ENS CACHAN

Étienne André

Everything You Always Wanted to Know About IMITATOR (But Were Afraid to Ask)

Research Report LSV-09-20

July 2009

Laboratoire Spécification et Vérification

CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE

Ecole Normale Supérieure de Cachan
61, avenue du Président Wilson
94235 Cachan Cedex France

This is a research report published by Laboratoire Spécification et Vérification.

Everything You Always Wanted to Know About Imitator (But Were Afraid to Ask)

Étienne André[1]

LSV, ENS de Cachan & CNRS, France

Abstract

We present here the user manual of IMITATOR, a tool for synthesizing constraints on timing bounds (seen as parameters) in the framework of timed automata. Unlike classical synthesis methods, the tool IMITATOR takes advantage of a given reference valuation of the parameters for which the system is known to behave properly. The goal of IMITATOR 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. We give here the installation requirements and the launching commands of IMITATOR, as well as the source code of a toy example.

1 Introduction

This document is the user manual of the tool IMITATOR [4] (*Inverse Method for Inferring Time AbstracT behaviOR*), an implementation of the *inverse method* described in [5]. This tool is being developed at LSV, ENS Cachan, France.

2 Imitator in a Nutshell

2.1 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

⋆ 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@lsv.ens-cachan.fr

©2009 Laboratoire Spécification et Vérification

which will guarantee a correct behavior of the system. Such automata are called *parametric timed automata* (PTA) [2,9]. Those PTAs allow to model various kinds of timed systems, e.g. communication protocols or asynchronous circuits.

The synthesis of constraints for PTAs has been mainly done by supposing given a set of "bad states" (see, e.g., [6,7]). 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, IMITATOR is based on a *good-state oriented* method.

2.2 Principle

The tool IMITATOR (*Inverse Method for Inferring Time AbstracT behaviOR*) implements the algorithm *InverseMethod*, described in [5]. We assume given a system modeled by a PTA $\mathcal{A}$. We are not given a set of bad states, but 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 $\mathcal{A}[\pi_0]$. The algorithm *InverseMethod* generalizes this *good* behavior by computing a constraint K_0 guaranteeing 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 and actions are identical. This is written $\mathcal{A}[\pi] \equiv_{TA} \mathcal{A}[\pi_0]$.

As an immediate practical application, one can optimize the value of some parameters of the system, provided this value still satisfies the constraint generated by IMITATOR. This is of particular interest in the framework of asynchronous circuits, where it is useful to safely minimize some timing bounds of the system, without changing the behavior of the system. For example, one can minimize some local stabilization timings, without changing the global delay for writing an input signal in a memory circuit.

The tool IMITATOR is available on its Web page [2].

2.3 The Algorithm

Let us briefly recall here the main idea of the algorithm *InverseMethod* [5]. Given a parametric timed automaton $\mathcal{A}$ and a reference instantiation π_0 of parameters, the algorithm outputs a constraint K_0 on the parameters such that:

(i) $\pi_0 \models K_0$,

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

The algorithm *InverseMethod* on which IMITATOR relies can be summarized as follows. Starting with $K := \mathit{True}$, 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 $\neg J$

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

[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.

```
ALGORITHM InverseMethod(A, π0)
Input      A  : PTA
           π0 : Reference valuation of P
Output     K0 : Constraint on the parameters
Variables  i  : Current iteration
           S  : Current set of reachable states (S = ⋃_{j=0}^{i} Post^j_{A(K)}({s0}))
           K  : Current constraint on the parameters

i := 0 ;  K := True ;  S := {s0}
DO
      DO UNTIL S is π0-compatible
            Select a π0-incompatible state (q, C) of S
            Select an inequality J of (∃X : C) such that π0 ⊨ ¬J
            K := K ∧ ¬J  ;   S := ⋃_{j=0}^{i} Post^j_{A(K)}({s0})
      OD
                              %% S π0-compatible
      IF Post_{A(K)}(S) = ∅
            THEN RETURN   K0 := ⋂_{(q,C)∈S} (∃X : C)
      FI
      i := i + 1
      S := S ∪ Post_{A(K)}(S)          %% S = ⋃_{j=0}^{i} Post^j_{A(K)}({s0})
OD
```

Fig. 1. Algorithm *InverseMethod*

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 $\pi_0 \models \neg J$).

2.4 General Structure

As described on Fig. 2, Imitator [4] takes as an input a PTA described in HyTech syntax. The tool drives indeed the model checker HyTech [8] in a basic manner. Imitator also takes as an input the reference valuation π_0. The program outputs a constraint K_0 on the parameters such that:

(i) $\pi_0 \models K_0$,

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

Imitator is a program written in Python, that uses HyTech 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.

Fig. 2. Imitator inputs and output

3 How to Use Imitator

3.1 Installation

Imitator is a program written in Python, and thus needs Python to be installed on the machine the tool will be launched on. Imitator is guaranteed to work properly with Python 2.4.4. However, as the program uses only very standard features, it should also work with most older and newer versions.

