Reachability Preservation Based Parameter Synthesis for Timed Automata

Étienne André[1(✉)], Giuseppe Lipari[2], Hoang Gia Nguyen[1], and Youcheng Sun[3]

[1] Université Paris 13, Sorbonne Paris Cité, LIPN, CNRS, UMR 7030, Paris, France
Etienne.Andre@lipn.fr

[2] CRIStAL – UMR 9189, Université de Lille, USR 3380 CNRS, Lille, France

[3] Scuola Superiore Sant'Anna, Pisa, Italy

Abstract. The synthesis of timing parameters consists in deriving conditions on the timing constants of a concurrent system such that it meets its specification. Parametric timed automata are a powerful formalism for parameter synthesis, although most problems are undecidable. We first address here the following reachability preservation problem: given a reference parameter valuation and a (bad) control state, do there exist other parameter valuations that reach this control state iff the reference parameter valuation does? We show that this problem is undecidable, and introduce a procedure that outputs a possibly underapproximated answer. We then show that our procedure can efficiently replace the behavioral cartography to partition a bounded parameter subspace into good and bad subparts; furthermore, our procedure can even outperform the classical bad-state driven parameter synthesis semi-algorithm, especially when distributed on a cluster.

1 Introduction

The design of critical real-time systems is notoriously error-prone, and requires formal verification to assess the absence of undesired behaviors. The theory of timed automata (TA) [1] provided in the past two decades designers with a powerful formalism to formally verify real-time systems. TA extend finite-state automata with clocks that can be compared with integers in guards and invariants. Unfortunately, the classical definition of TA is not tailored to verify systems only partially specified, especially when the value of some timing constants is not yet known. The synthesis of timing parameters consists in deriving conditions on the timing constants of a concurrent system such that it meets its specification. Parametric timed automata (PTA) [2] extend TA by allowing the use of parameters (i.e., unknown constants) in place of integer constants in the model.

This work was partially supported by a BQR grant "SynPaTiC" and by the ANR national research program "PACS" (ANR-2014).

K. Havelund et al. (Eds.): NFM 2015, LNCS 9058, pp. 50–65, 2015.
DOI: 10.1007/978-3-319-17524-9_5

Related Work. The expressive power of PTA comes at the cost of the undecidability of almost all interesting problems. The EF-emptiness problem[1] ("does there exist a parameter valuation such that a control state is reachable?") is undecidable if the model contains as little as three parameterized clocks [2]. Research around PTA since then consisted mainly in either exhibiting subclasses of PTA for which interesting problems become decidable, or devising efficient semi-algorithms that would terminate "often enough" to be useful. A famous subclass of PTA is L/U PTA [8,13] where each parameter can be used only either as upper bounds or as lower bounds, and for which the EF-emptiness problem becomes decidable. In [8], further problems have been shown to be decidable for L/U PTA, including the emptiness and the universality problem for infinite runs properties ("do all parameter valuations have an infinite accepting run?"), for integer parameter valuations. In [14], however, it was shown that the solution to the EF-synthesis problem ("find all parameter valuations such that a control state is reachable") for L/U PTA cannot be represented as a finite union of polyhedra, hence strongly limiting the practical interest of L/U PTA. Orthogonal to syntactical restrictions on the model is the search for restrictions on the parameter domain: in [14], an algorithm is proposed to synthesize integer parameter valuations in a bounded domain. This is of course decidable, and the authors devise two symbolic algorithms that perform better than enumeration.

More practical research on PTA include the development of tools (e.g., Romo [16], IMITATOR [6]) and their application to several fields such as hardware verification (e.g., [10]) and parametric schedulability analysis (e.g., [12]). In [3], we proposed the inverse method IM, a procedure that takes advantage of a reference parameter valuation and generalizes it in the form of a convex constraint, such that the discrete (linear-time) behavior of the system is preserved. In [5], we proposed the behavioral cartography BC: by iterating IM on integer points in a bounded parameter domain, we decompose this domain into constraints such that, for all parameter valuations in each constraint, the discrete behavior is the same. Then, BC can give a (possibly incomplete) solution to the EF-synthesis problem, by returning the union of all constraints for which the desired control state is reachable.

Contribution. In this work, our main goal is to address the EF-synthesis problem. Instead of attacking the state space exploration in a brute force manner (like [2, 14]), we propose to perform several explorations of smaller size, taking advantage of reference valuations in the line of the inverse method. More in details, our contributions are as follows:

1. We first address the following *reachability preservation problem* for PTA: given a reference parameter valuation π and a control state, do there exist other parameter valuations that reach this control state iff π does? We show that this problem is undecidable, and we introduce a procedure PRP (parametric reachability preservation) that gives a (possibly incomplete) answer.

[1] "EF" comes from the CTL syntax and stands for "exists finally".

2. Then, we show that PRP can efficiently replace IM in the behavioral cartography to partition a bounded parameter subspace into good and bad subparts, and give a solution to the EF-synthesis problem.
3. We then compare the PRP-based cartography with the classical parameter synthesis semi-algorithm "EFsynth" [2,14] that solves the EF-synthesis problem: not only PRP gives a more precise result, but it also performs surprisingly well, despite its repeated analyses. Comparisons are performed using parametric schedulability problems for real-time systems.
4. We finally briefly discuss a distributed version of PRP, that is faster and almost always outperforms EFsynth.

Outline. Section 2 recalls PTA, decision problems and existing results. Section 3.1 defines the *reachability preservation problem* and proves its undecidability; Section 3.2 introduces PRP and proves its correctness; Section 3.3 shows that PRP can be used to solve the EF-synthesis problem. Section 4 discusses a distributed version of PRP, and Section 5 describes an experimental comparison with BC and EFsynth. Section 6 concludes the paper and gives perspectives.

