Synthesis of Timing Parameters Satisfying Safety Properties

Étienne André and Romain Soulat

May 2011

Research report LSV-11-13

Synthesis of Timing Parameters Satisfying Safety Properties (full version)

Étienne André[†] and Romain Soulat[‡]

[†] School of Computing, National University of Singapore
[‡] LSV, ENS de Cachan & CNRS, France

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 system behaves the same as under the reference valuation in terms of traces, i.e., alternating sequences of locations and actions. Those algorithms do not guarantee the equality of traces, but are significantly quicker, synthesize larger sets of parameter valuations than the original inverse method, and still preserve various properties including safety properties. Those algorithms have been implemented in IMITATOR II and applied to examples of circuits and protocols.

1 Introduction

Timed Automata are finite-state automata augmented with a set of clocks, i.e., real-valued variables increasing uniformly, that are compared within guards and transitions with timing delays [AD94]. This formalism has become widely used in the verification of real-time concurrent systems, in particular using the UPPAAL model checker [LPY97]. 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 those 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 on the system. We will in particular focus here on *safety properties*, or the impossibility to reach 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]. The synthesis of constraints has been implemented in the context of PTAs or hybrid systems, e.g., in [AAB00] using tool TReX [CS01], or in [HRSV02] using an extension of Uppaal [LPY97] for linear parametric model checking. Note that [AAB00] is able to infer non-linear constraints. In [HRSV02], decidability results are given for the verification of a special class, called "L/U automata". Two subclasses of L/U automata, called lower-bound and upper-bound PTAs, are also considered in [WY03], with decidability results.

The problem of parameter synthesis for timed automata has been applied in particular to two main domains: telecommunication protocols and asynchronous circuits. For example, concerning telecommunication protocols, the Bounded Retransmission Protocol has been verified in [DKRT97] using Uppaal [LPY97] and Spin [Hol03], and the Root Contention Protocol in [CS01] using TReX [ABS01]. The synthesis of constraints has also been studied more specifically in the context of asynchronous circuits, mainly by Myers and co-workers (see, e.g., [YKM02]), and by Clarisó and Cortadella (see, e.g., [CC05,CC07]), who have proposed methods with approximations. They also proceed by analyzing failure traces and generating timing constraints that prevent the occurrence of such failures.

In [FJK08], the authors propose an extension based on the *counterexample guided abstraction refinement* (CEGAR) [CGJ$^+$00]. When finding a counterexample, the system obtains constraints on the parameters 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.

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.

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 instead of a set of "bad" states. Given a reference parameter valuation π_0, *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 into 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, these algorithms 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 of considering weaker properties is that these algorithms output weaker constraints, i.e., larger set of parameters. Beside, termination is improved when compared to the original inverse method and the computation time is reduced. These algorithms have been implemented in IMITATOR II, and we show that some of these algorithms behave much more efficiently than *IM*, by synthesizing significantly larger sets of parameters, and terminating quicker.

Plan of the Paper. We introduce preliminaries in Section 2. We briefly recall *IM* in Section 3. We introduce in Section 4 algorithms based on *IM* synthesizing weaker constraints for safety properties. We extend in Section 5 these algorithms in order to perform a behavioral cartography of the system, and show their interest compared to *IM*. We show in Section 6 the interest in practice by applying these algorithms to models of the literature. We finally introduce in Section 7 algorithmic optimizations for two variants allowing to considerably reduce the state space. We conclude in Section 8.

2 Preliminaries

2.1 Clocks

Let $\mathbb{R}_{\geq 0}$ be the set of non-negative real numbers.

Throughout this paper, we assume a fixed set $X = \{x_1, \dots, x_H\}$ of *clocks*. A clock is a variable x_i with value in $\mathbb{R}_{\geq 0}$. All clocks evolve linearly at the same rate.

We define a *clock valuation* as a function $w : X \to \mathbb{R}_{\geq 0}$ assigning a non-negative real value to each clock. We will often identify a valuation w with the point $(w(x_1), \dots, w(x_H))$.

Given a constant $d \in \mathbb{R}_{\geq 0}$, we use $X+d$ to denote the set $\{x_1+d, \dots, x_H+d\}$. Similarly, we write $w+d$ to denote the valuation such that $(w+d)(x) = w(x)+d$ for all $x \in X$.

2.2 Parameters

Throughout this paper, we assume a fixed set $P = \{p_1, \dots, p_M\}$ of *parameters*, i.e., unknown constants. Such parameters are disjoint with the sets of clocks and variables.

A *parameter valuation* π is a function $\pi : P \to \mathbb{R}_{\geq 0}$ assigning a nonnegative real value to each parameter. There is a one-to-one correspondence between valuations and points in $(\mathbb{R}_{\geq 0})^M$. We will often identify a valuation π with the point $(\pi(p_1), \dots, \pi(p_M))$.

2.3 Constraints

Definition 1 (Linear inequality). *Let V be a set of variables of the form $V = \{v_1, \dots, v_N\}$. A* linear inequality on V *is an inequality $e \prec e'$, where $\prec \in \{<, \leq\}$,* *and e, e' are two linear terms of the form*

$$\sum_{1 \leq i \leq N} \alpha_i v_i + d$$

where $v_i \in V$, $\alpha_i \in \mathbb{Q}_{\geq 0}$, for $1 \leq i \leq N$, and $d \in \mathbb{Q}_{\geq 0}$.

Note that we define the coefficients of the linear inequalities as *positive rationals*. It would of course be equivalent to define them as *positive integers*, as it is sometimes the case in the literature.

We assume in the following that all inequalities are linear, and we will simply refer to linear inequalities as *inequalities*.

Definition 2 (Negation of an inequality). *Let V be a set of variables of the form $V = \{v_1, \dots, v_N\}$. Given an inequality J on V of the form $e < e'$ (resp. $e \leq e'$), the* negation *of J, denoted by $\neg J$, is the linear inequality $e' \leq e$ (resp. $e' < e$).*

Definition 3 (Convex linear constraint). *Let V be a set of variables of the form $V = \{v_1, \dots, v_N\}$. A* convex linear constraint *on V is a conjunction of inequalities on V.*

We assume in the following that all constraints are both convex and linear, and we will simply refer to convex linear constraints as *constraints*.

Definition 4 (Constraint over the clocks). *An* inequality over the clocks *is an inequality on X. A* constraint over the clocks *is a constraint on X.*

Definition 5 (Constraint over the parameters). *An* inequality over the parameters *is an inequality on P. A* constraint over the parameters *is a constraint on P.*

Definition 6 (Constraint over the clocks and parameters). *An* inequality over the clocks and the parameters *is an inequality on $X \cup P$. A* constraint over the clocks and the parameters *is a constraint on $X \cup P$.*

Throughout this paper, given a set X of clocks and a set P of parameters, we will denote by $\mathcal{K}_X$ the set of all constraints over the clocks, by $\mathcal{K}_P$ the set of all constraints over the parameters, and by $\mathcal{K}_{X \cup P}$ the set of all constraints over the clocks and the parameters.

In the sequel, the letter J will denote an inequality over the parameters, the letter D will denote a constraint over the clocks, the letter K will denote a constraint over the parameters, and the letter C will denote a constraint over the clocks and the parameters.

Semantics of Constraints. Given a constraint D over the clocks and a clock valuation w, $D[w]$ denotes the expression obtained by replacing each clock x in D with $w(x)$. A clock valuation w *satisfies* constraint D (denoted by $w \models D$) if $D[w]$ evaluates to true.

Given a parameter valuation π and a constraint C over the clocks and the parameters, $C[\pi]$ denotes the constraint over the clocks obtained by replacing each parameter p in C with $\pi(p)$. Likewise, given a clock valuation w, $C[\pi][w]$ denotes the expression obtained by replacing each clock x in $C[\pi]$ with $w(x)$. We say that a parameter valuation π *satisfies* a constraint C, denoted by $\pi \models C$, if the set of clock valuations that satisfy $C[\pi]$ is nonempty. We use the notation $<w, \pi> \models C$ to indicate that $C[\pi][w]$ evaluates to true.

Given two constraints C_1 and C_2 over the clocks and the parameters, we say that C_1 is *included in* C_2, denoted by $C_1 \subseteq C_2$, if $\forall w, \pi : <w, \pi> \models C_1 \Rightarrow <w, \pi> \models C_2$. We have that $C_1 = C_2$ if and only if $C_1 \subseteq C_2$ and $C_2 \subseteq C_1$.

