IMITATOR: a Tool for Synthesizing Constraints on Timing Bounds of Timed Automata*

Étienne André

LSV – ENS de Cachan & CNRS, France

Abstract. We present here 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.

1 Context

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, 11].

The synthesis of constraints for PTA has been mainly done by supposing given a set of "bad states" (see, e.g., [8, 9]). 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.

2 Principle of IMITATOR

The tool IMITATOR (*Inverse Method for Inferring Time AbstracT behaviOR*) implements the algorithm *InverseMethod*, described in [4]. We assume given a

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

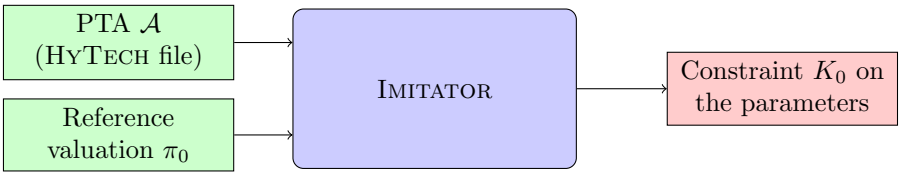
system modeled by a PTA $\mathcal{A}$. Whereas bad-state oriented methods consider a set of bad states, IMITATOR considers an initial tuple π_0 of values for the parameters, under which the system is known to behave properly. When the parameters are instantiated with π_0, the system is denoted by $\mathcal{A}[\pi_0]$. 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* [4] for *acyclic* systems (i.e., with only finite traces) by computing a constraint K_0 such that :

1. $\pi_0 \models K_0$,
2. $\mathcal{A}[\pi] \equiv_{TA} \mathcal{A}[\pi_0]$, for any $\pi \models K_0$.

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").

The tool IMITATOR is available on its Web page[1].

3 General Structure



As depicted above, 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.

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[2]. 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.

A simplified version of algorithm *InverseMethod* is given below, where the clock variables have been disregarded for the sake of simplicity. We denote by

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

[2] 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 [4] for more details.

$Post^i_{\mathcal{A}(K)}(S)$ 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.

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

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

$i := 0$; $K := True$; $S := \{s_0\}$
DO
  DO UNTIL there are no π_0-incompatible states in S
    Select a π_0-incompatible state (q, C) of S (i.e., s.t. $\pi_0 \not\models C$)
    Select a π_0-incompatible J in C (i.e., s.t. $\pi_0 \not\models J$)
    $K := K \wedge \neg J$; $S := \bigcup_{j=0}^{i} Post^j_{\mathcal{A}(K)}(\{(s_0)\})$
  OD
  IF $Post_{\mathcal{A}(K)}(S) = \emptyset$ **THEN RETURN** $K_0 := \bigcap_{(q,C)\in S}(\exists X : C)$
  FI
  $i := i + 1$; $S := S \cup Post_{\mathcal{A}(K)}(S)$
OD

This algorithm terminates and solves the inverse problem for acyclic systems. The acyclic class is interesting for hardware verification, e.g., when analyzing synchronous circuits over a fixed number (typically, 1 or 2) of clock cycles.

IMITATOR is a program written in Python, that drives HYTECH [10] for the computation of the *Post* operation. The Python program contains about 1500 lines of code, and it took about 4 man-months of work.

Remark. In order to handle cyclic examples, one modifies the algorithm by replacing, in the **IF** condition, $Post_{\mathcal{A}(K)}(S) = \emptyset$ by $Post_{\mathcal{A}(K)}(S) \subseteq S$. In that case, we ensure termination more often (see [4]). However, we do not guarantee any longer the identity of traces, but only the identity of reachable locations. This 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 $\pi \models K_0$.

4 An Illustrating Example

We consider an asynchronous "D flip-flop" circuit described in [7] and depicted on Fig. 1. 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. Each gate is modeled by a PTA,

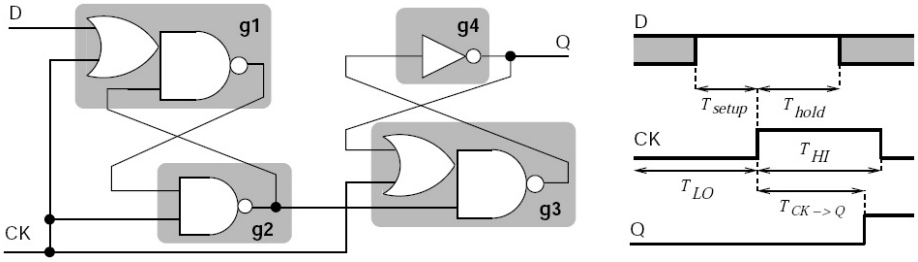
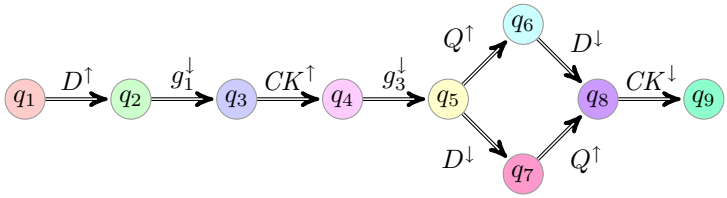