2 Preliminaries

Throughout this paper, we assume a set $X = \{x_1, \dots, x_H\}$ of *clocks*, i.e., real-valued variables that evolve at the same rate. A clock valuation w is a function $w : X \to \mathbb{R}_+$. We will often identify a clock valuation π with the *point* $(w(x_1), \dots, w(x_H))$. We denote by $X = 0$ the conjunction of equalities that assigns 0 to all clocks in X. Given $d \in \mathbb{R}_+$, $w + d$ denotes the valuation such that $(w + d)(x) = w(x) + d$, for all $x \in X$.

Throughout this paper, we assume a set $P = \{p_1, \dots, p_M\}$ of *parameters*, i.e., unknown constants. A parameter *valuation* π is a function $\pi : P \to \mathbb{Q}_+$. We will often identify a valuation π with the *point* $(\pi(p_1), \dots, \pi(p_M))$. An *integer* point is a valuation $\pi : P \to \mathbb{N}$.

An *inequality* over X and P is $e \prec 0$, where $\prec \in \{<, \leq, \geq, >\}$, and e is a linear term $\sum_{1 \leq i \leq N} \alpha_i z_i + d$ for some $N \in \mathbb{N}$, where $z_i \in X \cup P$, $\alpha_i \in \mathbb{Q}_+$, for $1 \leq i \leq N$, and $d \in \mathbb{Q}_+$. A (linear) constraint over X and P is a set of linear inequalities over X and P. We define in a similar manner inequalities and constraints over P. A parametric guard is a set of linear inequalities where exactly one z_i is a clock. We denote by $\mathcal{L}(P)$ and $\mathcal{L}(X \cup P)$ the set of all constraints over P, and over X and P respectively. We use $K \in \mathcal{L}(P)$ and $C \in \mathcal{L}(X \cup P)$.

Given a parameter valuation π, $C[\pi]$ denotes the constraint over X 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 π *satisfies* C, denoted by $\pi \models C$, if the set of clock valuations satisfying $C[\pi]$ is nonempty. We use the notation $<w|\pi> \models C$ to indicate that $C[\pi][w]$ evaluates to true.

We denote by $\top$ (resp. $\bot$) the constraint over P that corresponds to the set of all possible (resp. the empty set of) parameter valuations. We denote by $C{\downarrow_P}$ the projection of C onto P, i.e., obtained by eliminating the clock variables. We define the *time elapsing* of C, denoted by $C^\uparrow$, as the constraint over X and P obtained from C by delaying an arbitrary amount of time. Given $R \subseteq X$, we define the *reset* of C, denoted by $[C]_R$, as the constraint obtained from C by resetting the clocks in R, and keeping the other clocks unchanged.

Parametric timed automata are an extension of the class of timed automata to the parametric case, where parameters can be used within guards and invariants in place of constants [2].

Definition 1. *A PTA $\mathcal{A}$ is a tuple $\mathcal{A} = (\Sigma, L, l_0, X, P, I, E)$, where:*

- *Σ is a finite set of actions,*
- *L is a finite set of locations, $l_0 \in L$ is the initial location,*
- *X is a set of clocks, P is a set of parameters,*
- *I is the invariant, assigning to every $l \in L$ a parametric guard $I(l)$,*
- *E is a set of edges (l, g, a, R, l') where $l, l' \in L$ are the source and destination locations, $a \in \Sigma$, $R \subseteq X$ is a set of clocks to be reset, and g is a parametric guard.*

Throughout this paper, we will be interested in the reachability of *bad locations*. We assume a special location $l_{bad} \in L$; without loss of generality, we assume that this location is unique (the case with several bad locations can be reduced to one only using additional transitions to l_{bad}).

Given a PTA $\mathcal{A} = (\Sigma, L, l_0, X, P, I, E)$, and a parameter valuation π, $\mathcal{A}[\pi]$ denotes the TA obtained from $\mathcal{A}$ by substituting every occurrence of a parameter p_i by the constant $\pi(p_i)$ in the guards and invariants.

We borrow from [14] and adapt to our notations the semantics of a TA.

Definition 2 (Semantics of a TA). *Given a PTA $\mathcal{A} = (\Sigma, L, l_0, X, P, I, E)$, and a parameter valuation π, the semantics of $\mathcal{A}[\pi]$ is given by the timed transition system $(Q, q_0, \Rightarrow)$, with*

- $Q = \{(l, w) \in L \times \mathbb{R}_+^H \mid I(l)[\pi][w]$ *evaluates to true*$\}$, $q_0 = (l_0, X = 0)$
- $((l, w), a, (l', w')) \in \Rightarrow$ *if* $\exists w'' : (l, w) \xrightarrow{a} (l', w'') \xrightarrow{d} (l', w')$, *with*
  - *discrete transitions:* $(l, w) \xrightarrow{a} (l', w')$, *with* $a \in \Sigma$, *if* $(l, w), (l', w') \in Q$, *there exists* $(l, g, a, R, l') \in E$, $w' = [w]_R$, *and* $g[\pi][w]$ *evaluates to true.*
  - *delay transitions:* $(l, w) \xrightarrow{d} (l, w + d)$, *with* $d \in \mathbb{R}_+$, *if* $\forall d' \in [0, d], (l, w + d') \in Q$.

A concrete run of a TA is an alternating sequence of states of Q 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 $i = 0, \dots, m-1$, $a_i \in \Sigma$, and $(s_i, a_i, s_{i+1}) \in \Rightarrow$. Given a state $s = (l, w)$, we say that s is reachable (or that $\mathcal{A}[\pi]$ reaches s) if s belongs to a run of $\mathcal{A}[\pi]$; by extension, we say that l is reachable in $\mathcal{A}[\pi]$.