Similarly to the semantics of constraints over the clocks and the parameters, we say that a parameter valuation π *satisfies* a constraint K over the parameters, denoted by $\pi \models K$, if the expression obtained by replacing each parameter p in K with $\pi(p)$ evaluates to true. Given two constraints K_1 and K_2 over the parameters, we say that K_1 is *included in* K_2, denoted by $K_1 \subseteq K_2$, if $\forall \pi : \pi \models K_1 \Rightarrow \pi \models K_2$. We often say that K_2 is *weaker* than K_1, in the sense that it corresponds to a larger set of parameter valuations. We have that $K_1 = K_2$ if and only if $K_1 \subseteq K_2$ and $K_2 \subseteq K_1$.

Depending on the context, we will consider `true` as a constraint over the parameters (resp. as a constraint over the clocks and parameters), corresponding to the set of all possible values for P (resp. for X and P).

Operations on Constraints. We define in the following operations on constraints over the clocks and parameters.

Given a constraint C over the clocks and the parameters, and a set $X' \subseteq X$ of clocks, we denote by $\exists X' : C$ the constraint over the clocks and parameters obtained from C after *elimination* of the clocks in X'. Formally, recall that $X = \{x_1, \ldots, x_H\}$. Let $I_{X'}$ be the set of the indexes of the clocks appearing in X' (i.e., $I_{X'} \subseteq \{1, \ldots, H\}$). Let w (resp. w') denote a valuation of X (resp. X'). Let w'' denote a valuation of $X \setminus X'$. Then, $\exists X' : C = \{<w'', \pi> \mid \exists w', w : \forall i \in I_{X'}, w(x_i) = w'(x_i) \wedge \forall i \notin I_{X'}, w(x_i) = w''(x_i) \wedge <w, \pi> \models C\}$. Similarly, we denote by $\exists X : C$ the constraint over the parameters obtained from C after elimination of all the clocks (i.e., $\{\pi \mid \exists w : <w, \pi> \models C\}$).

Given a constraint C over the clocks and the parameters, and a set $X' \subseteq X$ of clocks, we denote by $C_{/X'}$ the constraint over the parameters obtained from C after *projection* onto the clocks of X', i.e., $\exists (X \setminus X') : C$.

Given a constraint C over the clocks and the parameters, we define the *time elapsing* of C, denoted by $C^{\uparrow}$, as the constraint obtained from C by delaying an arbitrary amount of time, i.e., by renaming X' with X in the following expression:

$$\exists X, d : C \wedge X' = X + d$$

where d is a new parameter with values in $\mathbb{R}_{\geq 0}$, and X' is a fresh set of clocks.

2.4 Labeled Transition Systems

We finally introduce labeled transition systems, which will be later used to represent the semantics of Parametric Timed Automata.

Definition 7 (LTS). *A* labeled transition system (LTS) *is a tuple* $\mathcal{L} = (S, s_{init}, \Sigma, \Rightarrow)$ *where*

- S *is a set of* states*,*
- $s_{init} \in S$ *is an* initial state*,*
- Σ *is a* set of symbols*,*
- *and* $\Rightarrow : S \times \Sigma \times S$ *is a* labeled transition relation*.*

We write $s \overset{a}{\Rightarrow} s'$ *for* $(s, a, s') \in \Rightarrow$*. A* run *of* $\mathcal{L}$ *is an infinite*[1] *alternating sequence of states* $s_i \in S$ *and symbols* $a_i \in \Sigma$ *of the form* $<s_0, a_0, s_1, a_1, \cdots>$ *such that* $s_0 = s_{init}$ *and* $s_i \overset{a_i}{\Rightarrow} s_{i+1}$ *for all* i*. A state* s_i *is* reachable *if it belongs to some run* r*.*

We say that a run is *cyclic* if, after some time, it always passes through the same sequence of states. It is acyclic otherwise.

2.5 Parametric Timed Automata

Parametric Timed Automata are an extension of the class of Timed Automata [AD94] to the parametric case and allow within guards and invariants the use of parameters in place of constants [AHV93].

Definition 8 (Parametric Timed Automaton). *A* parametric timed automaton (PTA) $\mathcal{A}$ *is a* 8*-tuple of the form* $\mathcal{A} = (\Sigma, Q, q_0, X, P, K, I, \rightarrow)$*, where*

- Σ *is a finite set of actions,*
- Q *is a finite set of locations,*
- $q_0 \in Q$ *is the initial location,*
- X *is a set of clocks,*
- P *is a set of parameters,*
- K *is a constraint on* P*,*
- I *is the invariant, assigning to every* $q \in Q$ *a constraint* $I(q)$ *on* X *and* P*, and*
- $\rightarrow$ *is a step relation consisting of elements of the form* (q, g, a, ρ, q')*, also denoted by* $q \overset{g,a,\rho}{\rightarrow} q'$*, where* $q, q' \in Q$*,* $a \in \Sigma$*,* $\rho \subseteq X$ *is a set of clocks to be reset by the step, and* g *(the step guard) is a constraint on* X *and* P*.*

[1] Without loss of generality, we focus on infinite behaviors in this paper.

The constraint K on the parameters corresponds to the *initial* constraint on the parameters, i.e., a constraint that will be true in all the states of the PTA (see semantics in Definition 13). For example, in a PTA with two parameters *min* and *max*, one may want to constrain *min* to be always smaller or equal to *max*, in which case K is defined to be $min \leq max$.

In the following, given a PTA $\mathcal{A} = (\Sigma, Q, q_0, X, P, K, I, \to)$, we will often denote this PTA by $\mathcal{A}(K)$ when clear from the context, in order to emphasize that only K will change in $\mathcal{A}$.

Given a PTA $\mathcal{A} = (\Sigma, Q, q_0, X, P, K, I, \to)$, for every parameter valuation $\pi = (\pi_1, \ldots, \pi_M)$, $\mathcal{A}[\pi]$ denotes the PTA $\mathcal{A}(K)$, where $K = \bigwedge_{i=1}^{M} p_i = \pi_i$. This corresponds to the PTA obtained from $\mathcal{A}$ by substituting every occurrence of a parameter p_i by constant π_i in the guards and invariants. Note that, as all parameters are instantiated, $\mathcal{A}[\pi]$ is a standard timed automaton. We say that $\mathcal{A}$ is *instantiated* with π.

We now define the semantics of PTAs. We first introduce the notion of state.

Definition 9 (State). *Let $\mathcal{A} = (\Sigma, Q, q_0, X, P, K, I, \to)$ be a PTA. A* state *s of $\mathcal{A}(K)$ is a pair (q, C) where $q \in Q$ is a location, and $C \in \mathcal{K}_{X \cup P}$ a constraint on X and P.*

For each valuation π of P, we may view a state s as the set of pairs (q, w) where w is a clock valuation such that $<w, \pi> \models C$.

We recall below the notion of π-compatibility [ACEF09].

Definition 10 (π-compatibility). *Let $\mathcal{A}$ be a PTA, and $s = (q, C)$ be a state of $\mathcal{A}$. The state s is said to be* compatible with π *or, more simply,* π-compatible *if $\pi \models C$.*

We now define the inclusion of a state in another one. This notion refers to the equality of definitions and inclusion of constraints, as formalized in the following definition.

Definition 11 (State inclusion). *We say that a state $s_1 = (q_1, C_1)$ is* included *in a state $s_2 = (q_2, C_2)$, denoted by $s_1 \subseteq s_2$, if $q_1 = q_2$ and $C_1 \subseteq C_2$.*

We say that two states $s_1 = (q_1, C_1)$ and $s_2 = (q_2, C_2)$ are equal*, denoted by $s_1 = s_2$, if $q_1 = q_2$ and $C_1 = C_2$.*

We now define the inclusion of a set of states in another one. Observe that this notion does not refer to the inclusion of each state of the first set into a state of the second set, but to the *equality* of each state of the first set with a state of the second set.

Definition 12 (Set inclusion). *We say that a set of states S_1 is* included *into a set of states S_2, denoted by $S_1 \sqsubseteq S_2$, if*

$$\forall s : s \in S_1 \Rightarrow s \in S_2.$$

We say that two sets of states S_1 and S_2 are equal*, denoted by $S_1 = S_2$, if $S_1 \sqsubseteq S_2$ and $S_2 \sqsubseteq S_1$.*

The *initial state* of $\mathcal{A}(K)$ is a state s_0 of the form (q_0, C_0), where $C_0 = K \wedge I(q_0) \wedge \bigwedge_{i=1}^{H-1} x_i = x_{i+1}$. In this expression, K is the initial constraint on the parameters, $I(q_0)$ is the invariant of the initial state, and the rest of the expression lets clocks evolve from the same initial value.

