + 2NETd + RIOd\\
&\wedge & COMct + NETd + RIOd &< PLCct
\end{aligned}$$

After instantiation of $PLCmtt$, $RIOd$, $COMd$, $NETd$, we get:

$$\begin{aligned}
& & SIGmrt &> 4COMct + 70\\
&\wedge & 4COMct &\geq 3PLCct + 135\\
&\wedge & PLCct &> COMct + 80\\
&\wedge & PLCct &< COMct + 90\\
&\wedge & 4COMct &< 3PLCct + 170
\end{aligned}$$

10.5 Fourth Instantiation Point

10.5.1 Valuation of the parameters

$$\begin{aligned}
PLCct &= 525\\
COMct &= 435\\
COMct &= 2071\\
PLCmtt &= 100\\
RIOd &= 70\\
COMd &= 25\\
NETd &= 10
\end{aligned}$$

Example	# PTAs	loc. per PTA	$\lvert X\rvert$	$\lvert P\rvert$	# iter.	$\lvert Post^*\rvert$	$\lvert K_0\rvert$	CPU time
Flip-flop (π_0) [10]	5	$[4, 16]$	5	12	8	11	7	2 s
And–Or [9]	3	$[4, 8]$	4	14	9	9	7	3 s
RCP (π_{30}) [18]	5	$[6, 11]$	6	5	18	229	2	64 s
RCP (π_{360}) [18]	5	$[6, 11]$	6	5	20	871	3	480 s
CSMA/CD [16,19]	3	$[6, 7]$	4	3	21	294	3	108 s
CSMA/CD [15]	3	$[3, 8]$	3	3	17	218	3	44 s
BRP [17]	6	$[2, 6]$	7	6	29	427	8	15 min 23
ABR [6]	3	$[2, 5]$	1	3	18	456	3	16 min 36
Latch	7	$[2; 5]$	7	15	11	18	8	20 s
SPSMALL [8]	10	$[3, 8]$	10	22	31	31	23	78 min
SIMOP [3]	5	$[5, 16]$	8	7	51	956	9	419 min

Fig. 12. Case studies using Imitator

10.5.2 Inequalities found (K_4)

$$\begin{array}{lrcl}
 & 4COMct + RIOd & < & SIGmrt \\
\wedge & PLCct & < & COMd + COMct + NETd + RIOd \\
\wedge & 3PLCct + COMd + 2NETd + RIOd & < & 4COMct \\
\wedge & 4COMct & < & PLCmtt + 3PLCct + RIOd \\
\wedge & COMct + NETd + RIOd & < & PLCct \\
\wedge & PLCct & < & PLCmtt + COMct \\
\wedge & 2NETd + RIOd & < & PLCmtt \\
\wedge & COMd + COMct + NETd & < & PLCct
\end{array}$$

After instantiation of $PLCmtt$, $RIOd$, $COMd$, $NETd$, we get:

$$\begin{array}{lrcl}
 & SIGmrt & > & 4COMct + 70 \\
\wedge & 4COMct & > & 3PLCct + 115 \\
\wedge & PLCct & < & COMct + 100 \\
\wedge & PLCct & > & COMct + 80 \\
\wedge & 4COMct & < & 3PLCct + 170
\end{array}$$

It is easy to verify that this constraint strictly includes K_3.

11 Summary of the Experiments

We give on Fig. 12 the summary of our experiments. We give from left to right the name of the example with a reference, the number of PTAs, the lower and upper bounds on the number of locations per PTA, the number of clocks, the number of parameters, the number of iterations of the algorithm, the number of states in $Post^*$, the number of inequalities in K_0 (after reduction), and the computation time on an Intel Quad Core 3 GHz with 3.2 Gb.

The instantiation π_0 we use for each example is either an instance satisfying the constraint Z generated by classical synthesis constraints when such a Z is available (e.g., in the flip-flop, RCP, BRP examples), or corresponds to typical data given with the case study (e.g., in the CSMA/CD, SIMOP, SPSMALL 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 [10]. This suggests to combine both information, in order to widen our constraint K_0, as described in the incremental method of [5]. In the second case, the constraint allows us to safely decrease (or increase) some components of the typical data π_0, as far as they still satisfy K_0.