We now recall the semantics of PTA.

Definition 3 (Symbolic state). *A symbolic state of a PTA $\mathcal{A}$ is a pair (l, C) where $l \in L$ is a location, and $C \in \mathcal{L}(X \cup P)$ its associated constraint.*

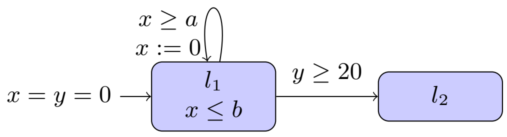


Fig. 1. An example of a PTA $\mathcal{A}_1$ [14]

A state $s = (l, C)$ is π-compatible if $\pi \models C$.

The initial state of $\mathcal{A}$ is $s_0 = (l_0, (X = 0)^{\uparrow} \wedge I(l_0))$.

The computation of the state space relies on the Succ operation. Given a symbolic state $s = (l, C)$, $\mathsf{Succ}(s) = \{(l', C') \mid \exists(l, g, a, R, l') \in E \text{ s.t. } C' = \big([(C \wedge g)]_R\big)^{\uparrow} \cap I(l')\}$. By extension, given a set S of states, $\mathsf{Succ}(S) = \{s' \mid \exists s \in S \text{ s.t. } s' \in \mathsf{Succ}(s)\}$.

A symbolic run of a PTA is an alternating sequence of symbolic 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 such that s_{i+1} belongs to $\mathsf{Succ}(s_i)$ and is obtained via action a_i.

Given a (concrete or symbolic) run $(l_0, C_0) \overset{a_0}{\Rightarrow} (l_1, C_1) \overset{a_1}{\Rightarrow} \cdots \overset{a_{m-1}}{\Rightarrow} (l_m, C_m)$, its corresponding trace is $l_0 \overset{a_0}{\Rightarrow} l_1 \overset{a_1}{\Rightarrow} \cdots \overset{a_{m-1}}{\Rightarrow} l_m$. The set of all traces of a TA is called its *trace set*. Two runs (concrete or symbolic) are said to be equivalent if their associated traces are equal.

Problems for PTA. We recall below two classical problems, as formalized in [14].

Problem 1 (EF-emptiness). Let $\mathcal{A}$ be a PTA. Is the set of parameter valuations π such that $\mathcal{A}[\pi]$ reaches l_{bad} empty?

Problem 2 (EF-synthesis). Let $\mathcal{A}$ be a PTA. Compute the set of parameter valuations π such that $\mathcal{A}[\pi]$ reaches l_{bad}.

Problem 1 is undecidable [2], and the set of parameter valuations solving Problem 2 cannot be computed in general. In [14], the following semi-algorithm is proposed, that gives a complete answer to Problem 2 when it terminates.

$$\mathsf{EFsynth}_{l_{bad}}((l, C), S) = \begin{cases} C{\downarrow_P} & \text{if } l = l_{bad} \\ \emptyset & \text{if } (l, C) \in S \\ \bigcup_{s' \in \mathsf{Succ}((l,C))} \mathsf{EFsynth}_{l_{bad}}\big(s', S \cup \{(l, C)\}\big) & \text{otherwise} \end{cases}$$

Example 1. Consider the PTA $\mathcal{A}_1$ in Fig. 1 [14], with clocks x and y and parameters a and b. Then $\mathsf{EFsynth}_{l_2}(s_0, \emptyset)$ does not terminate, and neither does it if the range of the parameters is bounded from above (e.g., $a, b \in [0, 50]$).

From the proof of correctness of $\mathsf{EFsynth}$ in [14], one can infer that the result of $\mathsf{EFsynth}$ is still a (possibly incomplete) answer to Problem 2 even when the algorithm is artificially stopped before its termination. By artificially stopping

$\mathsf{EFsynth}$, we mean bounding the recursion depth: when the depth indeed exceeds some bound, we replace the recursive call $\mathsf{EFsynth}_{l_{bad}}(s', S \cup \{(l, C)\})$ with $\perp$.

Proposition 1. *Let K be the result of $\mathsf{EFsynth}_{l_{bad}}(s_0, \emptyset)$ when $\mathsf{EFsynth}$ is stopped after being recursively called a bounded number of times. For all $\pi \models K$, l_{bad} is reachable in $\mathcal{A}[\pi]$.*

Behavioral Cartography. In [3], we introduced the inverse method IM. This procedure takes as input a reference parameter valuation π and outputs a constraint K such that 1) $\pi \models K$ and 2) for all $\pi' \models K$, the trace sets of $\mathcal{A}[\pi]$ and $\mathcal{A}[\pi']$ are the same; hence, the discrete (linear-time) behavior of the system is preserved. IM performs a breadth-first exploration of the symbolic state space of $\mathcal{A}$; whenever a π-incompatible state (l, C) is met, it is removed as follows: a π-incompatible inequality is selected within the projection of C onto P, and then its negation is added to a constraint maintained by IM. When a fixpoint is reached, IM returns the intersection of all parametric constraints associated to the remaining symbolic states.

A variant of IM named IM^K outputs a weaker (i.e., larger) constraint, that only guarantees that any trace of $\mathcal{A}[\pi']$ is a trace of $\mathcal{A}[\pi]$ [7]. It is similar to IM except that, instead of returning the intersection of all parametric constraints, it returns only the accumulation of π-incompatible inequalities. Hence, IM^K only forbids the traces not possible under π, without requiring that all traces of $\mathcal{A}[\pi]$ be possible in $\mathcal{A}[\pi']$.