The semantics of PTAs is given in the following under the form of a LTS.

Definition 13 (Semantics of PTAs). *Let $\mathcal{A} = (\Sigma, Q, q_0, X, P, K, I, \rightarrow)$ be a PTA. The* semantics of *$\mathcal{A}$ is the LTS $(S, S_0, \Rightarrow)$ over Σ where*

$$\begin{aligned} S &= \{(q, C) \in Q \times \mathcal{K}_{X \cup P} \mid C \subseteq I(q)\}, \\ S_0 &= \{(q_0, K \wedge I(q_0) \wedge \textstyle\bigwedge_{i=1}^{H-1} x_i = x_{i+1})\} \end{aligned}$$

and the transition predicate $\Rightarrow$ is specified by the following three rules.

- *$(q, C) \overset{a}{\rightarrow} (q', C')$ if $(q, g, a, \rho, q') \in \rightarrow$, and C' is a constraint on X and P defined, using the set of (renamed) clock variables X', by:*
  $C'(X') = (\exists X : \ (C(X) \wedge g(X) \wedge X' = \rho(X) \wedge I(q')(X')))$.
- *$(q, C) \overset{d}{\rightarrow} (q, C')$, where d is a new parameter with values in $\mathbb{R}_{\geq 0}$, which means that C' is given by:*
  $C'(X') = (\exists X : \ (C(X) \wedge X' = X + d \wedge I(q)(X')))$.
- *$(q, C) \overset{a}{\Rightarrow} (q', C')$ if $\exists C''$ such that $(q, C) \overset{a}{\rightarrow} (q', C'')$ and $(q', C'') \overset{d}{\rightarrow} (q', C')$, i.e., C' is a constraint on the clocks and the parameters obtained by removing X and d from the following expression[2]:*
  $C'(X') = (\exists X, d : \ (C(X) \wedge g(X) \wedge X' = \rho(X) \wedge I(q')(X') \wedge I(q')(X' + d)))$.
  It can be shown that C' can be put under the form of a (convex) constraint on the clocks and the parameters (using, e.g., Fourier-Motzkin elimination [Sch86]) of X and d.

A step of the semantics of a PTA $\mathcal{A}(K)$ will be referred to as a *step* of $\mathcal{A}(K)$. Similarly, a run of the semantics of a PTA $\mathcal{A}(K)$ will be referred to as a *run* of $\mathcal{A}(K)$.

A run of $\mathcal{A}(K)$ is a finite alternating sequence of states and actions of the form $s_0 \overset{a_0}{\Rightarrow} s_1 \overset{a_1}{\Rightarrow} \cdots \overset{a_{m-1}}{\Rightarrow} s_m$, such that for all $i = 0, \ldots, m-1$, $a_i \in \Sigma$ and $s_i \overset{a_i}{\Rightarrow} s_{i+1}$ is a step of $\mathcal{A}(K)$.

One defines $Post^i_{\mathcal{A}(K)}(S)$ as the set of states reachable from a set S of states in exactly i steps, and $Post^*_{\mathcal{A}(K)}(S)$ as the set of all states reachable from S in $\mathcal{A}(K)$ (i.e., $Post^*_{\mathcal{A}(K)}(S) = \bigcup_{i \geq 0} Post^i_{\mathcal{A}(K)}(S)$).

We will be in particular interested here in computing the set $Post^*_{\mathcal{A}(K)}(\{s_0\})$, where s_0 is the initial state of $\mathcal{A}(K)$. Note that if $Post^{i+1}_{\mathcal{A}(K)}(\{s_0\}) = \emptyset$ (or, more generally, if $Post^{i+1}_{\mathcal{A}(K)}(\{s_0\}) \subseteq \bigcup_{j=0}^{i} Post^j_{\mathcal{A}(K)}(\{s_0\})$), then $Post^*_{\mathcal{A}(K)}(\{s_0\}) = \bigcup_{j=0}^{i} Post^j_{\mathcal{A}(K)}(\{s_0\})$.

[2] In $C'(X')$, we use the expression $I(q')(X') \wedge I(q')(X' + d)$ instead of $\forall 0 \leq e \leq d : I(q')(X' + e)$, using the fact that $I(q')$ is convex.

3 The Inverse Method

3.1 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]. In particular, LTL formulas satisfied in $\mathcal{A}[\pi_0]$ are also satisfied in $\mathcal{A}[\pi]$ (and vice-versa).

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

```
input  : PTA $\mathcal{A}$ of initial state $s_0 = (q_0, C_0)$
input  : 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          $K \leftarrow K \wedge \neg J$ ;
7          $S \leftarrow \bigcup_{j=0}^{i} Post^{j}_{\mathcal{A}(K)}(\{s_0\})$ ;
8      if $Post_{\mathcal{A}(K)}(S) \sqsubseteq S$ then
9          return $\bigcap_{(q,C) \in S} (\exists X : C)$
10     $i \leftarrow i + 1$ ;
11     $S \leftarrow S \cup Post_{\mathcal{A}(K)}(S)$ ;          // $S = \bigcup_{j=0}^{i} Post^{j}_{\mathcal{A}(K)}(\{s_0\})$
```

We recall *IM* in Algorithm 1. Starting with $K =$ `true`, *IM* iteratively computes a growing set of reachable states. 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 the projection of C onto the parameters and $\neg J$ is added to K. The procedure is then started again with this new K, and so on, until all new states are *equal* to a state computed previously. We finally return the intersection of the projection onto the parameters of all the constraints associated with the reachable states. Actually, the two major steps of the algorithm are the following ones:

1. the iterative removal of the π_0-incompatible states (by negating a π_0-incompatible inequality J) prevents for any $\pi \models K_0$ the behavior different from π_0;
2. the final intersection of the constraints associated with all the reachable states guarantees that all the behaviors under π_0 are allowed for all $\pi \models K_0$.

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 $\mathcal{A}[\pi]$).

Furthermore, the fixpoint condition can also be lifted: instead of stopping the algorithm when all new states are *equal* to a state computed previously, stopping when all new states are *included* into a previous state outputs a weaker constraint, and still preserves safety.

3.2 Properties

Preservation of Trace Sets. The main property of *IM* is the preservation of trace sets [ACEF09]. Given $\mathcal{A}$ and π_0, the trace sets of $\mathcal{A}[\pi_0]$ and $\mathcal{A}[\pi]$ are the same, for all $\pi \models IM(\mathcal{A}, \pi_0)$. 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], which is a linear temporal logic constituted by both atomic state propositions and events.

Non-confluence. It has been shown that *IM* is non-confluent, i.e., several applications of the algorithm can lead to different results [And10c]. Non-confluence comes from the non-deterministic selection of a π_0-incompatible inequality J (line 5 in Algorithm 1). The algorithm behaves deterministically when such a situation of choice is non encountered. Finding (non-trivial) syntactic conditions such that this situation does not happen is the subject of ongoing work.

Non-maximality. 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 $\pi \not\models IM(\mathcal{A}, \pi_0)$ such that $\mathcal{A}[\pi]$ and $\mathcal{A}[\pi_0]$ have the same trace sets. However, it can be shown that, when the algorithm is deterministic, i.e., when only one π_0-incompatible inequality J can be selected at a time, the output constraint is maximal.

Termination. Reachability analysis is known to be undecidable in the framework of PTAs [AHV93,Doy07], and computations performed with tools on PTAs (such as HyTech [HHWT95]) do not always terminate. However, sufficient conditions have been shown for ensuring termination of *IM*, in particular for acyclic systems, which is the case for most of hardware verification case studies. Furthermore, even for cyclic systems, the application of *IM* always terminates in practice for all the case studies we have considered [And10a].

3.3 Discussion

A drawback of the inverse method 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. As a consequence, we introduce in the following optimized algorithms relaxing the property of equality of trace sets, but all preserving interesting properties related to (non-)reachability.

4 Optimized Algorithms Based on the Inverse Method

We introduce here several algorithms derived from *IM*: none of them guarantee the strict equality of trace sets, but all output 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.

We introduce[3] three algorithms derived from *IM*, namely $IM_{\subseteq}$ (Section 4.1), $IM^{\cup}$ (Section 4.2) and IM^{K} (Section 4.3). We then introduce combinations between these algorithms, i.e., $IM^{\cup}_{\subseteq}$ (Section 4.4) and $IM^{K}_{\subseteq}$ (Section 4.5). For each algorithm, we show that the constraint is weaker than *IM* (when applicable), study the termination, and study the properties guaranteed by the output constraint. We finally summarize the properties of the various algorithms and show their respective interest on an example of PTA (Section 4.6).