Imitator calls the HyTech model checker, and thus needs HyTech 1.04f to be installed. This version is the most recent one, and is available on the HyTech Web page.

3.2 The HyTech Input File

Beside the classical syntax of HyTech, the input file must follow a certain number of requirements, which are given below.

3.2.1 Variables

Any kind of variables (clocks, parameters, discrete, etc.) can be used. As in a standard HyTech file, they must be declared in the header of the file.

3.2.2 Parametric Timed Automata

A network of (at least one) PTA must be declared. Although HyTech allows other structures than PTAs, be aware that the behavior of Imitator for another kind of systems as PTAs is *unspecified*.

3.2.3 Initial region and π_0

An initial region named `init_reg` must be defined. As in a standard HyTech file, it must contain all the useful information concerning the initial state of the system (initial location, values of clocks and other variables, etc.). In the case where the initial location should have another name, it is possible to change it in the top of the source code of Imitator, where it is defined in the global constant `INIT_REG`.

The reference valuation π_0 definition must be given somewhere in this region `init_reg`, with the following requirements:

- This reference valuation must start with the tag `---START PI0---`;
- One definition of parameter must be given per line;
- The definition must be given using the standard HyTech syntax for affectations in a region definition (`& [param] = [value]`);

- All those definitions must[4] be *commented* (i.e., preceded by the HyTech comment mark `--`), so that Imitator uses those values, but not HyTech, which is used by Imitator in a fully parametric way;
- The reference valuation definition must end with the tag `---END PI0---`.

Imitator allows a little freedom within this syntax, but you are strongly adviced to strictly respect the given syntax.

An example of input file for Imitator corresponding to the example of Section 4 is given in Appendix A.

3.2.4 Analysis commands

An region named `post_reg` must be defined for the computation of the *Post* operation. In the case where this region should have another name, it is possible to change it in the top of the source code of Imitator, where it is defined in the global constant `POST_REG`.

This region can be defined in any way, since it will be modified by Imitator, and can thus contain any definition. For example, it can be defined in the following standard way:

```
post_reg := reach forward from init_reg endreach;
```

Finally, the following code must be inserted at the end of the input file.

```
prints "---START LOG---";
print(hide non_parameters in post_reg endhide);
prints "---END LOG---";
```

The two `prints` commands allow Imitator to parse the HyTech log file in order to find the resulting set of computed states. You can customize the `non parameters` command, but be aware that every variable which is not given a value in the reference valuation definition in the `init_reg` region *must* be hidden at that point, including all clocks. In particular, if for any reason you use other parameters than those defined in π_0, they should be hidden here.

Provided the requirements described in this section are fullfiled, the input file can contain anything else than what is described here. However, you are strongly adviced to remove any other analysis command, since everything which is defined in the input file will be executed at *every Post* computation, and can thus make this computation very slow – or even not terminate.

3.2.5 Summary of requirements

A quick reminder of the requirements for the input file:

(i) Definition of π_0 between two tags in the `input_reg` region;

(ii) Definition of the computation region `post_reg`;

(iii) Sequence of commands to print the result, in between two tags.

[4] It is actually possible to leave some values uncommented, but those “parameters” will be considered by Imitator not as parameters, but as valued constants, and will thus not appear in the final constraint K_0.

An example of input file for Imitator corresponding to the example of Section 4 is given in Appendix A.

3.3 *Calling* Imitator

Given an input HyTech file named `hytech_file.hy`, the following command calls Imitator:

```
python IMITATOR.py hytech_file
```

The HyTech file name must be given with no extension `.hy`.

Note that, in the current version of Imitator, the Python program `IMITATOR.py` should be in the same directory as the HyTech file Imitator is applied to.

3.3.1 *Options*

The options available for Imitator are explained in the following.

`--debug=[debug_mode]` (default: `--debug=no`)

Give some debugging information, that may also be useful to have more details on the way Imitator works. The available values for `debug_mode` are given in the following:

`result_only`	Give only the resulting constraint
`no`	Give little information (number of steps, computation time)
`low`	Give little additional debugging information
`medium`	Give quite a lot of debugging information
`high`	Give much debugging information
`total`	Give really too much information

`-h` or `--help`

Display the launch syntax described above, and the options detailed in this section.

`--keeplog`

Keep the directory containing the temporary files (copy of the original HyTech file, HyTech log files, and a text file containing the resulting constraints). By default, all those files are removed at the end of the computation. The original HyTech file to which Imitator is applied remains, of course, unmodified. See also the `--log_dir` option below.

`--log_dir=[dir_name]` (default: `log_dir = [hytech_file]`)

This option allows to customize the directory where the temporary files will be created. This is of interest when launching two different processes of Imitator applied to the same input file, which will create a conflict if both processes work on the same directory. The directory `dir_name` may exist; if not, it will be created.

`--norandom`

Choose deterministically the π_0-incompatible inequality. In other words, at a given step, under this option, Imitator will negate the first π_0-incompatible inequality which is encountered. By default, the tool computes the set of all π_0-incompatible inequalities, and chooses one randomly, which is a (very little) more time-consuming.

