Synthesis of Timing Parameters Satisfying Safety Properties

Étienne André and Romain Soulat

LSV, ENS Cachan & CNRS

Abstract. Safety properties are crucial when verifying real-time concurrent systems. When reasoning parametrically, i.e., with unknown constants, it is of high interest to infer a set of parameter valuations consistent with such safety properties. We present here algorithms based on the inverse method for parametric timed automata: given a reference parameter valuation, it infers a constraint such that, for any valuation satisfying this constraint, the discrete behavior of the system is the same as under the reference valuation in terms of traces, i.e., alternating sequences of locations and actions. These algorithms do not guarantee the equality of the trace sets, but are significantly quicker, synthesize larger sets of parameter valuations than the original method, and still preserve various properties including safety (i.e., non-reachability) properties. Those algorithms have been implemented in IMITATOR II and applied to various examples of asynchronous circuits and communication protocols.

Keywords: Real-Time Systems, Timed Automata, Verification, IMITATOR

1 Introduction

Timed Automata are finite-state automata augmented with clocks, i.e., real-valued variables increasing uniformly, that are compared within guards and transitions with timing delays [AD94]. Although techniques can be used in order to verify the correctness of a timed automaton for a given set of timing delays, these techniques become inefficient when verifying the system for a large number of sets of timing delays, and don't apply anymore when one wants to verify *dense* intervals of values, or optimize some of these delays. It is therefore interesting to reason parametrically, by assuming that those timing delays are unknown constants, or *parameters*, which give *Parametric Timed Automata (PTAs)* [AHV93].

We consider here the *good parameters problem* [FJK08]: "given a PTA $\mathcal{A}$ and a rectangular domain V bounding the value of each parameter, find all the parameter valuations within V such that $\mathcal{A}$ has a good behavior". Such good behaviors can refer to any kind of properties. We will in particular focus here on *safety properties*, i.e., the non-reachability of a given set of "bad" locations.

Parameters Synthesis for PTAs. The problem of parameter synthesis is known to be undecidable for PTAs, although semi-algorithms exist [AHV93] (i.e., if the algorithm terminates, the result is correct). The synthesis of constraints has been implemented in the context of PTAs or hybrid systems, e.g., in [AAB00] using TReX [CS01], or in [HRSV02] using an extension of Uppaal [LPY97] for linear parametric model checking. In [HRSV02], decidability results are given for a subclass of PTAs, viz., "L/U automata".

The problem of parameter synthesis for timed automata has been applied in particular to communication protocols (e.g., the Bounded Retransmission Protocol [DKRT97] using Uppaal and Spin [Hol03], and the Root Contention Protocol in [CS01] using TReX) and asynchronous circuits (see, e.g., [YKM02,CC07]).

The authors of [KP10] synthesize a set of parameter valuations under which a given property specified in the existential part of CTL without the next operator (ECTL_{-X}) holds in a system modeled by a network of PTAs. This is done by using bounded model checking techniques applied to PTAs.

In the framework of Linear Hybrid Automata, techniques based on *counterexample guided abstraction refinement* (CEGAR) [CGJ$^+$00] have been proposed. In [JKWC07], a method of iterative relaxation abstraction is proposed, combining CEGAR and linear programming. In [FJK08], when finding a counterexample, the system obtains constraints that *make* the counterexample infeasible. When all the counterexamples have been eliminated, the resulting constraints describe a set of parameters for which the system is safe. Also note that an approach similar to the inverse method is proposed in [AKRS08], in order to synthesize initial values for the variables of a linear hybrid system.

Contribution. We introduced in [ACEF09] the inverse method *IM* for PTAs. Different from CEGAR-based methods, this original semi-algorithm for parameter synthesis is based on a "good" parameter valuation π_0 instead of a set of "bad" states. *IM* synthesizes a constraint K_0 on the parameters such that, for all parameter valuation π satisfying K_0, the trace set, i.e., the discrete behavior, of $\mathcal{A}$ under π is the same as for $\mathcal{A}$ under π_0. This preserves in particular linear time properties. However, this equality of trace sets may be seen as a too strong property in practice. Indeed, one is rarely interested in a strict ordering of the events, but rather in the partial match with the original trace set, or more generally in the non-reachability of a given set of bad locations.

We present here several algorithms based on *IM*, which do not preserve the equality of trace sets, but preserve various properties. In particular, they all preserve non-reachability: if a location is not reachable in $\mathcal{A}$ under π_0, it will not be reachable in $\mathcal{A}$ under π, for π satisfying K_0. The main advantage is that these algorithms synthesize weaker constraints, i.e., larger sets of parameters. Beside, termination is improved when compared to the original *IM* and the computation time is reduced, as shown in practice in the implementation Imitator II.

Plan of the Paper. We briefly recall *IM* in Section 2. We introduce in Section 3 algorithms based on *IM* synthesizing weaker constraints for safety properties, and show their interest compared to *IM*. We extend in Section 4 these algorithms in order to perform a behavioral cartography of the system. We show in Section 5 the interest in practice by applying these algorithms to models of the literature. We also introduce algorithmic optimizations for two variants allowing to considerably reduce the state space. We conclude in Section 6.