4.1 Algorithm with State Inclusion in the Fixpoint

Description. The algorithm $IM_{\subseteq}$ is obtained from *IM* by stopping 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. We give the full description of $IM_{\subseteq}$ in Algorithm 2.

Weaker Constraint. We show that the constraint output by $IM_{\subseteq}$ is weaker than the one output by *IM*.

Proposition 1. *Let $\mathcal{A}$ be a PTA, and π_0 a parameter valuation. Then, we have:*

$$IM(\mathcal{A}, \pi_0) \subseteq IM_{\subseteq}(\mathcal{A}, \pi_0).$$

Proof. See Proposition 3.30 in [And10c]. □

[3] Algorithms $IM_{\subseteq}$ has been introduced in [And10c]. As a consequence, we simply refer to [And10c] for the respective proofs. Algorithm $IM^{\cup}$ has also been introduced in [And10c], but we give here a more precise characterization (see Proposition 8). Algorithm $IM^{\cup}_{\subseteq}$ has been briefly mentioned in [And10c] without being properly characterized. Algorithms IM^{K} and $IM^{K}_{\subseteq}$ are brand new contributions.

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

```
input  : PTA $\mathcal{A}$ of initial state $s_0 = (q_0, C_0)$
input  : 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          $K \leftarrow K \wedge \neg J$ ;
7          $S \leftarrow \bigcup_{j=0}^{i} Post^{j}_{\mathcal{A}(K)}(\{s_0\})$ ;
8      if $\forall s \in Post_{\mathcal{A}(K)}(S), \exists s' \in S : s \subseteq s'$ then
9          return $\bigcap_{(q,C) \in S} (\exists X : C)$
10     $i \leftarrow i + 1$ ;
11     $S \leftarrow S \cup Post_{\mathcal{A}(K)}(S)$ ;                // $S = \bigcup_{j=0}^{i} Post^{j}_{\mathcal{A}(K)}(\{s_0\})$
```

Earlier Termination. This algorithm $IM_{\subseteq}$ 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. This is formalized in the following proposition.

Proposition 2. *Let $\mathcal{A}$ be a PTA, and π_0 a parameter valuation. If $IM(\mathcal{A}, \pi_0)$ terminates, then $IM_{\subseteq}(\mathcal{A}, \pi_0)$ also terminates.*

Proof. Based on the fact that the two algorithms are equal, except the fixpoint condition, which is weaker for $IM_{\subseteq}$. □

Note that the reciprocal statement does not hold: there are actually examples of PTAs for which the application of $IM_{\subseteq}$ terminates although IM does not terminate (see Example 3.29 in [And10c]).

Equality of Traces Prefixes. Beside, $IM_{\subseteq}$ 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). In other words, the sets of traces of length i, with $i \leq n$, are equivalent in $\mathcal{A}[\pi]$ and $\mathcal{A}[\pi]$, for all $\pi \models IM_{\subseteq}(\mathcal{A}, \pi_0)$. This is formalized the following proposition.

Proposition 3. *Let $\mathcal{A}$ be a PTA, and π_0 a parameter valuation. 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, and*

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.*

Proof. See Proposition 3.26 in [And10c]. □

Equality of Reachable Locations. Furthermore, we show the interesting property that the set of reachable locations in $\mathcal{A}[\pi]$, for all $\pi \models IM_{\subseteq}(\mathcal{A}, \pi_0)$, is the same as in $\mathcal{A}[\pi_0]$.

Proposition 4. *Let $\mathcal{A}$ be a PTA, and π_0 a parameter valuation. 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.*

Proof. See proof of Proposition 3.27 in [And10c]. □

Preservation of Non-reachability. When considering the absence of bad behavior, i.e., the non-reachability of a given location, Proposition 3 only shows that, if a location is not reachable in $\mathcal{A}[\pi_0]$, it is also not reachable in $\mathcal{A}[\pi]$, for all $\pi \models IM_{\subseteq}(\mathcal{A}, \pi_0)$, *within traces up to length n*. Although this may be of interest for bounded verification, it is usually more interesting to guarantee that a given location is *never* reachable, i.e., does not belong to any trace of unbounded length. We now show that, if a given location is not reachable in $\mathcal{A}[\pi_0]$, it will also not be reachable in $\mathcal{A}[\pi]$, for all $\pi \models IM_{\subseteq}(\mathcal{A}, \pi_0)$.

Proposition 5. *Let $\mathcal{A}$ be a PTA, π_0 a parameter valuation, and q a location of $\mathcal{A}$. Suppose that $IM_{\subseteq}(\mathcal{A}, \pi_0)$ terminates with output K_0. If q does not belong to the trace set of $\mathcal{A}[\pi_0]$, then q does not belong to the trace set of $\mathcal{A}[\pi]$, for all $\pi \models K_0$.*

Proof. By corollary of Proposition 4. □

4.2 Algorithm with Union of the Constraints

Description. 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 each of the *last* state of a run. This notion of last state is easy to understand for finite runs. When considering infinite (and necessarily[4] cyclic) runs, the last state refers to the second occurrence of a same state within a run, i.e., to the first time that a state is equal to a previous state of the same run. We give the full description of $IM^{\cup}$ in Algorithm 3.

This algorithm is identical to the original inverse method IM (see Algorithm 1) except in two points:

[4] Recall that, if the runs are infinite but not cyclic, the algorithm does not terminate.

Algorithm 3: Algorithm $IM^{\cup}(\mathcal{A}, \pi_0)$

```
input : PTA $\mathcal{A}$ of initial state $s_0 = (q_0, C_0)$
input : Parameter valuation $\pi_0$
output: Constraint $K_0$ on the parameters
1  $i \leftarrow 0$ ;  $K \leftarrow$ true ;  $S \leftarrow \{s_0\}$ ;  $S_{last} \leftarrow \{\}$
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          $K \leftarrow K \wedge \neg J$ ;
7          $S \leftarrow \bigcup_{j=0}^{i} Post^{j}_{\mathcal{A}(K)}(\{s_0\})$ ;
8      foreach $s \in Post_{\mathcal{A}(K)}(S)$ do
9          if $Post_{\mathcal{A}(K)}(\{s\}) = \emptyset$ or $s \in S$ then
10             $S_{last} \leftarrow S_{last} \cup \{s\}$
11     if $Post_{\mathcal{A}(K)}(S) \sqsubseteq S$ then
12         return $\bigcup_{(q,C) \in S_{last}} (\exists X : C)$
13     $i \leftarrow i + 1$ ;
14     $S \leftarrow S \cup Post_{\mathcal{A}(K)}(S)$ ;                // $S = \bigcup_{j=0}^{i} Post^{j}_{\mathcal{A}(K)}(\{s_0\})$
```

1. Lines 8 to 10 compute the last states of each run: either states being the last of a finite run (i.e., when $Post_{\mathcal{A}(K)}(\{s\})$ is empty), or states being equal to states computed previously (i.e., when $s \in S$).
2. Line 12 returns the union of the constraints on the parameters associated with the last states of the runs, instead of the intersection of the constraints associated with all the states computed.

Note that the lines mentioned at item 1 are actually only modified here in order to compute the different return of item 2, and do not interfere with the rest of the algorithm.

Weaker Constraint. We show that the constraint output by $IM^{\cup}$ is weaker than the one output by IM.

Proposition 6. *Let $\mathcal{A}$ be a PTA, and π_0 a parameter valuation. Then:*

$$IM(\mathcal{A}, \pi_0) \subseteq IM^{\cup}(\mathcal{A}, \pi_0).$$

Proof. See Proposition 3.40 in [And10c].

Note that the constraints output by $IM_{\subseteq}$ and $IM^{\cup}$ are incomparable (see example in Section 4.6 for which two incomparable constraints are synthesized).

Same Termination. We show that the termination is the same as for *IM*.

Proposition 7. *Let $\mathcal{A}$ be a PTA, and π_0 a parameter valuation. Then, $IM^{\cup}(\mathcal{A}, \pi_0)$ terminates if and only if $IM(\mathcal{A}, \pi_0)$ terminates.*

Proof. See Proposition 3.39 in [And10c]. $\square$

At Least one Trace. Although it is clear that the equality of trace sets is no longer guaranteed for $\pi \models IM^{\cup}(\mathcal{A}, \pi_0)$, some properties are still preserved by this variant. By performing the union of the last states of each trace, 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 $\mathcal{A}[\pi_0]$. In other words, for all $\pi \models K_0$, for each trace of the trace set of $\mathcal{A}[\pi]$, there exists an identical trace in the trace set of $\mathcal{A}[\pi_0]$. Furthermore, each trace of $\mathcal{A}[\pi_0]$ is reachable for *at least* one valuation $\pi \models K_0$.

