Valmem Project – Deliverable 3.3

IMITATOR : A Prototype of Verifier

Étienne André[1], Emmanuelle Encrenaz[2], Laurent Fribourg[1]

[1] LSV – ENS de Cachan & CNRS, France

[2] LIP6 – Université Pierre et Marie Curie & CNRS, France

1 Introduction

We consider in this report 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 to model 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.

The application to asynchronous circuits, e.g., the SPSMALL memory, is natural : we are given a model of the circuit in terms of parametric timed automata, as well as timing values for the traversal delays guaranteeing a good (or standard) behavior of the circuit. We consider the system to be fully parametric (i.e., the traversal delays are considered to be parameters) and we infer a constraint on those parameters ensuring that the system will have the same behavior. We can finally instantiate some of the parameters of the constraint in order to minimize or maximize some particular delays, e.g., the setup or hold timings of the input signals.

We shortly present the algorithm InverseMethod in Sect. 2, and then its implementation IMITATOR in Sect. 3. Afterwards, we present two case studies in Sect. 4, i.e., a simple latch circuit, and a portion of the SPSMALL memory. We give some final remarks in Sect. 5.

2 The Algorithm

We use in this section the same formalism as in [1]. We consider a timed system modeled with (a network of) Parametric Timed Automata (PTA) $\mathcal{A}$. Our goal is, starting from an instantiation reference of the parameters π_0, to compute a constraint K_0 on the parameters, such that, under any valuation π of the

ALGORITHM InverseMethod($\mathcal{A}$, π_0)

Input $\mathcal{A}$: PTA
π_0 : Valuation of P
Output K_0 : Constraint on the parameters
Variables i : Current iteration
S : Current set of symbolic states ($S = Post^i_{\mathcal{A}(K)}$)
S' : Former set of symbolic states
K : Current constraint on the parameters

$i := 0$; $K := True$; $S := \{s_0\}$; $S' := \{s_0\}$
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 $\pi_0 \models \neg J$
    $S' := S$
    $K := K \wedge \neg J$
    $S := Post^i_{\mathcal{A}(K)}$
  OD
  %% S π_0-compatible
  $S' := S$
  $S := Post_{\mathcal{A}(K)}(S)$
  $i := i + 1$
  IF $S = S'$ *%% S π_0-compatible and $S = Post^*_{\mathcal{A}(K)}$*
    THEN RETURN $K_0 := \bigcap_{(q,C)\in S}(\exists X : C)$
  FI
OD

Figure 1: Algorithm InverseMethod

parameters such that $\pi \models K_0$, the behavior of $\mathcal{A}[\pi_0]$ and the behavior of $\mathcal{A}[\pi]$ are the same, i.e., their sets of traces are equal.

We now present the algorithm InverseMethod on Fig. 1. For a more complete presentation, as well as for correctness proof and termination condition, see [1]. 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 $\pi_0 \models \neg J$).

Note also that $\pi_0 \models K_0$, as the final set of states S is π_0-compatible and K_0 is $\bigcap_{(q,C)\in S}(\exists X : C)$.

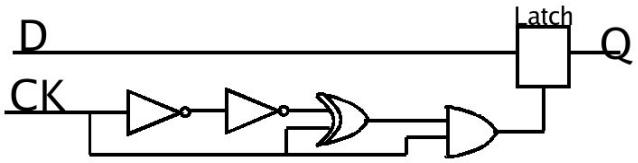

Figure 2: A latch circuit

3 The Program IMITATOR

Our algorithm InverseMethod has been implemented under the form of a program named IMITATOR (standing for *Inverse Method for Inferring Time Abstract behaviOR*). This program, containing about 1500 lines of code, is written in Python and needed about two man-months. It calls parametric model checker HyTech [3] in order to compute the *Post* operation, and makes use of Prolog in order to check the π_0-compatibility. The call to Prolog was decided for simplicity reasons, but it could be easily performed with Python, as this only verifies that an instantiation models a constraint. The selection of a π_0-compatible state and a π_0-incompatible inequality J is done in a random manner. In practice, we observe 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.

More information, as well as the source code of the case studies presented here, can be found on the program's webpage (`http://www.lsv.ens-cachan.fr/~andre/IMITATOR/`).

4 Case Studies

All experiments were run on an Intel Quad Core 3 GHz with 3.2 Gb.

4.1 Example of Latch Circuit

This simple circuit was given by Remy Chevallier (ST-Microelectronics) in order to test our implementation. This circuit, depicted on Fig. 2, 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 :

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

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 :

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

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.

4.2 SPSMALL Memory

We studied a portion of the SPSMALL memory designed and sold by ST-Microelectronics. This memory is described in [2].

The model was automatically generated by LIP6 from the transistor netlist. The following instantiation of the parameters was extracted after simulation by Remy Chevallier (ST-Microelectronics) :

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

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

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

Example	# of PTA	loc. per PTA	$\|X\|$	$\|P\|$	# of iter.	$\|Post^*\|$	$\|K_0\|$	CPU time
Latch	7	[2; 5]	7	15	11	18	8	20 s
SPSMALL [2]	10	[3, 8]	10	22	31	31	23	1h18

Figure 3: Case studies using IMITATOR

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:

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

Which allows us to minimize $T_{setupwen}$ and T_{setupd} to 47 and 100 respectively. And we guarantee that the system will have exactly the same behavior as before. Note that, by behavior, we mean a sequence of internal events. We could have other behaviors which still do not reach the bad state. Thus, we could apply again IMITATOR on the values $T_{setupwen} = 46$ and $T_{setupd} = 99$, after having checked that the bad state of the system is not reached, and minimizing again their values, until the bad state is reached.

5 Conclusion

We give on Fig. 3 the summary of our experiments. We give from left to right the name of the example with a reference, the number of PTA, 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, and the computation time on an Intel Quad Core 3 GHz with 3.2 Gb.

Note that the computation time of several examples could be reduced, by using a library for computing operations on polyhedra. rather than HyTech. Presently, it is indeed not possible in the current tool to consider circuits modeled by more than 10 PTA, which is a strong limit to the scalability of our method. We are now working on how to remove this constraint. Possible options are the composition of several subsystems of 10 PTA, or the implementation of another tool using a library for computing operations on polyhedra.

References

[1] É. André, T. Chatain, E. Encrenaz, and L. Fribourg. An inverse method for parametric timed automata. In *RP'08*, Electronic Notes in Theoretical Computer Science, Liverpool, UK, 2008. Elsevier Science Publishers.

[2] R. Chevallier, E. Encrenaz-Tiphène, L. Fribourg, and W. Xu. Verification of the generic architecture of a memory circuit using parametric timed automata. In *FORMATS '06*, volume 4202 of *LNCS*, Paris, France, 2006. Springer.

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