2 The Inverse Method

Preliminaries. [1] Given a set X of clocks and a set P of parameters, a constraint C over X and P is a conjunction of linear inequalities on X and P. Given a parameter valuation (or point) π, we write $\pi \models C$ when the constraint where all parameters within C have been replaced by their value as in π is satisfied by a non-empty set of clock valuations. We denote by $\exists X : C$ the constraint over P obtained from C after elimination of the clocks in X.

Definition 1. *A PTA $\mathcal{A}$ is $(\Sigma, Q, q_0, X, P, K, I, \rightarrow)$ with Σ a finite set of actions, Q a finite set of locations, $q_0 \in Q$ the initial location, X a set of clocks, P a set of parameters, K a constraint over P, I the invariant assigning to every $q \in Q$ a constraint over X and P, and $\rightarrow$ a step relation consisting of elements (q, g, a, ρ, q'), where $q, q' \in Q$, $a \in \Sigma$, $\rho \subseteq X$ is the set of clocks to be reset, and the guard g is a constraint over X and P.*

The semantics of a PTA $\mathcal{A}$ is defined in terms of states, i.e., couples (q, C) where $q \in Q$ and C is a constraint over X and P. Given a point π, we say that a state (q, C) is π-compatible if $\pi \models C$. Runs are alternating sequences of states and actions, and traces are time-abstract runs, i.e., alternating sequences of *locations* and actions. The trace set of $\mathcal{A}$ corresponds to the traces associated with all the runs of $\mathcal{A}$. Given $\mathcal{A}$ and π, we denote by $\mathcal{A}[\pi]$ the (non-parametric) timed automaton where each occurrence of a parameter has been replaced by its constant value as in π. Given two states $s_1 = (q_1, C_1)$ and $s_1 = (q_2, C_2)$, we say that s_1 is *included* into s_2 if $q_1 = q_2$ and $C_1 \subseteq C_2$, where $\subseteq$ denotes the inclusion of constraints. One defines $Post^i_{\mathcal{A}(K)}(S)$ as the set of states reachable from a set S of states in exactly i steps under K, and $Post^*_{\mathcal{A}(K)}(S) = \bigcup_{i \geq 0} Post^i_{\mathcal{A}(K)}(S)$.

Description. Given a PTA $\mathcal{A}$ and a reference parameter valuation π_0, the inverse method *IM* synthesizes a constraint K_0 on the parameters such that, for all $\pi \models K_0$, $\mathcal{A}[\pi_0]$ and $\mathcal{A}[\pi]$ have the same trace sets [ACEF09]. We recall *IM* in Algorithm 1, which consists in two major steps.

1. The iterative removal of the π_0-incompatible states, i.e., states whose constraint onto the parameters is not satisfied by π_0, prevents for any $\pi \models K_0$ the behavior different from π_0 (by negating a π_0-incompatible inequality J).
2. The final intersection of the projection onto the parameters of the constraints associated with all the reachable states guarantees that all the behaviors under π_0 are allowed for all $\pi \models K_0$.

[1] Fully detailed definitions are available in [AS11].

Algorithm 1: Inverse method algorithm $\mathit{IM}(\mathcal{A}, \pi_0)$

```
input  : PTA $\mathcal{A}$ of initial state $s_0$, parameter valuation $\pi_0$
output: Constraint $K_0$ on the parameters
1 $i \leftarrow 0$ ;  $K \leftarrow$ true ;  $S \leftarrow \{s_0\}$
2 while true do
3     while there are $\pi_0$-incompatible states in $S$ do
4         Select a $\pi_0$-incompatible state $(q, C)$ of $S$ (i.e., s.t. $\pi_0 \not\models C$) ;
5         Select a $\pi_0$-incompatible $J$ in $(\exists X : C)$ (i.e., s.t. $\pi_0 \not\models J$) ;
6         $K \leftarrow K \land \neg J$ ;  $S \leftarrow \bigcup_{j=0}^{i} Post^j_{\mathcal{A}(K)}(\{s_0\})$ ;
7     if $Post_{\mathcal{A}(K)}(S) \sqsubseteq S$ then return $\bigcap_{(q,C) \in S} (\exists X : C)$
8     $i \leftarrow i + 1$ ;  $S \leftarrow S \cup Post_{\mathcal{A}(K)}(S)$ ;        // $S = \bigcup_{j=0}^{i} Post^j_{\mathcal{A}(K)}(\{s_0\})$
```

Item 1 is compulsory in order to prevent the system to enter "bad" (i.e., π_0-incompatible) states. However, item 2 can be lifted when one is only interested in safety properties. Indeed, in this case, it is acceptable that only part of the behavior of $\mathcal{A}[\pi_0]$ is available in $\mathcal{A}[\pi]$ (as long as the behavior absent from $\mathcal{A}[\pi_0]$ is also absent from $\mathcal{A}[\pi]$).

Properties. The main property of *IM* is the *preservation of trace sets*. As a consequence, linear-time properties are preserved. This is the case of properties expressed using the Linear Time Logics (LTL) [Pnu77], but also using the SE-LTL logics [CCO$^+$04], constituted by both atomic state propositions and events.