This is formalized in the following proposition.

Proposition 8. *Let $\mathcal{A}$ be a PTA and π_0 a parameter valuation. Let $K_0 = IM^{\cup}(\mathcal{A}, \pi_0)$. Then:*

1. *For all $\pi \models K_0$, every trace of $\mathcal{A}[\pi]$ is equal to a trace of $\mathcal{A}[\pi_0]$.*
2. *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.*

Proof. Item 1 comes from Theorem 3.37 in [And10c]. We show item 2 in the following. Let $K_0 = IM^{\cup}(\mathcal{A}, \pi_0)$. From the algorithm, K_0 is of the form $\bigcup_{s_i \in S_{last}} K_i$, i.e., a disjunction of constraints K_i, each of them corresponding to a run of $\mathcal{A}[\pi_0]$. Let T_i be a trace of $\mathcal{A}[\pi_0]$. Let K be the constraint just before the end of $IM^{\cup}$ (i.e., K corresponds to the iterative conjunction of the negations of the π_0-incompatible inequalities). Since $IM^{\cup}$ is identical to *IM* with the exception of the returned constraint (and the computation of S_{last}), we have the following property as in *IM*: for each run of $\mathcal{A}[\pi_0]$, there exists an equivalent run in $\mathcal{A}(K)$ (from Lemma 3.3 in [And10c]). Hence, there exists a run R_i equivalent to T_i, i.e., sharing the same sequence of locations and actions. Let C_i be the constraint associated to the last state of R_i. By definition, Algorithm $IM^{\cup}$ returns a disjunction of constraints, containing in particular the projection of C_i onto the parameters, say K_i. Note that, since C_i is necessarily satisfiable, so is K_i. By the semantics of PTAs, for all $\pi \models K_i$, a run equivalent to R_i can be reached. Since R_i is equivalent to T_i, for all $\pi \models K_i$, the trace set of $\mathcal{A}[\pi]$ contains T_i. The fact that K_i is satisfiable, and hence contains at least one parameter valuation, gives the result. $\square$

Preservation of Non-reachability. We now show that non-reachability of locations is preserved.

Proposition 9. *Let $\mathcal{A}$ be a PTA, π_0 a parameter valuation, and q a location of $\mathcal{A}$. Suppose that $IM^{\cup}(\mathcal{A}, \pi_0)$ terminates with output K_0. If q does not belong to the trace set of $\mathcal{A}[\pi_0]$, then q does not belong to the trace set of $\mathcal{A}[\pi]$, for all $\pi \models K_0$.*

Proof. By contraposition on item 1 of Proposition 8. □

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

4.3 Algorithm with Simple Return

Description. 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. We give the full description of IM^K in Algorithm 4.

Algorithm 4: Algorithm $IM^K(\mathcal{A}, \pi_0)$

```
input : PTA $\mathcal{A}$ of initial state $s_0 = (q_0, C_0)$
input : 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          $K \leftarrow K \wedge \neg J$ ;
7          $S \leftarrow \bigcup_{j=0}^{i} Post^j_{\mathcal{A}(K)}(\{s_0\})$ ;
8      if $\forall s \in Post_{\mathcal{A}(K)}(S), \exists s' \in S : s \subseteq s'$ then
9          return $K$
10     $i \leftarrow i + 1$ ;
11     $S \leftarrow S \cup Post_{\mathcal{A}(K)}(S)$ ;        // $S = \bigcup_{j=0}^{i} Post^j_{\mathcal{A}(K)}(\{s_0\})$
```

Weaker Constraint. We state in the following proposition that the constraint output by IM^K is weaker than the one output by IM.

Proposition 10. *Let $\mathcal{A}$ be a PTA, and π_0 a parameter valuation. Then, we have:*

$$IM(\mathcal{A}, \pi_0) \subseteq IM^K(\mathcal{A}, \pi_0).$$

Proof. First recall that the two algorithms have the same fixpoint condition. Moreover, in IM we have that $K_0 \subseteq K$ (from Lemma 3.1 in [And10c]), where K_0 is the constraint returned by IM. Then, the proof comes from the fact that IM returns K_0 whereas IM^K returns K. □

Also note that the constraints output by $IM_\subseteq$ and IM^K are incomparable (see example in Section 4.6 for which two incomparable constraints are synthesized).

Same Termination. Termination is the same for IM^K and IM.

Proposition 11. *Let $\mathcal{A}$ be a PTA, and π_0 a parameter valuation. Then, $IM^K(\mathcal{A}, \pi_0)$ terminates if and only if $IM(\mathcal{A}, \pi_0)$ terminates.*

Proof. Based on the fact that the fixpoint conditions of the two algorithms are the same, and the variables involved in the fixpoint (viz., S and K) are computed in the same way. $\square$

Subset of the Trace Set. 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.

Proposition 12. *Let $\mathcal{A}$ be a PTA and π_0 a parameter valuation. 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 $\mathcal{A}[\pi_0]$.*

Proof. See proof of Theorem 3.37 in [And10c][5]. $\square$

Preservation of Non-reachability. This algorithm only preserves the non-reachability of locations.

Proposition 13. *Let $\mathcal{A}$ be a PTA, π_0 a parameter valuation, and q a location of $\mathcal{A}$. Suppose that $IM^K(\mathcal{A}, \pi_0)$ terminates with output K_0. If q does not belong to the trace set of $\mathcal{A}[\pi_0]$, then q does not belong to the trace set of $\mathcal{A}[\pi]$, for all $\pi \models K_0$.*

Proof. By contraposition on Proposition 12. $\square$

4.4 Combination: Inclusion in Fixpoint and Union

Description. One can combine the variant of the fixpoint (viz., $IM_\subseteq$) with the first variant of the constraint output (viz., $IM^\cup$), thus leading to a new algorithm $IM^\cup_\subseteq$. We give the full description of $IM^\cup_\subseteq$ in Algorithm 5.

Weaker Constraint. We show that the constraint output by $IM^\cup$ is weaker than the ones output by both $IM_\subseteq$ and $IM^\cup$.

Proposition 14. *Let $\mathcal{A}$ be a PTA, and π_0 a parameter valuation. Then:*

1. $IM_\subseteq(\mathcal{A}, \pi_0) \subseteq IM^\cup_\subseteq(\mathcal{A}, \pi_0)$.
2. $IM^\cup(\mathcal{A}, \pi_0) \subseteq IM^\cup_\subseteq(\mathcal{A}, \pi_0)$.

Proof. The constraint output by $IM^\cup_\subseteq$ is weaker than the one output by $IM_\subseteq$ for the same reason that the constraint output by $IM_\subseteq$ is weaker than IM (see Proposition 1). And similarly for $IM^\cup$ (see Proposition 6).

Note that the constraints output by $IM^\cup_\subseteq$ and IM^K are incomparable (see example in Section 4.6 for which two incomparable constraints are synthesized).

[5] This result is actually proven in [And10c] for $IM^\cup$, but can be proved in exactly the same way for IM^K. We prove here a stronger result for $IM^\cup$ (see Proposition 8).

Algorithm 5: Algorithm $IM^{\cup}_{\subseteq}(\mathcal{A}, \pi_0)$

```
input : PTA $\mathcal{A}$ of initial state $s_0 = (q_0, C_0)$
input : Parameter valuation $\pi_0$
output: Constraint $K_0$ on the parameters
1  $i \leftarrow 0$ ;  $K \leftarrow$ true ;  $S \leftarrow \{s_0\}$ ;  $S_{last} \leftarrow \{\}$
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          $K \leftarrow K \wedge \neg J$ ;
7          $S \leftarrow \bigcup_{j=0}^{i} Post^{j}_{\mathcal{A}(K)}(\{s_0\})$ ;
8      foreach $s \in Post_{\mathcal{A}(K)}(S)$ do
9          if $Post_{\mathcal{A}(K)}(\{s\}) = \emptyset$ or $s \in S$ then
10             $S_{last} \leftarrow S_{last} \cup \{s\}$
11     if $\forall s \in Post_{\mathcal{A}(K)}(S), \exists s' \in S : s \subseteq s'$ then
12         return $\bigcup_{(q,C) \in S_{last}} (\exists X : C)$
13     $i \leftarrow i + 1$ ;
14     $S \leftarrow S \cup Post_{\mathcal{A}(K)}(S)$ ;          // $S = \bigcup_{j=0}^{i} Post^{j}_{\mathcal{A}(K)}(\{s_0\})$
```

Same Termination as $IM_{\subseteq}$.