In [5], we introduced the behavioral cartography BC: by iterating IM on the integer points in a bounded parameter domain V (usually a product of intervals in $|P|$ dimensions), one can decompose V into tiles, i.e., parametric constraints in which the discrete behavior is uniform. Hence all parameter valuations in a tile satisfy the same set of linear-time properties. Then, given such a property (expressed using, e.g., LTL), one can partition V into good and bad tiles depending whether this property is or not satisfied in each tile.

This method has two theoretical drawbacks: first, some calls to IM may not terminate and, second, BC does not formally guarantee that any "dense" part of V will be covered beside the integer points. However, in practice not only the whole dense part of V is almost always covered, but large (infinite) parts of the parameter space beyond V are often covered.

3 Solving the EF-Emptiness Problem Using Reachability Preservation

3.1 Undecidability of the Preservation of Reachability

Parameter synthesis with respect to a bad location is known to be undecidable [2]. Here, we take advantage of a reference parameter valuation π, for which it is possible to decide whether l_{bad} is reachable [1]. The assumption of a known parameter valuation seems realistic to us: in system design, it is often the case that one knows (from a previous design, of using empirical methods) a first valuation; however, finding other valuations may be much more difficult, and may require to restart the design phase from zero. Here, given a reference parameter valuation, we are interested in the preservation of the reachability of l_{bad} by other parameter valuations. Given two TA $\mathcal{A}[\pi]$ and $\mathcal{A}[\pi']$, we say that $\mathcal{A}[\pi']$ *preserves the reachability* of l_{bad} in $\mathcal{A}[\pi]$ when l_{bad} is reachable in $\mathcal{A}[\pi]$ if and only if l_{bad} is reachable in $\mathcal{A}[\pi']$. We call PREACH the problem of the preservation of reachability. In the following, we show that, given π, deciding whether at least one parameter valuation $\pi' \neq \pi$ preserves the reachability of l_{bad} in $\mathcal{A}[\pi]$ is undecidable.

Problem 3 (PREACH-emptiness). Let $\mathcal{A}$ be a PTA, and π a parameter valuation. Does there exist $\pi' \neq \pi$ such that $\mathcal{A}[\pi']$ preserves the reachability of l_{bad} in $\mathcal{A}[\pi]$?

Problem 4 (PREACH-synthesis). Let $\mathcal{A}$ be a PTA, and π a parameter valuation. Compute the set of parameter valuations π' such that $\mathcal{A}[\pi']$ preserves the reachability of l_{bad} in $\mathcal{A}[\pi]$.

We show below that Problem 3 is undecidable.

Theorem 1. *PREACH-emptiness is undecidable.*

Proof. Given a parameter valuation reaching some location, we reduce the existence of a different parameter valuation reaching the same location from the halting problem of a 2-counter machine.

1. First, recall that [2] defines the encoding of a 2-counter machine (2CM) using a PTA $\mathcal{A}_{2CM}$ that contains two parameters a and b.[2] Then [2] shows that the 2CM halts iff there exists at least one non-null parameter valuation such that a special location l_{halt} is reachable in $\mathcal{A}_{2CM}$.
2. Now, let us add a gadget to $\mathcal{A}_{2CM}$ that adds a direct transition from the initial location l_0 to l_{halt} with a guard $a = b = 0$.[3] Let $\mathcal{A}$ be this new PTA, as depicted in Fig. 2. Now, we have:
   (a) If the 2CM halts, then l_{halt} is still reachable in $\mathcal{A}$ for some non-null parameter valuation since it was already reachable in $\mathcal{A}_{2CM}$. Additionally, due to our gadget, l_{halt} is also reachable in $\mathcal{A}$ for $a = b = 0$.
   (b) If the 2CM does not halt, l_{halt} is again reachable in $\mathcal{A}$ for $a = b = 0$ due to our gadget, but no other parameter valuation can reach l_{halt}, just as in item 1.

   Hence, given $\pi : a = b = 0$, there exists a parameter valuation $\pi' \neq \pi$ such that $\mathcal{A}[\pi']$ preserves the reachability of l_{halt} in $\mathcal{A}[\pi]$ iff the 2CM halts.

[2] Strictly speaking, their construction uses six parameters, but it is well-known (shown, e.g., in [14]) that they can be reduced to two.

[3] This guard is not allowed in PTA, but can be simulated using an extra clock x and an urgent location followed by a transition with guard $x = a \wedge x = b$.

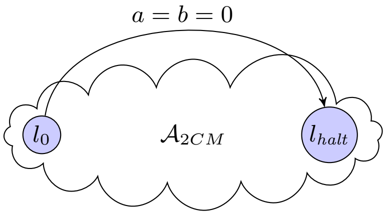


Fig. 2. Undecidability of PREACH-emptiness: PTA $\mathcal{A}$

3.2 Parameter Synthesis Preserving the Reachability

To propose a solution to Problem 4, we introduce here $\mathsf{PRP}(\mathcal{A}, \pi)$, that is inspired by two existing algorithms, viz., EFsynth and the variant IM^K of IM [7]. PRP (standing for parametric reachability preservation) is at first close to IM^K, and then switches to an algorithm that resembles EFsynth:

– As long as no bad location is reached, PRP generalizes the trace set of $\mathcal{A}[\pi]$ by removing π-incompatible states; this is done by negating π-incompatible inequalities, and returning the intersection of such negated inequalities, in the line of IM^K.
– When at least one bad location is met, PRP switches to an algorithm close to EFsynth, i.e., it simply gathers the constraints associated with the bad locations, and returns their union. However, a main difference with EFsynth is that PRP does not explore π-incompatible states: although this is not necessary to ensure correctness (in fact, this makes PRP not complete), this is a key heuristics to keep the state space of reasonable size.