It has been shown that *IM* is *non-confluent*, i.e., several applications of *IM* can lead to different results [And10b]. This comes from the non-deterministic selection of a π_0-incompatible inequality J (line 5 in Algorithm 1). *IM* behaves deterministically when such a situation of choice is non encountered. The non-confluence of *IM* leads to the *non-maximality* of the output constraint. In other words, given $\mathcal{A}$ and π_0, there may exist points $\pi \not\models \mathit{IM}(\mathcal{A}, \pi_0)$ such that $\mathcal{A}[\pi]$ and $\mathcal{A}[\pi_0]$ have the same trace sets. However, it can be shown that, when *IM* is deterministic, the output constraint is maximal.

Reachability analysis is known to be undecidable for PTAs [AHV93]. Hence, although we showed sufficient conditions, *IM* does not terminate in general.

3 Optimized Algorithms Based on the Inverse Method

A drawback of *IM* is that the notion of equality of trace sets may be seen as too strict in some cases. If one is interested in the non-reachability of a certain set of bad states, then there may exist different trace sets avoiding this set of bad states. We introduce here several algorithms derived from *IM*: none of them guarantee the strict equality of trace sets, but all synthesize weaker constraints than *IM* and still feature interesting properties. They all preserve in particular safety properties, i.e., non-reachability of a given location. In other words, if a given "bad" location is not reached in $\mathcal{A}[\pi_0]$, it will also not be reached by $\mathcal{A}[\pi]$, for π satisfying the constraint output by the algorithm. The corollary is that the set of locations reachable in $\mathcal{A}[\pi]$ is included into the set reachable in $\mathcal{A}[\pi_0]$.

We introduce algorithms derived from IM, namely $IM_{\subseteq}$, $IM^{\cup}$, and IM^{K}. We then introduce combinations between these algorithms. For each algorithm, we briefly state that the constraint is weaker than IM (when applicable), study the termination, and formally state the properties guaranteed by the output constraint. (We do not recall the preservation of non-reachability.) The fully detailed algorithms and all formal properties with proofs can be found in [AS11].

3.1 Algorithm with State Inclusion in the Fixpoint

The algorithm $IM_{\subseteq}$ is obtained from IM by terminating the algorithm, not when all new states are *equal* to a state computed previously, but when all new states are *included* into a previous state.

The constraint output by $IM_{\subseteq}$ is weaker than the one output by IM, and $IM_{\subseteq}$ entails an earlier termination than IM for the same input, and hence a smaller memory usage because states are merged as soon as one is included into another one. $IM_{\subseteq}$ preserves the equality of traces up to length n, where n is the number of iterations of $IM_{\subseteq}$ (i.e., the depth of the state space exploration).

Proposition 1. *Suppose that $IM_{\subseteq}(\mathcal{A}, \pi_0)$ terminates with output K_0 after n iterations of the outer **do** loop. Then, we have:*

1. $\pi_0 \models K_0$,
2. *for all $\pi \models K_0$, for each trace T_0 of $\mathcal{A}[\pi_0]$, there exists a trace T of $\mathcal{A}[\pi]$ such that the prefix of length n of T_0 and the prefix of length n of T are equal,*
3. *for all $\pi \models K_0$, for each trace T of $\mathcal{A}[\pi]$, there exists a trace T_0 of $\mathcal{A}[\pi_0]$ such that the prefix of length n of T_0 and the prefix of length n of T are equal.*

Proposition 2. *Suppose that $IM_{\subseteq}(\mathcal{A}, \pi_0)$ terminates with output K_0. Then, for all $\pi \models K_0$, the sets of reachable locations of $\mathcal{A}[\pi]$ and $\mathcal{A}[\pi_0]$ are the same.*

3.2 Algorithm with Union of the Constraints

The algorithm $IM^{\cup}$ is obtained from IM by returning, not the intersection of the constraints associated with *all* the reachable states, but the *union* of the constraints associated with the *last* state of each run. This notion of last state is easy to understand for finite runs. When considering infinite (and necessarily[2] cyclic) runs, it refers to the second occurrence of a same state within a run, i.e., the first time that a state is equal to a previous state of the same run.

The constraint output by $IM^{\cup}$ is weaker than the one output by IM. Note that the constraints output by $IM_{\subseteq}$ and $IM^{\cup}$ are incomparable (see example in Section 3.6 for which two incomparable constraints are synthesized). The termination is the same as for IM.

[2] If the runs are infinite but not cyclic, the algorithm does not terminate.

Although the equality of trace sets is no longer guaranteed for $\pi \models IM^{\cup}(\mathcal{A}, \pi_0)$, we have the guarantee that, for all $\pi \models K_0$, the trace set of $\mathcal{A}[\pi]$ is a subset of the trace set of $\mathcal{A}[\pi_0]$. Furthermore, each trace of $\mathcal{A}[\pi_0]$ is reachable for *at least* one valuation $\pi \models K_0$.

Proposition 3. *Let* $K_0 = IM^{\cup}(\mathcal{A}, \pi_0)$*. Then:*

1. $\pi_0 \models K_0$*;*
2. *For all* $\pi \models K_0$*, every trace of* $\mathcal{A}[\pi]$ *is equal to a trace of* $\mathcal{A}[\pi_0]$*;*
3. *For all trace* T *of* $\mathcal{A}[\pi_0]$*, there exists* $\pi \models K_0$ *such that the trace set of* $\mathcal{A}[\pi]$ *contains* T*.*

Finally note that, due to the disjunctive form of the returned constraint, the synthesized constraint is not necessarily convex.