Proposition 15. *Let $\mathcal{A}$ be a PTA, and π_0 a parameter valuation. Then, $IM^{\cup}_{\subseteq}(\mathcal{A}, \pi_0)$ terminates if and only if $IM_{\subseteq}(\mathcal{A}, \pi_0)$ terminates.*

Proof. Based on the fact that the fixpoint condition is the same for $IM_{\subseteq}$ and $IM^{\cup}_{\subseteq}$ (only the returned constraint is modified). □

Preservation of Non-reachability. This algorithm combines the properties of $IM_{\subseteq}$ and $IM^{\cup}$.

Proposition 16. *Let $\mathcal{A}$ be a PTA and π_0 a parameter valuation. Suppose that $IM^{\cup}_{\subseteq}(\mathcal{A}, \pi_0)$ terminates with output K_0 after n iterations of the outer* ***do*** *loop. Then:*

1. *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, and*
2. *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.*

Proof. From Propositions 5 and 9. □

Although not of high interest in practice, this result leads to the property of non-reachability.

Proposition 17. *Let $\mathcal{A}$ be a PTA, π_0 a parameter valuation, and q a location of $\mathcal{A}$. Suppose that $IM^{\cup}_{\subseteq}(\mathcal{A}, \pi_0)$ terminates with output K_0. If q does not belong to the trace set of $\mathcal{A}[\pi_0]$, then q does not belong to the trace set of $\mathcal{A}[\pi]$, for all $\pi \models K_0$.*

Proof. From Proposition 17, we have that q is not reachable in any trace of length smaller or equal to n. The result is obtained following a reasoning similar to the proof of Proposition 5. □

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

4.5 Combination: Inclusion in Fixpoint and Direct Return

Description. 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 a new algorithm $IM^K_{\subseteq}$. We give the full description of $IM^{\cup}_{\subseteq}$ in Algorithm 6.

Algorithm 6: Algorithm $IM^K_{\subseteq}(\mathcal{A}, \pi_0)$

```
input : PTA $\mathcal{A}$ of initial state $s_0 = (q_0, C_0)$
input : 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          $K \leftarrow K \wedge \neg J$ ;
7          $S \leftarrow \bigcup_{j=0}^{i} Post^{j}_{\mathcal{A}(K)}(\{s_0\})$ ;
8      if $\forall s \in Post_{\mathcal{A}(K)}(S), \exists s' \in S : s \subseteq s'$ then
9          return $K$
10     $i \leftarrow i + 1$ ;
11     $S \leftarrow S \cup Post_{\mathcal{A}(K)}(S)$ ;          // $S = \bigcup_{j=0}^{i} Post^{j}_{\mathcal{A}(K)}(\{s_0\})$
```

Weaker Constraint. We show that the constraint output by $IM^K_{\subseteq}$ is weaker than the ones output by both IM^K and $IM^{\cup}_{\subseteq}$.

Proposition 18. *Let $\mathcal{A}$ be a PTA, and π_0 a parameter valuation. Then:*

1. $IM^K(\mathcal{A}, \pi_0) \subseteq IM^K_{\subseteq}(\mathcal{A}, \pi_0)$.
2. $IM^{\cup}_{\subseteq}(\mathcal{A}, \pi_0) \subseteq IM^K_{\subseteq}(\mathcal{A}, \pi_0)$.

Proof. The constraint output by $IM_{\subseteq}^{K}$ is weaker than the one output by IM^{K} for the same reason that the constraint output by $IM_{\subseteq}$ is weaker than IM (see Proposition 1). The constraint output by $IM_{\subseteq}^{K}$ is weaker than the one output by $IM_{\subseteq}^{\cup}$ for the same reason that the constraint output by IM^{K} is weaker than $IM^{\cup}$ (see Proposition 10).

Same Termination as $IM_{\subseteq}$.

Proposition 19. *Let $\mathcal{A}$ be a PTA, and π_0 a parameter valuation. Then, $IM_{\subseteq}^{K}(\mathcal{A}, \pi_0)$ terminates if and only if $IM_{\subseteq}(\mathcal{A}, \pi_0)$ terminates.*

Proof. Based on the fact that the fixpoint condition is the same for $IM_{\subseteq}$ and $IM_{\subseteq}^{K}$ (only the returned constraint is modified). □

Preservation of Non-reachability. This algorithm only preserves the non-reachability of locations.

Proposition 20. *Let $\mathcal{A}$ be a PTA, π_0 a parameter valuation, and q a location of $\mathcal{A}$. Suppose that $IM_{\subseteq}^{K}(\mathcal{A}, \pi_0)$ terminates with output K_0. If q does not belong to the trace set of $\mathcal{A}[\pi_0]$, then q does not belong to the trace set of $\mathcal{A}[\pi]$, for all $\pi \models K_0$.*

Proof. From Propositions 5 and 17. □

4.6 Summary of the Algorithms

We summarize in Table 1 the properties of each algorithm.

Property	IM	$IM_{\subseteq}$	$IM^{\cup}$	IM^{K}	$IM_{\subseteq}^{\cup}$	$IM_{\subseteq}^{K}$
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 $IM(\mathcal{A}, \pi_0) \subseteq$

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

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

$IM_{\subseteq}(\mathcal{A}, \pi_0)$. 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 the properties of Table 1. 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.

Non-maximality. Actually, none of these algorithms are maximal, i.e., none of them synthesize the maximal constraint corresponding to the property they are characterized with. This is due to the non-confluence of the algorithms, itself due to the random selection of a π_0-incompatible inequality. However, it can be shown that the constraint is maximal when no such random selection of a π_0-incompatible inequality occurs, i.e., when the algorithm runs in a fully deterministic way. We address the issue of synthesizing a maximal constraint in Section 5. Also note that the comparison between the constraints (see Figure 1 (right)) holds only for deterministic analyses.

An Example of PTA. Let us consider the PTA $\mathcal{A}_{var}$ depicted in Figure 2 and containing in particular 2 parameters[6].

Consider the following reference parameter valuation π_0:

$$p_1 = 1 \quad \wedge \quad p_2 = 4$$

In $\mathcal{A}[\pi_0]$, location q_4 is not reachable, and can be considered as a "bad" location.

Applying the original version IM of our inverse method to $\mathcal{A}_{var}$ and π_0, the following constraint is synthesized:

$$p_2 \geq 3 \quad \wedge \quad 5 * p_1 > p_2 \quad \wedge \quad p_2 \geq 4 * p_1$$

[6] Note that we allow here the use of both parameters and constants in guards and variants, in order to have an intuitive example, and be able to represent the parameter space in 2 dimensions. Our results extend in a straightforward manner to this framework.

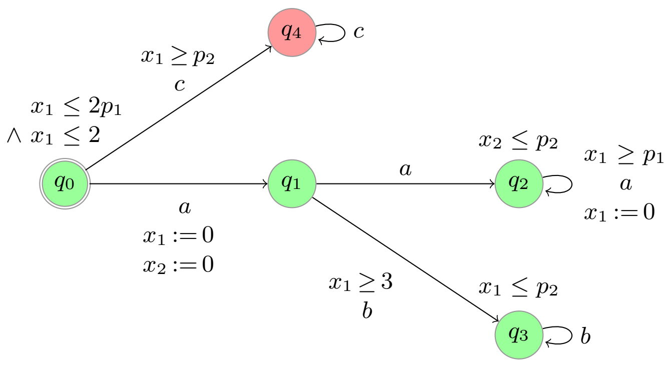

Fig. 2. A PTA $\mathcal{A}_{var}$ for comparing the variants of IM

Applying $IM^{\cup}$ to $\mathcal{A}_{var}$ and π_0, we get:

$$\begin{array}{l} \quad p_2 \geq 4 * p_1 \wedge 5 * p_1 > p_2 \\ \vee\ 2 * p_2 > p_1 \wedge 5 * p_1 > p_2 \quad \wedge \quad p_2 \geq 3 \end{array}$$

Note that, due to the non-deterministic selection of a π_0-inequality, an alternative result is:

$$\begin{array}{l} \quad p_2 \geq 4 * p_1 \wedge 5 * p_1 > p_2 \quad \wedge \quad p_2 > 2 \\ \vee \qquad p_2 \geq 3 \wedge 5 * p_1 > p_2 \end{array}$$

Applying IM^{K} to $\mathcal{A}_{var}$ and π_0, we get the following constraint:

$$2 * p_2 > p_1 \quad \wedge \quad 5 * p_1 > p_2$$

Note that an alternative result is:

$$p_2 > 2 \quad \wedge \quad 5 * p_1 > p_2$$

Applying $IM_{\subseteq}$ to $\mathcal{A}_{var}$ and π_0, we get:

$$p_2 \geq 3 \quad \wedge \quad p_2 \geq p_1$$

Applying $IM_{\subseteq}^{\cup}$ to $\mathcal{A}_{var}$ and π_0, we get the following constraint:

$$\begin{array}{l} \quad 2 * p_2 > p1 \wedge p_2 \geq p_1 \\ \vee\ 2 * p_2 > p1 \wedge p_2 \geq 3 \end{array}$$

Note that an alternative result is:

$$\begin{array}{l} \quad p_2 \geq p_1 \wedge p_2 > 2 \\ \vee\ p_2 \geq 3 \end{array}$$

Applying $IM_{\subseteq}^{K}$ to $\mathcal{A}_{var}$ and π_0, we get the following constraint:

$$2 * p_2 > p_1$$

Note that an alternative result is:

$$p_2 > 2$$

Comparison of the Constraints. 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. More generally, one can see intuitively 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 $2 * p_1 > p_2 \vee p_2 > 2$.