We introduce PRP in Algorithm 1. It is a breadth-first exploration procedure that maintains the following variables: S (resp. S_{new}) is the set of states computed at the previous (resp. current) iterations; Bad is a Boolean flag that remembers whether a bad location has been met; K_{good} is the intersection of the negation of all π-incompatible inequalities, that will be returned if no bad state is met; K_{bad} is the union of the projection onto P of all bad states, that will be returned otherwise; i remembers the current exploration depth.

The procedure consists in a (potentially infinite) **while** loop. First, lines 3–4 take care of the π-incompatible states and resembles IM^K. These states are discarded from the exploration, i.e., they are removed from the set of new states (line 4). Then, if the exploration has not yet met any bad state, K_{good} is refined so as to prevent any such π-incompatible state (l, C) to be reached: a π-incompatible inequality J is selected within the projection of C onto P, and then its negation is added to K_{good}. This mechanism is borrowed to IM (and its variant IM^K).

Second, lines 8–9 take care of the bad states. If any bad state is reached (line 8), then the Bad flag is set to true, the union of the projection onto P of the constraints associated with these bad states is added to K_{bad}, and these states are discarded, i.e., their successor states will not be computed (line 9).

Algorithm 1. $\mathsf{PRP}(\mathcal{A}, \pi)$

```
input  : PTA $\mathcal{A}$ of initial state $s_0$, parameter valuation $\pi$
output : Constraint over the parameters
1  $S \leftarrow \emptyset$ ;  $S_{new} \leftarrow \{s_0\}$ ;  $Bad \leftarrow$ false ;  $K_{good} \leftarrow \top$ ;  $K_{bad} \leftarrow \bot$ ;  $i \leftarrow 0$
2  while true do
3      foreach $\pi$-incompatible state $(l, C)$ in $S_{new}$ do
4          $S_{new} \leftarrow S_{new} \setminus \{(l, C)\}$
5          if $Bad =$ false then
6              Select a $\pi$-incompatible inequality $J$ in $C{\downarrow_P}$ (i.e., s.t. $\pi \not\models J$)
7              $K_{good} \leftarrow K_{good} \wedge \neg J$
8      foreach bad state $(l_{bad}, C)$ in $S_{new}$ do
9          $Bad \leftarrow$ true ;  $K_{bad} \leftarrow K_{bad} \vee C{\downarrow_P}$ ;  $S_{new} \leftarrow S_{new} \setminus \{(l_{bad}, C)\}$
10     if $S_{new} \subseteq S$ then
11         if $Bad =$ true then return $K_{bad}$ else return $K_{good}$ ;
12     $S \leftarrow S \cup S_{new}$ ;  $S_{new} \leftarrow \mathsf{Succ}(S_{new})$ ;  $i \leftarrow i + 1$
```

The third part is a classical fixpoint condition: if no new state has been met at this iteration (line 10), then the result is returned, i.e., either K_{bad} if some bad states have been met, or K_{good} otherwise. If new states have been met, then the procedure explores one step further in depth (line 12).

We will show in Theorem 2 that PRP outputs a sound (though possibly incomplete) answer to Problem 4. In fact, PRP verifies a stronger property: if l_{bad} is reachable in $\mathcal{A}[\pi]$, PRP outputs a constraint K guaranteeing that l_{bad} is reachable for any parameter valuation satisfying K. However, if l_{bad} is unreachable in $\mathcal{A}[\pi]$, the constraint K output by PRP satisfies the same property as IM^K, i.e., the trace set of $\mathcal{A}[\pi']$ is a subset of the trace set of $\mathcal{A}[\pi]$, for all $\pi' \models K$. This is formalized in Proposition 2.

Proposition 2. *Let $\mathcal{A}$ be a PTA, and π a parameter valuation. Suppose PRP $(\mathcal{A}, \pi)$ terminates with result K. Then, $\pi \models K$ and, for all $\pi' \models K$:*

- *if l_{bad} is reachable in $\mathcal{A}[\pi]$, then l_{bad} is reachable in $\mathcal{A}[\pi']$;*
- *if l_{bad} is unreachable in $\mathcal{A}[\pi]$, then every trace of $\mathcal{A}[\pi']$ is a trace of $\mathcal{A}[\pi]$.*

Theorem 2. *Let $\mathcal{A}$ be a PTA, and π a parameter valuation. Suppose PRP$(\mathcal{A}, \pi)$ terminates with result K. Then, $\pi \models K$ and, for all $\pi' \models K$, l_{bad} is reachable in $\mathcal{A}[\pi]$ iff l_{bad} is reachable in $\mathcal{A}[\pi']$.*

Proof. From Proposition 2.

Remark 1. PRP may not terminate, which is natural since Problem 3 is undecidable. Furthermore, even if it terminates, the result output by PRP may be non complete; in fact, this is designed on purpose (since we stop the exploration of π-incompatible states) so as to prevent a too large exploration. Enlarging the output constraint can be done by repeatedly calling PRP on other points than π, which will be done in Section 3.3.

Example 2. Let us apply PRP to the PTA $\mathcal{A}_1$ in Fig. 1. For point $\pi_1 : (a = 20, b = 10)$, PRP outputs constraint $20 > b \wedge a > b \wedge b \geq 0$, which guarantees the unreachability of l_{bad}. For point $\pi_2 : (a = 30, b = 30)$, PRP outputs constraint $b > 20 \wedge a \geq 0$, which guarantees the reachability of l_{bad}. For point $\pi_3 : (a = 0, b = 40)$, PRP does not terminate.

We now state in Theorem 3 that, even when PRP is interrupted before its termination, PRP outputs a sound (though possibly incomplete) answer to Problem 4, provided some bad states have already been met. The result comes from the fact that the first item of the proof of Proposition 2 holds even if PRP has not terminated. (Note that the converse case, when $Bad = \texttt{false}$, does not hold if PRP has not terminated: although no bad state has been met yet, there could be some in the future.)