3.3 Algorithm with Direct Return

The algorithm IM^K is obtained from IM by returning only the constraint K computed during the algorithm instead of the intersection of the constraints associated to all the reachable states.

The constraint output by IM^K is weaker than the one output by IM. Also note that the constraints output by $IM_{\subseteq}$ and IM^K are incomparable (see example in Section 3.6). Termination is the same for IM^K and IM.

Proposition 4. *Let* $K_0 = IM^K(\mathcal{A}, \pi_0)$*. Then, for all* $\pi \models K_0$*, every trace of* $\mathcal{A}[\pi]$ *is equal to a trace of* $\mathcal{A}[\pi_0]$*.*

This algorithm only prevents π_0-incompatible states to be reached but, contrarily to IM and $IM^{\cup}$, does not guarantee that any "good" state will be reached. Hence, this algorithm only preserves the non-reachability of locations.

3.4 Combination: Inclusion in Fixpoint and Union

One combine the variant of the fixpoint (viz., $IM_{\subseteq}$) with the first variant of the constraint output (viz., $IM^{\cup}$), thus leading to $IM^{\cup}_{\subseteq}$. The constraint output by $IM^{\cup}$ is weaker than the ones output by both $IM_{\subseteq}$ and $IM^{\cup}$. Note that the constraints output by $IM^{\cup}_{\subseteq}$ and IM^K are incomparable (see example in Section 3.6 for which two incomparable constraints are synthesized). The termination is the same as for $IM_{\subseteq}$. This algorithm combines the properties of $IM_{\subseteq}$ and $IM^{\cup}$. Although not of high interest in practice, this result implies preservation of non-reachability. Finally note that, due to the disjunctive form of the returned constraint, the output constraint is not necessarily convex.

3.5 Combination: Inclusion in Fixpoint and Direct Return

One can also combine the variant of the fixpoint (viz., $IM_{\subseteq}$) with the second variant of the constraint output (viz., IM^K), thus leading to $IM^K_{\subseteq}$. The constraint output by $IM^K_{\subseteq}$ is weaker than the ones output by both IM^K and $IM^{\cup}_{\subseteq}$. Termination is the same as for $IM_{\subseteq}$. This algorithm only preserves the non-reachability of locations.

3.6 Summary of the Algorithms

We summarize in Table 1 the properties of each algorithm.

Property	IM	$IM_{\subseteq}$	$IM^{\cup}$	IM^{K}	$IM^{\cup}_{\subseteq}$	$IM^{K}_{\subseteq}$
Equality of trace sets	√	×	×	×	×	×
Equality of trace sets up to n	√	√	×	×	×	×
Inclusion into the trace set of $\mathcal{A}[\pi_0]$	√	×	√	√	×	×
Preservation of at least one trace	√	×	√	×	×	×
Equality of location sets	√	√	×	×	×	×
Convex output	√	√	×	√	×	√
Preservation of non-reachability	√	√	√	√	√	√

Table 1. Comparison of the properties of the variants of *IM*

We give in Figure 1 (left) the relation between terminations: an oriented edge from A to B means that, for the same input, termination of variant A implies termination of B. We give in Figure 1 (right) the relations between the constraints synthesized by each variant: for example, given $\mathcal{A}$ and π_0, we have that $IM(\mathcal{A}, \pi_0) \subseteq IM_{\subseteq}(\mathcal{A}, \pi_0)$. Obviously, the weakest constraint is the one synthesized by $IM^{K}_{\subseteq}$. This variant should be thus used when one is interested only in safety properties; however, when one is interested in stronger properties (e.g., preservation of at least one trace of $\mathcal{A}[\pi_0]$), one may want to use another variant according to their respective properties. We believe that the most interesting algorithms are IM, for the equality of trace sets, $IM^{\cup}$, for the preservation of at least one maximal trace, and $IM^{K}_{\subseteq}$, for the sole preservation of non-reachability.

$$\begin{tikzcd}
IM \arrow[r, Leftrightarrow] \arrow[d, Rightarrow] & IM^{\cup} \arrow[r, Leftrightarrow] & IM^{K} \\
IM_{\subseteq} \arrow[r, Leftrightarrow] & IM^{\cup}_{\subseteq} \arrow[r, Leftrightarrow] & IM^{K}_{\subseteq}
\end{tikzcd}$$

$$\begin{tikzcd}
 & IM_{\subseteq} \arrow[dr, dash, dashed, "\subseteq" sloped] & & \\
IM \arrow[ur, dash, dashed, "\subseteq" sloped] \arrow[dr, dash, dashed, "\subseteq" sloped] & & IM^{\cup}_{\subseteq} \arrow[r, dash, dashed, "\subseteq"] & IM^{K}_{\subseteq} \\
 & IM^{\cup} \arrow[ur, dash, dashed, "\subseteq" sloped] \arrow[r, dash, dashed, "\subseteq"] & IM^{K} \arrow[ur, dash, dashed, "\subseteq" sloped] &
\end{tikzcd}$$

Fig. 1. Comparison of termination (left) and constraint output (right)

Non-maximality. Actually, none of these algorithms synthesize the maximal constraint corresponding to the property they are characterized with. This is due to their non-confluence, itself due to the random selection of a π_0-incompatible inequality. However, it can be shown that the constraint is maximal when the algorithm runs in a fully deterministic way. We address the issue of synthesizing a maximal constraint in Section 4. Also note that the comparison between the constraints (see Figure 1 (right)) holds only for deterministic analyses.