Fig. 1. Flip-flop circuit

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[3] of those 5 PTAs. The output signal of a gate G_i is named g_i (note that $Q = g_4$). 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 the following instantiation π_0 of the parameters:

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

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. 1 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 (alternating sequences of locations and actions) of the system is depicted below under the form of an oriented graph, where q_i, $1 \leq i \leq 9$, are locations of $\mathcal{A}$.



Applying IMITATOR to $\mathcal{A}$ and π_0, we get the following constraint K_0[4]:

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

For any valuation π satisfying K_0 and for the same environment, the set of traces of the system $\mathcal{A}[\pi]$ coincides with the one depicted above, i.e., $\mathcal{A}[\pi] \equiv_{TA} \mathcal{A}[\pi_0]$. For a comparison of K_0 with the constraint found in [7], see [5].

[3] The standard parallel composition of several PTAs is a PTA.

[4] 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.

5 Experiments

We applied the tool IMITATOR to various case studies from the literature, including a flip-flop circuit (described in Sect. 4), two protocols (root contention and CSMA/CD), as well as two real case studies : a portion of the memory circuit SPSMALL designed by ST-Microelectronics, and a distributed control system (SIMOP). All those experiments are detailed in [5]. The HYTECH source code of all the examples is available on IMITATOR webpage.

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

The above table gives from left to right the name of the example, the number of PTAs composing the system $\mathcal{A}$, the lower and upper bounds on the number of locations per PTA, the numbers of clocks and parameters of $\mathcal{A}$, of iterations of the algorithm, of reached symbolic states, of inequalities in K_0 (after reduction), and the computation time on an Intel Quad Core 3 GHz with 3.2 Gb.

All these examples are acyclic[5], and thus guarantee the equality of traces, except SIMOP. In this latter case, we are only interested in avoiding a given bad location, and the equality of reachable locations is sufficient.

In the flip-flop and RCP examples, we took as π_0 an instance satisfying a constraint issued from a classical synthesis method of the literature. In this case, the constraint generated by our method may be the same as the constraint from the literature, but not necessarily : for example, in the case of the flip-flop circuit, K_0 is uncomparable with the original constraint of [7] (see [5] for details).

In the CSMA/CD, SIMOP, VALMEM examples, the instantiation π_0 corresponds to typical data associated to the case study. In this case, the constraint K_0 allows us to optimize some values of the typical data π_0. This is useful, for example in order to safely relax some requirements on the environment of asynchronous circuits (see, e.g., [6]). In the SPSMALL case study, this allows us to safely optimize some nominal setup timing by 8 % (see [5]).

Running HYTECH in a brute manner (fully parametric forward analysis) quickly leads to a saturation of the memory for most examples. One reason for which IMITATOR behaves well in practice is that the procedure drastically reduces the number of reachable states by quickly restraining K_0.

6 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 parame-

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

ters 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 can be used in an *incremental* way as a complementary tool to enlarge constraints given by classical methods. For example, in the flip-flop case (see Sect. 4), the constraint, say Z, found in [7] is uncomparable with our constraint K_0. We can run IMITATOR once more with a reference valuation $\pi_1 \in Z \setminus K_0$. This gives a new constraint K_1, s.t. $K_0 \cup K_1$ is strictly larger than Z (see [5]).

Acknowledgments. I thank anonymous referees for their 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, LSV, ENS Cachan, France, 2009.
4. É. André, T. Chatain, E. Encrenaz, and L. Fribourg. An inverse method for parametric timed automata. *International Journal of Foundations of Computer Science (IJFCS)*. To appear.
5. 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.
6. 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.
7. R. Clarisó and J. Cortadella. The octahedron abstract domain. In *SAS '04*, 2004.
8. 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.
9. 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.
10. T. A. Henzinger, P. Ho, and H. Wong-Toi. A user guide to HYTECH. In *TACAS*, pages 41–71, 1995.
11. T. Hune, J. Romijn, M. Stoelinga, and F. W. Vaandrager. Linear parametric model checking of timed automata. In *TACAS '01*, pages 189–203. Springer-Verlag, 2001.
12. X. Nicollin, J. Sifakis, and S. Yovine. Compiling real-time specifications into extended automata. *IEEE Trans. on Software Engineering*, 18:794–804, 1992.
13. 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.
14. Farn Wang. Symbolic parametric safety analysis of linear hybrid systems with BDD-like data-structures. *IEEE Trans. Softw. Eng.*, 31(1):38–51, 2005.