Theorem 3. *Let $\mathcal{A}$ be a PTA, and π a parameter valuation. Let be K the value of K_{bad} at the end of iteration i of PRP$(\mathcal{A}, \pi)$, for some $i \geq 0$, such that $Bad =$ `true`. Then: 1) l_{bad} is reachable in $\mathcal{A}[\pi]$, and 2) for all $\pi' \models K$, l_{bad} is reachable in $\mathcal{A}[\pi']$.*

Example 3. Let us again apply PRP to the PTA $\mathcal{A}_1$ in Fig. 1. For this PTA and $\pi_3 : (a = 0, b = 40)$, PRP with a depth limit of 10 terminates with $Bad =$ `true`. From Theorem 3, the output constraint is valid, i.e., guarantees the reachability of l_{bad}.

3.3 EF-Synthesis Using PRP

Given a bounded parameter domain, IM can be iterated on integer points to perform a behavioral cartography; then, the tiles can be partitioned in good and bad according to a linear-time property. If the property of interest is simply a (non-)reachability property, then PRP can be used in place of IM within BC, giving birth to a procedure PRPC (see Algorithm 2). PRP is called repeatedly with as an argument the first integer point not yet covered by any constraint (line 2 in Algorithm 2).

The "cartography" output by PRPC is less precise than the one output by the classical BC, because the constraints outputs by PRP are not tiles anymore: Theorem 2 only guarantees the preservation of reachability, and hence different parameter valuations within a constraint may correspond to different trace sets. To output a set of parameter valuations solving EF-synthesis, it suffices to return the union of the constraints for which l_{bad} is reachable.

Now, a key feature of PRPC is to explore a relatively small part of the whole parametric state space at a time, and to still output larger constraints than BC. We will show in Section 5 that using PRP instead of IM in the cartography indeed dramatically increases its efficiency.

Remark 2. In the general case, PRPC may not terminate, due to the non-termination of PRP. However, it is possible to set up a maximum exploration depth for PRP: when this depth is reached, the algorithm stops. If some bad states have

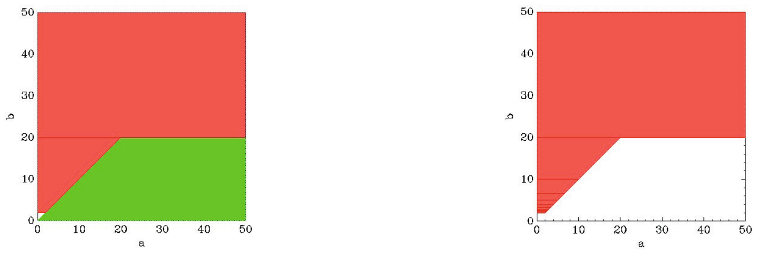
Algorithm 2. $\mathsf{PRPC}(\mathcal{A}, V)$

```
input  : PTA $\mathcal{A}$, bounded parameter domain $V$
output : Set $\mathcal{C}$ of constraints over the parameters (initially empty)
1 while there are integer points in $V$ not covered by $\mathcal{C}$ do
2     Select an integer point $\pi$ in $V$ not covered by $\mathcal{C}$
3     $\mathcal{C} \leftarrow \mathcal{C} \cup \mathsf{PRP}(\mathcal{A}, \pi)$
4 return $\mathcal{C}$
```

been met, the resulting constraint can be safely used (from Theorem 3); otherwise the constraint is just discarded and the reference point on which PRP was called will never be covered. In this case, termination of PRPC is always guaranteed, with a partial result (some integer points may still be uncovered).

Let us now compare EFsynth and PRPC, that can both output (possibly incomplete) solutions to the EF-synthesis problem. On the one hand, EFsynth should be faster (although we will see in Section 5 that it is not even true in general), because it performs only one exploration, whereas PRPC has to launch PRP on many integer points. On the other hand, PRPC will use less memory, since a smaller part of the state space is explored at a time (due to the non-exploration of π-incompatible states). Furthermore, its main interest is that it synthesizes a more valuable result: whereas EFsynth outputs only a possibly under-approximated set of bad parameter valuations (reaching l_{bad}) and leaves the whole rest of parameter valuations unknown, PRPC outputs possibly under-approximated sets of both bad and good parameter valuations, giving much more valuable information. Finally, just as BC, PRPC can possibly cover parameter valuations beyond the limits of V, which is not possible for EFsynth.

Example 4. Consider again the PTA $\mathcal{A}_1$ in Fig. 1, and let us apply EFsynth and PRPC with a bounded exploration depth of 10; recall that this is safe from Proposition 1 and Theorem 3. We apply PRPC to an unconstrained model with $V : a, b \in [0, 50]$. We apply EFsynth to a model where a and b are constrained to be in $[0, 50]$. We give in a graphical manner in Fig. 3a (resp. Fig. 3b) the results output by PRPC (resp. EFsynth). PRPC synthesizes all the good parameter valuations (below, in green), i.e., that do not reach l_2, and all the bad parameter valuations (above, in red), i.e., that reach l_2, with the exception of a small area near $(0, 0)$ (in white). All constraints output by PRPC are infinite (which is not shown in the figure), and hence cover the whole part outside V too. As of EFsynth, the same bad valuations as for PRPC are covered, but only within V, and no information is given about the good valuations. Hence, since EFsynth was stopped prematurely, no information can be given for the non-covered part: in particular, the white part of V cannot be decided, whereas PRPC covers everything except the small area near $(0, 0)$. This is a major advantage of PRPC over EFsynth in terms of precision of the result. Also recall that EFsynth covers only (a part of) V whereas PRPC covers here the whole parameter space beyond V.