Comparison Using an Example of PTA. Let us consider the PTA $\mathcal{A}_{var}$ depicted below. We consider the following π_0: $p_1 = 1 \wedge p_2 = 4$. In $\mathcal{A}[\pi_0]$, location q_4 is not reachable, and can be considered as a "bad" location.

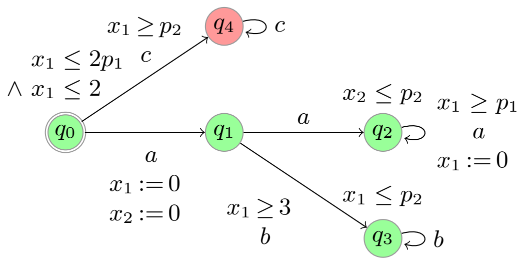

Fig. 2. A PTA $\mathcal{A}_{var}$ for comparing the variants of *IM*

Let us suppose that a bad behavior of $\mathcal{A}_{var}$ corresponds to the fact that a trace goes into location q_4. Under π_0, the system has a good behavior. As a consequence, by the property of non-reachability of a location met by all algorithms, the constraint synthesized by any algorithm also prevents the traces to enter q_4. One can show that the parameter valuations allowing the system to enter the bad location q_4 are comprised in the domain $2 * p_1 \leq p_2 \wedge p_2 \leq 2$. As a consequence, the (non-convex) maximal set of parameters avoiding the bad location q_4 is $2 * p_1 > p_2 \vee p_2 > 2$.

We give below the six constraints synthesized by the six versions of the inverse method. For each graphics, we depict in dark blue the parameter domain covered by the constraint, and in light red the parameter domain corresponding to a bad behavior (the constraint itself is given in [AS11]). The "good" zone not covered by the constraint is depicted in very light gray. The dot represents π_0.

This example illustrates well the relationship between the different constraints. In particular, the constraint synthesized by $IM^K_\subseteq$ dramatically improves the set of parameters synthesized by *IM*. Also note that we chose on purpose an example such that none of the methods synthesizes a maximal constraint (observe that even $IM^K_\subseteq$ does not cover the whole "good" zone). This will be addressed in Section 4.

Experimental Validation. The example above shows clearly the gain of the algorithms w.r.t. *IM*. However, for real case studies, although checking the gain of these algorithms in terms of computation time is possible, measuring the gain in terms of the "size" of the constraints synthesized requires measures of polyhedra, which is not trivial when they are non-convex. Hence, we postpone this study to the framework of the cartography (see Section 5).

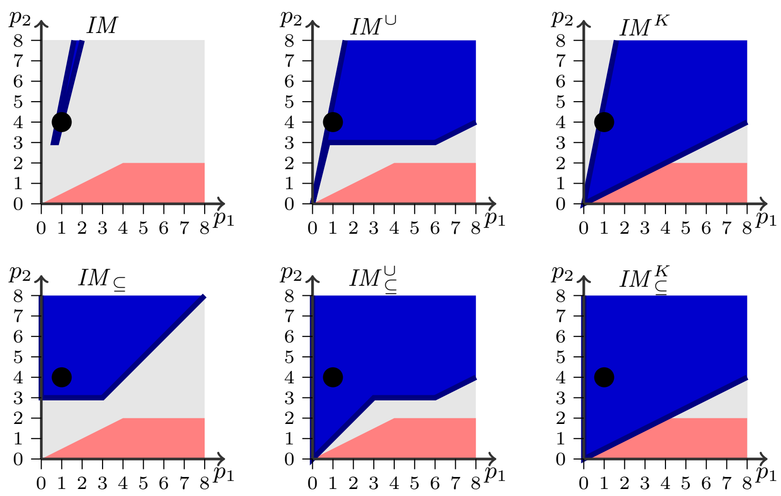

Fig. 3. Comparison of the constraints synthesized for $\mathcal{A}_{var}$

4 Behavioral Cartography

Although *IM* has been shown of interest for a large panel of case studies, its main shortcoming is the non-maximality of the output constraint. Moreover, the good parameters problem relates to the synthesis of parameter valuations corresponding to *any* good behavior, not to a single one.

The behavioral cartography algorithm *BC* relies on the idea of covering the parameter space within a rectangular real-valued parameter domain V_0 [AF10]. By iterating *IM* over all the *integer* points of V_0 (of which there are a finite number), one is able to decompose V_0 into a list *Tiling* of *tiles*, i.e., dense sets of parameters in which the trace sets are the same.

Then, given a property φ on trace sets (viz., a linear time property), one can partition the parameter space between good tiles (for which all points satisfy φ) and bad tiles. This can be done by checking the property for one point in each tile (using, e.g., UPPAAL, applied to the PTA instantiated with the considered point). Then the set of parameters satisfying φ corresponds to the union of the good tiles. Note that *BC* is independent from φ; only the partition between good and bad tiles involves φ.