`--timed`

Print the current computation time on screen for every action which is performed (new step, selection of a π_0-incompatible inequality, etc.). By default, only the global computation time is given at the end of computation (except under the option `--debug=result_only`).

`-v` **or** `--version`

Print version information.

3.3.2 Examples of calls

`python IMITATOR.py flipflop`

Call Imitator with the default options. The original HyTech file full name is `flipflop.hy`, and Imitator will create a temporary directory `flipflop/` in which several temporary files will be created. At the end of the computation, this directory will be removed.

`python IMITATOR.py BRP --norandom --debug=result_only`

The original HyTech file is `BRP.hy`. Only the final constraint will be printed on screen.

`python IMITATOR.py spsmall --log_dir=experiments --keeplog`

The original HyTech file is `spsmall.hy`, and Imitator will create a directory `experiments/` in which several temporary files will be created. At the end of the computation, both this directory and all the temporary files will be kept.

3.4 The Resulting Constraint

The resulting constraint K_0 is printed on the standard output.

To keep a trace of it, use option `--keeplog`. In that case, you can find it in a text file in the temporary directory (see option `--log_dir`).

4 A Toy Example

Let us consider the parametric timed automaton (PTA) given in Fig. 3. 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.

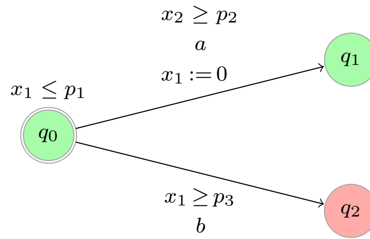

Fig. 3. A toy parametric timed automaton

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 $\pi_0 : p_1 = 4 \wedge p_2 = 2 \wedge p_3 = 6$, under which the PTA is assumed to have a "good" behavior. Then our tool Imitator will generate the constraint $K_0 : p_2 > 0 \wedge p_1 < p_3 \wedge p_2 \leq p_1$. For all instantiation π of the parameters satisfying K_0, our method guarantees that the PTA behaves 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 reachability 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). The HyTech input file of this example is given in Appendix A.

Further Examples

The description of a range of case studies from the literature studied with Imitator, as well as real case studies, is available in [3]. Both the source code and the result of those examples are available on Imitator Web page.

5 Imitator Strikes Back

A more sophisticated version of Imitator is under project. It will make use of a library for computing operations on polyhedra, allowing us to get better computation times. Indeed, HyTech performs a costly static composition of the different timed automata of the system, which can be very time-consuming in the case of several medium-sized automata.

Acknowledgments

Laurent Fribourg and Emmanuelle Encrenaz have been great contributors of Imitator, on a theoretical point of view, and to find applications both from the literature and real case studies. Jeremy Sproston and Farn Wang provided examples to be compared with other tools.

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] 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.

[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] 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.

[7] 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.

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

[9] 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.

A HyTech Source Code of the Example

```
1  --**************************************************--
2  --**************************************************--
3  --    Toy Example for IMITATOR
4  --
5  --    Modeling by Etienne ANDRE (LSV)
6  --    IMITATOR: http://www.lsv.ens-cachan.fr/~andre/IMITATOR/
7  --
8  --    Created :          29/11/2008
9  --    Last modified : 29/11/2008
10 --**************************************************--
11 --**************************************************--
12
13 var      x1, x2
14                  : clock;
15
16          p1, p2, p3
17                          : parameter;
18
19 --**************************************************--
20 --**************************************************--
21 -- AUTOMATON
22 --**************************************************--
23 --**************************************************--
24
25 --**************************************************--
26 automaton toy
27 --**************************************************--
28 synclabs: a, b;
29 initially Q0;
30
31 loc Q0: while x1 <= p1 wait {}
32         when x2 >= p2 sync a do {x1' = 0} goto Q1;
33         when x1 >= p3 sync b do {}        goto Q2;
34
35 loc Q1: while x1 >= 0 wait {}
36         when True do {} goto Q1;
37
38 loc Q2: while x1 >= 0 wait {}
39         when True do {} goto Q2;
40 end -- toy
41
42 --**************************************************--
43 --**************************************************--
44 -- ANALYSIS
```

```
45 --*************************************************--
46 --*************************************************--
47
48 var init_reg , post_reg
49         : region;
50
51 init_reg :=
52         ----------------------------------
53         -- Initial locations
54         ----------------------------------
55           loc[toy] = Q0
56
57         ----------------------------------
58         -- Clocks
59         ----------------------------------
60         & x1 = 0
61         & x2 = 0
62
63         ----------------------------------
64         ----START PI0----
65         ----------------------------------
66 --      & p1 = 4
67 --      & p2 = 2
68 --      & p3 = 6
69         ----------------------------------
70         ----END PI0----
71         ----------------------------------
72 ;
73
74 post_reg := reach forward from init_reg endreach;
75
76
77 prints "----START LOG----";
78 print(hide non_parameters in post_reg endhide);
79 prints "----END LOG----";
```