(a) PRPC (b) EFsynth

Fig. 3. EF-synthesis using PRPC and EFsynth for $\mathcal{A}_1$

4 Towards Distributed Parameter Synthesis

In [4], we proposed two distribution algorithms to execute BC on a set of computers (e.g., on a cluster), implemented in Imitator using the message passing interface (MPI). Distributing BC is intrinsically easy: it is trivial that two executions of IM from two different parameter valuations can be performed on two different nodes. However, distributing it *efficiently* is challenging: calling two executions of IM from two contiguous integer points has a very large probability to yield the same tile in both cases, and hence to result in a loss of time for one of the two nodes. Hence, the critical question is how to distribute efficiently the reference valuations ("points") on which to call IM. In [4], we proposed a master-workers scheme, where a master distributes the points to the workers, using two point distribution algorithms:

1. A sequential point enumeration: each integer point not yet covered by any tile is sent to a worker, i.e., $(0, 0)$, then $(0, 1)$ and so on (in two dimensions). This algorithm suffers from the aforementioned problem of close integer points, but still performs reasonably well (up to 7 times faster using 36 nodes).
2. A random point distribution followed by a sequential enumeration: points are selected randomly and, when points not yet covered by any tile become scarce, the master switches to a sequential point enumeration to ensure that all integer points are covered. The fact that the points not covered by any tile become scarce is detected after the number of unsuccessful attempts to randomly choose an uncovered point goes beyond a certain threshold (e.g., 100). This algorithm performs better (up to 12 times faster using 36 nodes).

Here, we will use a third master-workers distribution method, that dynamically splits the parametric domain V in subparts: when a worker completes the covering of its subpart, the master splits another subpart into two parts, and assigns one of the two part to that worker. From our results, this algorithm (implemented in the working version of Imitator) is more efficient than the two algorithms of [4].

Remark 3 (Fairness). Of course, comparing a distributed algorithm (PRPC) with a monolithic one (EFsynth) is unfair. However, to the best of our knowledge, no distributed algorithm for parameter synthesis has been proposed (except [4]). One could argue that EFsynth could at least take advantage of multi-cores, e.g., using one core to compute the successor states while another performs the (costly) equality check, or by computing in parallel the successor states of several states – but PRPC could take advantage of exactly the same enhancements.

5 Experimental Comparison

We compare here several algorithms to solve the EF-synthesis problem using Imitator [6]. In its latest version, Imitator implements EFsynth, BC and PRPC, and can run PRPC in a distributed fashion. Experiments were run using Imitator 2.6.2 (build 845) on a Linux-based cluster. The nodes of this cluster feature two 6-core Intel Xeon X5670 running at 2.93 GHz CPUs (therefore, 12 cores in a NUMA fashion). Each node has 24 GiB of memory and runs a 64-bit Linux 3.2 kernel. The code was compiled using OCaml 3.12.1. The message-passing library we used is Bull's OpenMPI variant for Bullx, and the nodes are interconnected by a 40 Gb/s InfiniBand network.[4]

5.1 Case Studies

Our first case study is the PTA $\mathcal{A}_1$ in Fig. 1, with $V : a, b \in [0, 50]$.

Sched1 and Sched2 are two parametric schedulability problems on a single processor. The goal is to synthesize task parameter valuations guaranteeing that every task meets its relative deadline. For Sched1, we consider two parameters D_2 and T_2 that correspond to the relative deadline and the period of task 2 respectively. We set V to $D_2, T_2 \in [20, 100]$. For Sched2 (adapted from the example studied in [9, 14]), we consider two parameters b and z, which correspond to upper bounds on the execution time of tasks 1 and 3, that is $C_1 \in [10, b]$ and $C_3 \in [20, z]$. A third parameter (always valuated in our experiments) is a, that is used in the relative deadline and the period of tasks 1, 2, 3. Precisely: $D_1 = T_1 = a$, $D_2 = T_2 = 2a$ and $D_3 = T_3 = 3a$. Finally, task τ_2 has a release jitter $J_2 \in \{0, 2\}$. We will study Sched2 with two different V. First, we valuate $a = 50$, we set $V : b \in [10, 50], z \in [20, 100]$ and we synthesize parameters for both $J_2 = 0$ ("Sched2.50.0") and $J_2 = 2$ ("Sched2.50.2"). Second, we valuate $a = 100$, we set $V : b \in [10, 1000], z \in [20, 1000]$ and we consider $J_2 = 0$ ("Sched2.100.0") and $J_2 = 2$ ("Sched2.100.2").

Sched5 models the schedulability of 5 fixed-priority tasks in a single processor. SPSMALL is a model of an asynchronous memory circuit [10].

[4] Sources, binaries, models and results are available at www.lipn.fr/~andre/PRP/.

Table 1. Comparison of algorithms to solve the EF-synthesis problem

Case study	$\lvert H\rvert$	$\lvert V\rvert$	EFsynth	BC	PRPC	PRPC distr(12)
$\mathcal{A}_1$	2	2,601	0.401*	TO	0.078*	0.050*
Sched1	13	6,561	TO	TO	1595	219
Sched2.50.0	6	3,321	9.25	990	14.55	4.77
Sched2.50.2	6	3,321	662	TO	213	84
Sched2.100.0	6	972,971	21.4	2093	116	10.1
Sched2.100.2	6	972,971	3757	TO	4557	1543
Sched5	21	1,681	352	TO	TO	917
SPSMALL	11	3,082	7.49	587	118	11.2

5.2 Summary of the Experiments and Discussion

Table 1 gives from left to right the case study, the number of clocks, the number of integer points in V and the computation time in seconds for EFsynth, BC, PRPC, and the distributed version of PRPC using the part-splitting point distribution running on 12 nodes. "TO" indicates a timeout ($> 5000\,s$).