In practice, not only the integer valuations of V_0 are covered by *Tiling*, but also most of the real-valued space of V_0. Furthermore, the space covered by *Tiling* often largely exceeds the limits of V_0. However, there may exist a finite number of "small holes" within V_0 (containing no integer point) that are not covered by any tile of *Tiling*. A refinement of *BC* is to consider a tighter grid, i.e., not only integer points, but rational points multiple of a smaller step than 1. We showed that, for a rectangle V_0 large enough and a grid tight enough, the full coverage of the whole real-valued parameter space (inside and outside V_0) is ensured for some classes of PTAs, in particular for acyclic systems (see [And10b] for details).

Combination with the Variants. By replacing within BC the call to IM by a call to one of the algorithms introduced in Section 3, one changes the properties of the tiles: for each tile, the corresponding trace set inherits the properties of the considered variant, and does not necessarily preserve the equality of trace sets. However, as shown in Section 3, they all preserve (at least) the non-reachability.

The main advantage of the combination of BC with one of the algorithms, say IM', of Section 3 is that the coverage of V_0 needs *a smaller number of tiles*, i.e., of calls to IM'. Indeed, due to the weaker constraint synthesized by IM', and hence larger sets of parameters, one needs less calls to IM' in order to cover V_0. Furthermore, due to the quicker termination of IM' when compared to IM, the computation time decreases considerably. Finally, due to an earlier termination IM' (i.e., less states computed) and the lower number of calls to IM' (hence, less trace sets to remember), the memory consumption also decreases.

5 Implementation and Experiments

All these algorithms, as well as the original IM, have been implemented in IMITATOR II [And10a]. We give in Table 2 the summary of various experiments of parameter synthesis applied to case studies from the literature as well as industrial case studies. For each case study, we apply each version of BC to a given V_0. Then, we split the tiles between good and bad w.r.t. a property. Finally, we synthesize a constraint corresponding to this property. For each case study, V_0 is either entirely covered, or "almost entirely covered". We give from left to right the name of the case study, the number of parameters varying in the cartography and the number of points within V_0. We then give the number of tiles and the computation time for each algorithm. We denote by $BC_{\subseteq}$ the variant of BC calling $IM_{\subseteq}$ instead of IM (and similarly for the other algorithms). All experiments were performed on an Intel Core 2 Duo 2,33 Ghz with 3,2 Go memory, using the `no-random`, `no-dot` and `no-log` options of IMITATOR II.

Example			Tiles						Time (s)					
Name	$\lvert P \rvert$	$\lvert V_0 \rvert$	BC	$BC^{\cup}$	BC^{K}	$BC_{\subseteq}$	$BC_{\subseteq}^{\cup}$	$BC_{\subseteq}^{K}$	BC	$BC^{\cup}$	BC^{K}	$BC_{\subseteq}$	$BC_{\subseteq}^{\cup}$	$BC_{\subseteq}^{K}$
$\mathcal{A}_{var}$	2	72	14	10	10	7	5	5	0.101	0.079	0.073	0.036	0.028	0.026
Flip-flop	2	644	8	7	7	8	7	7	0.823	0.855	0.696	0.831	0.848	0.699
AND–OR	5	151 200	16	14	16	14	14	14	274	7154	105	199	551	68.4
Latch	4	73 062	5	3	3	5	3	3	16.2	25.2	9.2	15.9	25	9.1
CSMA/CD	3	2 000	139	57	57	139	57	57	112	276	76.0	46.7	88.0	22.6
SPSMALL	2	3 082	272	78	77	272	78	77	894	405	342	894	406	340

Table 2. Comparison of the algorithms for the behavioral cartography

Since V_0 is always (at least "almost") entirely covered by *Tiling*, the number of tiles needed to cover V_0 gives a measure of the size of each tile in terms of parameter valuations: the lesser tiles needed, the larger the sets of parameter valuations are, the more efficient the corresponding algorithm is. Since the good property for all case studies is a property of (non-)reachability, the constraint computed is the same for all versions of BC. The latest version[3] of IMITATOR II as well as all the mentioned case studies can be found on IMITATOR II's Web page[4]. Details on case studies can be found in [AS11].

As expected from Section 2, all algorithms bring a significant gain in term of size of the constraint, because the number of tiles needed to cover V_0 is almost always smaller than for IM. Only $IM_{\subseteq}$ has a number of tiles often equal to IM; however, the computation is often much quicker for $IM_{\subseteq}$. As expected, the most efficient algorithm is $IM_{\subseteq}^{K}$: both the number of tiles and the computation times decrease significantly. When one is only interested in reachability properties, one should then use this algorithm.

The only surprising result is the fact that $IM^{\cup}$ is sometimes slower than IM, although the number of tiles is smaller. This is due to the way it is implemented in IMITATOR II. Handling non-convex constraints is a difficult problem; hence, we compute a list of constraints associated with the last state of each trace. Unfortunately, many of these constraints are actually equal to each other. For systems with thousands of traces and hundreds of tiles, we manipulate hundreds of thousands of constraints; every time a new point is picked up, one should check whether it belongs to this huge set before calling (or not) IM on this point. This also explains the relatively disappointing speed performance of $IM_{\subseteq}^{\cup}$. Improving this implementation is the subject of ongoing work. A possible option would be to remove the constraints equal to each other in this constraint set; this would dramatically decrease the size of the set, but would induce additional costs for checking constraint equality.