Let us compare the constraints synthesized by the variants for $\mathcal{A}_{var}$. We give in Figure 3 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 "good" zone not covered by the constraint is depicted in very light gray. The point represents π_0.

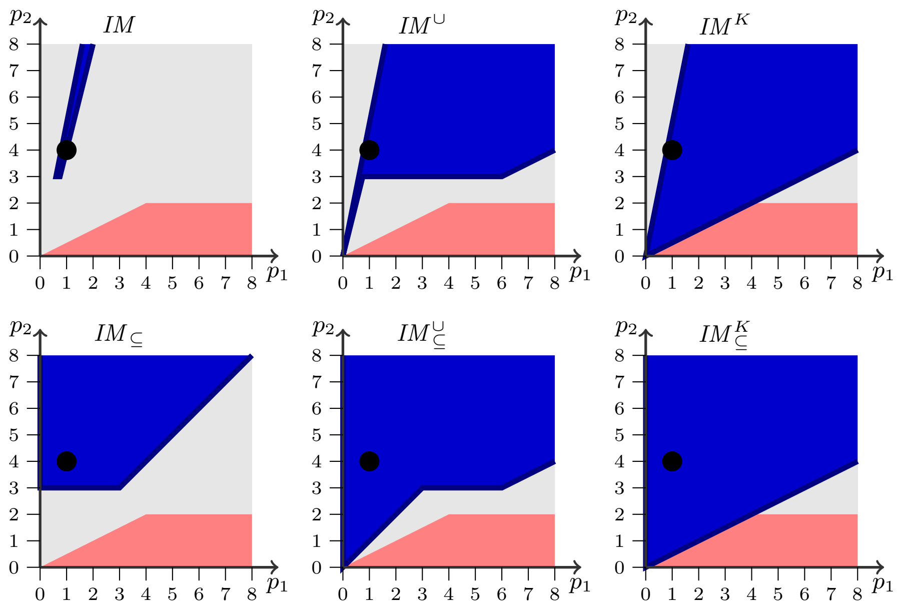

Fig. 3. Comparison of the constraints synthesized for $\mathcal{A}_{var}$

This example illustrates well the relationship between the different constraints. In particular, the constraint synthesized by $IM_{\subseteq}^{K}$ 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 $IM_{\subseteq}^{K}$ does not cover the whole "good" zone). This will be addressed in Section 5.

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 memory consumption and computation time is easy, it is not trivial to measure the gain in terms of the "size" of the constraints synthesized, i.e., in terms of the size of the set of parameter valuations covered. It requires measures of polyhedra, which is in particular not easy when they are non-convex. As a consequence, we postpone this study to the next section, where we perform a coverage of the parameter space with several constraints. When a given part of the parameter space is covered by several constraints, the less constraints, the larger the sets of parameter valuations are (see Table 2 in Section 6).

5 Behavioral Cartography

Although the inverse method has been shown of interest for a large panel of case studies (see, e.g., [And10a]), the main shortcoming of *IM* is that the constraint output by the algorithm is not maximal, i.e., there may exist other parameter valuations outside K_0 with the same trace set as under the reference valuation. Moreover, the good parameters problem relates to the synthesis of parameter valuations corresponding to *any* good behavior, not to a single one. We introduced in Section 4 various algorithms derived from *IM* synthesizing a weaker constraint, not necessarily preserving the equality of the trace sets.

An alternative method has been proposed in [AF10], viz., the behavioral cartography algorithm. By iterating the inverse method on (for instance) integer points in a bounded parameter domain, we are able to cover the parameter space with *tiles*, i.e., dense sets of parameters in which the trace sets are the same. Then, given a property φ on trace sets (basically linear time properties), one can partition the parameter space between good tiles (tiles for which all points satisfy φ) and bad tiles. Then, the set of parameters satisfying φ corresponds to the union of the good tiles. Note that this algorithm is independent from φ; only the partition between good and bad tiles involves φ.

We show in the following how the behavioral cartography can be favorably combined with the algorithms defined in Section 4. This has the major advantage to significantly reduce the number of tiles, hence both the state space and the computation time.

Behavioral Cartography Algorithm. The behavioral cartography algorithm relies on the idea of covering the parameter space within a rectangular real-valued parameter domain V_0 [AF10]. By iterating the inverse method *IM* recalled in Section 3 over all the *integer* valuations of the rectangle V_0 (of which there are a finite number), one is able to decompose the parameter space included into V_0 into a list *Tiling* of behavioral tiles. In each of these behavioral tiles, the time-abstract behavior of the system, i.e., the trace set, is the same for all points of the tiles. We recall this algorithm $BC(\mathcal{A}, V_0)$ in Algorithm 7.

Algorithm 7: Behavioral Cartography Algorithm $BC(\mathcal{A}, V_0)$

```
input  : A PTA $\mathcal{A}$, a finite rectangle $V_0 \subseteq \mathbb{R}_{\geq 0}^M$
output: $Tiling$: list of tiles (initially empty)
1 repeat
2     select an integer valuation $\pi \in V_0$;
3     if $\pi$ does not belong to any tile of Tiling then
4         Add $IM(\mathcal{A}, \pi)$ to Tiling;
5 until Tiling contains all the integer valuations of $V_0$;
```

The algorithm BC guarantees to cover at least the *integer* points within V_0. 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 [And10c] for details).

Synthesis for Safety Properties. Once the parameter space is (partially or totally) covered by *Tiling*, it is straightforward to partition the tiles between good or bad, depending on a property on traces that one wants to check. This can be done by checking the property for one point in each tile (using a non-parametric model checker, e.g., Uppaal [LPY97], applied to the PTA instantiated by the considered point). Alternatively, BC can be refined in order to check on-the-fly, for each tile, whether the corresponding trace set (computed by IM) reaches or not some given "bad" location. This is automatically performed by Imitator II (see Section 6), with no additional cost since the trace set is computed by IM.

Once the set of tiles is divided into good and bad, one can synthesize a set of parameters, by returning the union of the constraints associated to the good tiles w.r.t. the property one wants to check.

Combination with the Variants. By replacing within BC the call to IM by a call to one of the algorithms introduced in Section 4, one changes the properties of the tiles: for each tile, the corresponding trace set inherits the properties of the considered variant, and does not necessary preserve the equality of trace sets. However, as shown in Section 4, 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 4 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.

We denote by $BC_\subseteq$ the variant of BC calling $IM_\subseteq$ instead of IM (and similarly for the other variants). As for IM, depending on the property one wants to check, one can select the most appropriate variant of BC. In particular, for properties involving non-reachability of locations only, the most accurate version is $BC_\subseteq^K$ (see experiments in Table 2).

6 Implementation and Experiments

All these algorithms, as well as the original IM, have been implemented in IMITATOR II, a tool for synthesizing parameters in the framework of PTAs [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 applied a version of the behavioral cartography to a certain rectangle V_0. Then, using a good property one wants to check, we split the tiles between good and bad. Finally, we synthesize a constraint corresponding to the constraints associated to the good tiles. For each case study, V_0 is either entirely covered, or "almost entirely covered" (there are sometimes some small holes). Furthermore, the whole real-valued parameter domain is entirely covered for some case studies: this is the case, in particular, of AND–OR and SPSMALL for all algorithms, and $\mathcal{A}_{var}$ for all algorithms except IM.