For $\mathcal{A}_1$, none of the algorithms terminate; hence, termination is ensured by bounding the exploration depth to 10 (marked with * in Table 1). From Proposition 1 and Theorem 3, the result is still correct; however, this does not hold for BC. For the other case studies, all algorithms terminate (except in case of timeouts), and always cover entirely V. To allow a fair comparison, parameters for EFsynth are bounded in the model as in V; without these bounds, EFsynth never terminates for these case studies.

First, we see that PRPC dramatically outperforms BC for all case studies. This is due to the fact that the constraints output by PRP (that preserve only non-reachability) are much weaker than those output by IM (that preserve trace set equality). Second, we see that PRPC compares rather well with EFsynth, and is faster on three case studies; PRPC furthermore outputs a more valuable constraint for $\mathcal{A}_1$ (see Example 4). PRPC can even verify case studies that EFsynth cannot (Sched1).

The distributed version of PRPC is faster than PRPC for all case studies. Most importantly, the distributed PRPC outperforms EFsynth for all but two case studies. The good timing efficiency of PRPC is somehow surprising, since it was devised to output a more precise result and to use less memory, but not necessarily to be faster. We believe that PRPC allows to explore small state spaces at a time and, despite the repeated executions, this is less costly than handling a large state space (as in EFsynth), especially when performing equality checks when a new state is computed.

6 Conclusion

In this work, we address the synthesis of timing parameters for reachability properties. We introduce PRP that outputs an answer to the parameter synthesis problem of the preservation of the reachability of some bad control state l_{bad}, which we showed to be undecidable. By repeatedly iterating PRP on some (integer) points, one can cover a bounded parameter domain with constraints guaranteeing either the reachability or the non-reachability of l_{bad}. This approach competes well in terms of efficiency with the classical bad state synthesis EFsynth, and gives a more precise result than EFsynth while using less memory. Finally, our distributed version almost always outperforms EFsynth.

The approach recently proposed to synthesize parameters using IC3 for reachability properties [11] looks promising; it would be interesting to investigate a combination of that work with a PRP-like procedure, especially if distributed.

So far, we only investigated the preservation of the reachability; investigating infinite runs properties is of interest too. In this case, it would be interesting to combine our distributed setting with the multi-core algorithm recently proposed for (non-parametric) timed automata [15].

Acknowledgments. We thank Camille Coti for a valuable help while using the Magi cluster, and Didier Lime for useful comments on Section 3.1.

References

1. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science **126**(2), 183–235 (1994)
2. Alur, R., Henzinger, T.A., Vardi, M.Y.: Parametric real-time reasoning. In: STOC, pp. 592–601. ACM (1993)
3. André, É., Chatain, T., Encrenaz, E., Fribourg, L.: An inverse method for parametric timed automata. IJFCS **20**(5), 819–836 (2009)
4. André, É., Coti, C., Evangelista, S.: Distributed behavioral cartography of timed automata. In: EuroMPI/ASIA 201414, pp. 109–114. ACM (2014)
5. André, É., Fribourg, L.: Behavioral cartography of timed automata. In: Kučera, A., Potapov, I. (eds.) RP 2010. LNCS, vol. 6227, pp. 76–90. Springer, Heidelberg (2010)
6. André, É., Fribourg, L., Kühne, U., Soulat, R.: IMITATOR 2.5: a tool for analyzing robustness in scheduling problems. In: Giannakopoulou, D., Méry, D. (eds.) FM 2012. LNCS, vol. 7436, pp. 33–36. Springer, Heidelberg (2012)
7. André, É., Soulat, R.: Synthesis of timing parameters satisfying safety properties. In: Delzanno, G., Potapov, I. (eds.) RP 2011. LNCS, vol. 6945, pp. 31–44. Springer, Heidelberg (2011)
8. Bozzelli, L., La Torre, S.: Decision problems for lower/upper bound parametric timed automata. Formal Methods in System Design **35**(2), 121–151 (2009)
9. Bucci, G., Fedeli, A., Sassoli, L., Vicario, E.: Timed state space analysis of real-time preemptive systems. Transactions on Software Engineering **30**(2), 97–111 (2004)
10. Chevallier, R., Encrenaz-Tiphène, E., Fribourg, L., Xu, W.: Timed verification of the generic architecture of a memory circuit using parametric timed automata. Formal Methods in System Design **34**(1), 59–81 (2009)
11. Cimatti, A., Griggio, A., Mover, S., Tonetta, S.: Parameter synthesis with IC3. In: FMCAD, pp. 165–168. IEEE (2013)
12. Cimatti, A., Palopoli, L., Ramadian, Y.: Symbolic computation of schedulability regions using parametric timed automata. In: RTSS, pp. 80–89. IEEE Computer Society (2008)

13. Hune, T., Romijn, J., Stoelinga, M., Vaandrager, F.W.: Linear parametric model checking of timed automata. JLAP **52–53**, 183–220 (2002)
14. Jovanović, A., Lime, D., Roux, O.H.: Integer parameter synthesis for timed automata. IEEE Transactions on Software Engineering (2014, to appear)
15. Laarman, A., Olesen, M.C., Dalsgaard, A.E., Larsen, K.G., van de Pol, J.: Multi-core emptiness checking of timed büchi automata using inclusion abstraction. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 968–983. Springer, Heidelberg (2013)
16. Lime, D., Roux, O.H., Seidner, C., Traonouez, L.-M.: Romeo: a parametric model-checker for petri nets with stopwatches. In: Kowalewski, S., Philippou, A. (eds.) TACAS 2009. LNCS, vol. 5505, pp. 54–57. Springer, Heidelberg (2009)