On-the-fly Computation of K. We finally introduce here another modification of some of the algorithms in order to avoid the non-necessary duplication of some reachable states, leading to a dramatic diminution of the state space. Indeed, we met cases where two states (q, C) and (q, C') are not equal at the time they are computed, but are equal with the final intersection K of the constraints, i.e. $(q, C \wedge K) = (q, C' \wedge K)$. Such a situation is depicted in the trace sets of Figure 4, where identical states under $IM(\mathcal{A}, \pi_0)$ are unmerged on the left part of the figure and merged on the right part. We can solve this problem by performing *dynamically* the intersection of the constraints, i.e., adding $\exists X : C$ to all the states previously computed, every time a new state (q, C) is computed. This has the effect of merging such states, and hence often considerably decreasing the state space. With this modification, the algorithm only needs to return K at the end of the computation, since the intersection is performed on the fly. We give this algorithm IM_{otf} in Algorithm 2.

This modification can be extended to $IM_{\subseteq}$ in a straightforward manner, by applying to IM_{otf} the fixpoint modification as described in Section 3.1. However, applying it to other algorithms would modify their correctness, since the final intersection of the constraints is not performed in the other algorithms.

[3] Note that the software named IMITATOR 3 is an independent fork of IMITATOR II for hybrid systems [FK11]. The latest version of IMITATOR for PTAs is IMITATOR 2.3.

[4] `http://www.lsv.ens-cachan.fr/Software/imitator/imitator2.3/`

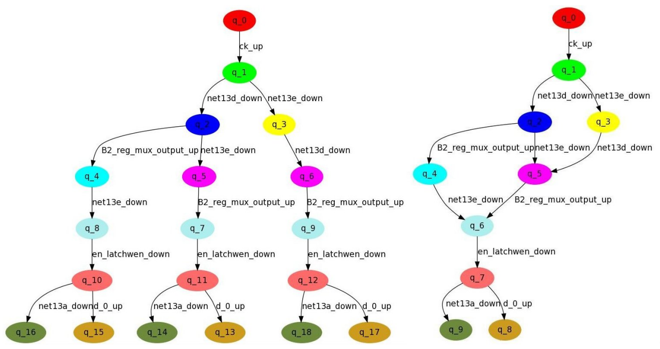

Fig. 4. Example of state space explosion due to unmerged states

Algorithm 2: $IM_{otf}(\mathcal{A}, \pi_0)$

```
input  : PTA $\mathcal{A}$ of initial state $s_0$, parameter valuation $\pi_0$
output: Constraint $K_0$ on the parameters
1  $i \leftarrow 0$;  $K \leftarrow$ true;  $S \leftarrow \{s_0\}$
2  while true do
3      foreach $s = (q, C) \in S$ do
4          if $s$ is $\pi_0$-incompatible then
5              Select a $\pi_0$-incompatible $J$ in $(\exists X : C)$
6              $K \leftarrow K \wedge \neg J$ ;
7              foreach $(q', C') \in S$ do
8                  $C' \leftarrow C' \wedge \neg J$
9          else
10             $K \leftarrow K \wedge \exists X : C$ ;
11             foreach $(q', C') \in S$ do
12                 $C' \leftarrow C' \wedge \exists X : C$
13     if $Post_{\mathcal{A}(K)}(S) \sqsubseteq S$ then return $K$
14     $i \leftarrow i + 1$;  $S \leftarrow S \cup Post_{\mathcal{A}(K)}(S)$ ;        // $S = \bigcup_{j=0}^{i} Post^{j}_{\mathcal{A}(K)}(\{s_0\})$
```

Using this modification, we successfully computed a set of parameters for the SPSMALL memory designed by ST-Microelectronics. We analyzed a much larger version of the "small" model considered above (see differences between these models in [And10b]). The larger model of this memory contains 28 clocks and 62 parameters. The computation consists in 98 iterations of the outer **DO** loop of *IM*. Without this optimization, *IM* crashed from lack of memory at iteration 27 (on a 2 GB memory machine), but the size of the state space was exponential, so we believe that the full computation would have required a huge amount of memory, preventing more powerful machine to perform the computation. Using this optimization, we computed quickly a constraint, made of the conjunction of 49 linear constraints. Full details are available in [And10b].

6 Conclusion

We introduced here several algorithms based on the inverse method for PTAs. Given a PTA $\mathcal{A}$ and a reference parameter valuation π_0, these algorithms synthesize a constraint K_0 around π_0, all preserving non-reachability properties: if a location (in general "bad") is not reached for π_0, it is also not reachable for any $\pi \models K_0$. Furthermore, each algorithm preserves different properties: strict equality of trace sets, inclusion within the trace set of $\mathcal{A}[\pi_0]$, preservation of at least one trace of $\mathcal{A}[\pi_0]$, etc. The major advantage of these variants is the faster computation of K_0, and a larger set of parameter valuations defined by K_0. These algorithms have been implemented in IMITATOR II and show significant gains of time and size of the constraint when compared to the original *IM*. When used in the behavioral cartography for synthesizing a constraint w.r.t. a given property, they cover both using less tiles and in general faster the parameter space.