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[7] of the initial parameter rectangle domain. We then give the number of tiles for each algorithm and the computation time in seconds for each algorithm. Note that the total number of parameters of the system may actually be greater than the number of parameters varying of the cartography, in which case the dimension of the constraint synthesized will be equal to the total number of parameters.

Example			Tiles						Time (s)					
Name	$\|P\|$	$\|V_0\|$	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
SPSMALL'	2	100	23	10	10	23	10	10	110	62	61	110	62	61

Table 2. Comparison of the algorithms for the behavioral cartography

[7] For all these case studies, we called the inverse method on integer points only.

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, and can be found in [And10c].

All experiments were performed on an Intel Core 2 Duo 2,33 Ghz with 3,2 Go memory, with IMITATOR II using the no-random, no-dot and no-log modes (see [And10b]). The latest version[8] of IMITATOR II as well as all the mentioned case studies can be found on IMITATOR II's Web page[9].

Sample Example $\mathcal{A}_{var}$. Recall from Section 4.6 that the property of good behavior is directly encoded as the non-reachability of a "bad" location (q_4). We consider the following V_0: $p_1 \in [1; 8]$ and $p_2 \in [0; 8]$.

Flip-flop Circuit. The property of good behavior corresponds to some ordering of the actions in the traces [CC07,And10c], which we verify using an observer (hence, this becomes a property of pure non-reachability). Considering the low number of tiles, and the reasonably small size of the trace sets, the separation between good and bad tiles has been performed manually. We consider the following V_0: $\delta_3^+ \in [8; 30]$ and $\delta_4^+ \in [3; 30]$.

AND–OR Circuit. This circuit has been studied in [CC05]. The 5 parameters varying in the cartography are as follows: $delta_{a^\uparrow}^+ \in [13; 30]$, $delta_{a^\downarrow}^+ \in [16; 30]$, $delta_{b^\downarrow}^+ \in [7; 20]$, $\delta_{And}^+ \in [3; 10]$, $\delta_{Or}^+ \in [1; 5]$. Note that, although only 5 parameters vary, the model is fully parametric and contains 12 parameters. Hence, the constraint synthesized is in 12 dimensions.

Latch Circuit. This circuit is an elementary asynchronous circuit studied in [And10c]. The 4 parameters varying in the cartography are as follows: $\delta_{And}^\uparrow \in [60; 100]$, $\delta_{Latch}^\uparrow \in [240; 250]$, $T_{Hold} \in [320, 400]$, $T_{Setup} \in [1, 2]$. Note that a fifth parameter does not vary in the cartography, and the analysis has been performed using all 5 parameters.

CSMA/CD Protocol. We consider here a non-probabilistic version of the CSMA/CD Protocol, studied in the context of probabilistic timed automata in [KNSW07,AFS09]. The 3 parameters are as follows: $\lambda \in [1; 20]$, $\sigma \in [1; 10]$, $slot \in [1, 10]$.

[8] Note that the software named IMITATOR 3 is an independent fork of IMITATOR II specialized for hybrid systems, and does not feature any of the work presented here. Hence, the latest version of IMITATOR for timed automata, including the algorithms presented here, is IMITATOR 2.3.

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

SPSMALL Memory. This corresponds to the model of a memory circuit sold by ST-Microelectronics [And10c]. The property of good behavior corresponds to the guarantee of a given maximum traversal time of the memory by the electric current. Although this property is a timed property, we can apply our method by adding an observer, which goes to a "good" location if the property is satisfied, and to a "bad" one otherwise. Hence, it is straightforward to separate the trace sets (and hence, the tiles) into good and bad, which IMITATOR II performs automatically.

The considered V_0 of SPSMALL is: $t^{D}_{setup} \in [65; 110]$ and $t^{WEN}_{setup} \in [0; 66]$. The SPSMALL' is identical to SPSMALL with a restricted V_0 (actually optimized for the minimization of these two timing parameters): $t^{D}_{setup} \in [89; 98]$ and $t^{WEN}_{setup} \in [25; 34]$. Both models are fully parametric (and contain actually 26 parameters), although only 2 parameters vary in the cartography.

Interpretation. As expected from Section 3, all algorithms bring a significant gain in term of size of the constraint, as shown in Table 2, 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 time is often much quicker for this variant. As expected, the most efficient algorithm in term of computation time and constraint output is $IM^{K}_{\subseteq}$: 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 from Table 2 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; as a consequence, we compute a list of all constraints associated to all the last states of a trace (see Section 4.2). For systems with trace sets containing 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 enormous set before calling (or not) IM on this point. This also explains the relatively disappointing speed performance of $IM^{\cup}_{\subseteq}$. Improving this implementation is the subject of ongoing work. An option would be to remove the constraints equivalent with each other in this constraint set; this would dramatically decrease the size of the set, but would induce additional costs for checking constraint equality.

7 On-the-fly Computation of K

Description. We introduce here another modification of the algorithms in order to avoid the non-necessary duplication of some reachable states. Indeed, we met the issue when states are duplicated while computing $Post^{i}_{\mathcal{A}(K)}(S)$, although they are merged later while computing the final intersection of the constraints on the parameters. This duplication can potentially cause a memory explosion. The cause of this phenomenon is that the two states (q, C) and (q, C') are not equal at the time they are computed but are equal with the addition of K, i.e.

$(q, C \wedge K) = (q, C' \wedge K)$. The modification implemented here solves 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.

However, the practical gain of this modification is not systematic. For case studies where no such states are duplicated, this modification can be very costly, both in time and memory. However, for large case studies, such as the industrial case study SPSMALL (a smaller model of which was mentioned in Section 6), this modification has a dramatic gain in efficiency. Actually, without this modification, the larger model of the SPSMALL memory could not be analyzed because of an explosion of the memory.

Compatibility With Other Variants. This modification is compatible with IM and $IM_{\subseteq}$, which gives IM^{OtF} and $IM_{\subseteq}^{OtF}$ respectively. However, applying this modification to other algorithms would modify their correctness, since the final intersection of the constraints on the parameters is not performed in the other algorithms.

Application to the SPSMALL Memory. We have successfully computed a set of timing parameters for an industrial case study, viz., the SPSMALL memory designed by ST-Microelectronics. We analyzed a much larger version of the "small" model considered in Section 6 (see differences between these models in [And10c]). The larger model of this memory contains 28 automata, 28 clocks and 62 parameters. The computation consists in 98 iterations of the outer **DO** loop of IM. Without the 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 and most machines would have crashed for lack of memory. Using this optimization, we compute quickly a constraint, made of the conjunction of 49 linear constraints. Full details are available in [And10c].

8 Conclusion

We introduce 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, satisfying certain properties. The major advantage of these variants is the faster computation of K_0, and a larger set of parameter valuations defined by K_0. All preserve non-reachability properties: if a (generally bad) location 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. 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 algorithm, these algorithms cover both using less tiles and generally faster the parameter space when synthesizing a constraint w.r.t. a given property.

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.

We presented in [AFS09] an extension of the inverse method to probabilistic systems: given a parametric probabilistic timed automaton $\mathcal{A}$ and a reference parameter 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 $\mathcal{A}[\pi]$. We are interested in studying what properties each of the algorithms presented here preserves in the probabilistic framework.

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.

[ABS01] A. Annichini, A. Bouajjani, and M. Sighireanu. Trex: A tool for reachability analysis of complex systems. In *CAV '01*, pages 368–372. Springer-Verlag, 2001.

[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.

[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. IMITATOR II user manual. Research Report LSV-10-20, Laboratoire Spécification et Vérification, ENS Cachan, France, November 2010.

[And10c] 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.

[CC05] R. Clarisó and J. Cortadella. Verification of concurrent systems with parametric delays using octahedra. In *ACSD '05*. IEEE Computer Society, 2005.

[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.

[Doy07] Laurent Doyen. Robust parametric reachability for timed automata. *Information Processing Letters*, 102(5):208–213, 2007.

[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.

[HHWT95] T. A. Henzinger, P. H. Ho, and H. Wong-Toi. A user guide to HyTech. In *TACAS*, pages 41–71, 1995.

[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.

[KNSW07] M. Kwiatkowska, G. Norman, J. Sproston, and F. Wang. Symbolic model checking for probabilistic timed automata. *Information and Computation*, 205(7):1027–1077, 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.

[Sch86] Alexander Schrijver. *Theory of linear and integer programming.* John Wiley & Sons, Inc., New York, NY, USA, 1986.

[WY03] F. Wang and H.C. Yen. Timing parameter characterization of real-time systems. In *CIAA '03*, volume 2759 of *LNCS*, pages 23–34, 2003.

[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.