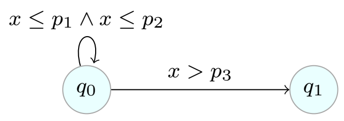

Fig. 13. PTA showing the non-confluence of Imitator

This is useful, for example in order to relax some requirements on the environment of asynchronous circuits (see, e.g., [8]).

12 Final Remarks

Given a reference valuation π_0, Imitator solves the inverse problem for systems modeled by PTA with acyclic traces : it returns a constraint K_0 on the parameters guaranteeing that the sets of traces of $\mathcal{A}[\pi_0]$ and $\mathcal{A}[\pi]$ are identical, for any valuation π such that $\pi \models K_0$. K_0 prevents all the bad behaviors (e.g. deadlocks), since it *imitates* the reference behavior of π_0, while constraints generated by classical methods may not prevent bad behaviors other than those specified by the bad states.

Imitator is a complementary tool which can be used to improve constraints of classical methods. Considering the flip-flop example of Sect. 3, Imitator first generates a constraint K_0 incomparable with constraint, say Z, of [10]. We can run Imitator again with a reference valuation $\pi_1 \in Z \setminus K_0$, which outputs K_1, s.t. $K_0 \cup K_1$ is strictly larger than Z.

Note that the computation time of several examples could be reduced, because of the use of HyTech. Indeed, HyTech computes an *a priori* composition of the automata, which is time-consuming in the case of several medium-sized automata. Another tool using a library for computing operations on polyhedra is under project.

Non-Confluence of Algorithm InverseMethod

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. Consider the PTA depicted on Fig. 13. This PTA contains 2 locations q_0 and q_1, one clock x, and 3 parameters p_1, p_2, p_3. We consider the following reference valuation π_0 of the parameters:

$$p_1 = 1 \ \wedge \ p_2 = 1 \ \wedge \ p_3 = 2$$

It is easy to see that an application of InverseMethod to this PTA and this instantiation π_0 will output a constraint K_0 either equal to $p_1 \leq p_3$, or equal to $p_2 \leq p_3$, depending on which $\neg J$ is selected in the algorithm.

It follows from this remark about the non-confluence that the constraint K_0 output by InverseMethod 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. The maximal constraint is actually probably not in conjunctive form in the general case: on the example of Fig. 13, it is easy to see that the maximal constraint guaranteeing the same behavior as under π_0 is $p_1 \leq p_3 \vee p_2 \leq p_3$, which is not in conjunctive form.

In practice, we observe on all the experiments detailed in this report 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 always maximal. It would be interesting to evaluate how large is the constraint generated by InverseMethod.

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] Étienne André. Imitator: A tool for synthesizing constraints on timing bounds of timed automata. In *ICTAC'09*, LNCS. Springer, August 2009. To appear.

[5] Étienne André, Thomas Chatain, Emmanuelle Encrenaz, and Laurent Fribourg. An inverse method for parametric timed automata. *International Journal of Foundations of Computer Science*, 2009. To appear.

[6] B. Bérard and L. Fribourg. Automated verification of a parametric real-time program: The ABR conformance protocol. In *CAV'99*, volume 1633 of *LNCS*, pages 96–107, Trento, Italy, 1999. Springer.

[7] P. Bouyer, C. Dufourd, E. Fleury, and A. Petit. Updatable timed automata. *Theoretical Computer Science*, 321(2-3):291–345, August 2004.

[8] 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.

[9] R. Clarisó and J. Cortadella. Verification of concurrent systems with parametric delays using octahedra. In *ACSD '05*. IEEE Computer Society, 2005.

[10] R. Clarisó and J. Cortadella. The octahedron abstract domain. *Sci. Comput. Program.*, 64(1):115–139, 2007.

[11] 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.

[12] 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.

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

[14] 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.

[15] M. Kwiatkowska, G. Norman, J. Sproston, and F. Wang. Symbolic model checking for probabilistic timed automata. In *FORMATS/FTRTFT*, pages 293–308, 2004.

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

[17] P.R. D'Argenio, J.P. Katoen, T.C. Ruys, and G.J. Tretmans. The bounded retransmission protocol must be on time! In *TACAS '97*. Springer, 1997.

[18] 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.

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