Also recall that, although the algorithms preserve properties based on traces, i.e., *untimed* behaviors, it is possible to synthesize constraints guaranteeing *timed* properties by making use of an observer; this is the case in particular for the SPSMALL memory.

As a future work, the inverse method and the cartography algorithm, as well as the variants introduced here, could be extended in a rather straightforward way to the backward case.

Furthermore, we presented in [AFS09] an extension of the inverse method to probabilistic systems: given a parametric probabilistic timed automaton $\mathcal{A}$ and a reference valuation π_0, we synthesize a constraint K_0 by applying *IM* to a non-probabilistic version of $\mathcal{A}$ and π_0. Then, we guarantee that, for all $\pi \models K_0$, the values of the minimum (resp. maximum) probabilities of reachability properties are the same in $\mathcal{A}[\pi]$. Studying what properties each of the algorithms presented here preserves in the probabilistic framework is the subject of ongoing work.

It would also be of interest to consider the combination of these algorithms with the extension of the inverse method to linear hybrid automata [FK11].

References

[AAB00] A. Annichini, E. Asarin, and A. Bouajjani. Symbolic techniques for parametric reasoning about counter and clock systems. In *CAV'00*, pages 419–434. Springer-Verlag, 2000.

[ACEF09] É. André, T. Chatain, E. Encrenaz, and L. Fribourg. An inverse method for parametric timed automata. *International Journal of Foundations of Computer Science*, 20(5):819–836, 2009.

[AD94] R. Alur and D.L. Dill. A theory of timed automata. *TCS*, 126(2):183–235, 1994.

[AF10] É. André and L. Fribourg. Behavioral cartography of timed automata. In *RP'10*, volume 6227 of *LNCS*, pages 76–90. Springer, 2010.

[AFS09] É. André, L. Fribourg, and J. Sproston. An extension of the inverse method to probabilistic timed automata. In *AVoCS'09*, volume 23 of *Electronic Communications of the EASST*, 2009.

[AHV93] R. Alur, T. A. Henzinger, and M. Y. Vardi. Parametric real-time reasoning. In *STOC'93*, pages 592–601. ACM, 1993.

[AKRS08] R. Alur, A. Kanade, S. Ramesh, and K. C. Shashidhar. Symbolic analysis for improving simulation coverage of simulink/stateflow models. In *EMSOFT'08*, pages 89–98. ACM, 2008.

[And10a] Étienne André. IMITATOR II: A tool for solving the good parameters problem in timed automata. In *INFINITY'10*, volume 39 of *EPTCS*, pages 91–99, 2010.

[And10b] Étienne André. *An Inverse Method for the Synthesis of Timing Parameters in Concurrent Systems.* Ph.d. thesis, Laboratoire Spécification et Vérification, ENS Cachan, France, 2010.

[AS11] É. André and R. Soulat. Synthesis of timing parameters satisfying safety properties (full version). Research report, Laboratoire Spécification et Vérification, ENS Cachan, France, 2011. Available at `http://www.lsv.ens-cachan.fr/Publis/RAPPORTS_LSV/PDF/rr-lsv-2011-13.pdf`.

[CC07] R. Clarisó and J. Cortadella. The octahedron abstract domain. *Sci. Comput. Program.*, 64(1):115–139, 2007.

[CCO$^+$04] S. Chaki, E. M. Clarke, J. Ouaknine, N. Sharygina, and N. Sinha. State/event-based software model checking. In *IFM'04*, volume 2999 of *LNCS*, pages 128–147. Springer, 2004.

[CGJ$^+$00] 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.

[CS01] A. Collomb–Annichini and M. Sighireanu. Parameterized reachability analysis of the IEEE 1394 Root Contention Protocol using TReX. In *RT-TOOLS'01*, 2001.

[DKRT97] 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.

[FJK08] 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.

[FK11] L. Fribourg and U. Kühne. Parametric verification and test coverage for hybrid automata using the inverse method. In *RP'11*, volume 6945 of *LNCS*. Springer, 2011. To appear.

[Hol03] Gerard Holzmann. *The Spin model checker: primer and reference manual.* Addison-Wesley Professional, 2003.

[HRSV02] 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.

[JKWC07] S.K. Jha, B.H. Krogh, J.E. Weimer, and E.M. Clarke. Reachability for linear hybrid automata using iterative relaxation abstraction. In *HSCC'07*, pages 287–300. Springer-Verlag, 2007.

[KP10] M. Knapik and W. Penczek. Bounded model checking for parametric time automata. In *SUMo'10*, 2010.

[LPY97] K.G. Larsen, P. Pettersson, and W. Yi. Uppaal in a nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1-2):134–152, 1997.

[Pnu77] Amir Pnueli. The temporal logic of programs. In *SFCS'77*, pages 46–57. IEEE Computer Society, 1977.

[YKM02] T. Yoneda, T. Kitai, and C. J. Myers. Automatic derivation of timing constraints by failure analysis. In *CAV'02*, pages 195–208. Springer-Verlag, 2002.