# Behavioral Cartography of Timed Automata*

Étienne André and Laurent Fribourg

LSV – ENS de Cachan & CNRS, France

**Abstract.** We aim at finding a set of timing parameters for which a given timed automaton has a "good" behavior. We present here a novel approach based on the decomposition of the parametric space into *behavioral tiles*, i.e., sets of parameter valuations for which the behavior of the system is uniform. This gives us a *behavioral cartography* according to the values of the parameters. It is then straightforward to partition the space into a "good" and a "bad" subspace, according to the behavior of the tiles. We extend this method to probabilistic systems, allowing to decompose the parametric space into tiles for which the minimal (resp. maximal) probability of reaching a given location is uniform. An implementation has been made, and experiments successfully conducted.

## 1 Introduction

The admissible behaviors of timed automata are determined by sets of linear constraints over timing parameters. The parameters in these constraints represent constants or values chosen by the designer. The behavior is very sensitive to the values of these parameters, and it is rather difficult to find their correct values. The *good parameters problem* is the following (see [14]): Given a parametrized timed automaton $\mathcal{A}$ and a rectangular real-valued parameter domain, what is the largest set of parameters values for which $\mathcal{A}$ behaves well? We say that $\mathcal{A}$ behaves well if it satisfies a certain set of properties. We are interested here in properties that are invariant for automata having the same set of *traces* (alternating sequences of locations and actions, i.e., time-abstract runs) [3]. This is in particular the case of linear-time properties [7].

*Related Work.* The parameter design problem for timed automata was formulated in [15], where a straightforward solution is given, based on the generation of the whole parametric state space until a fixpoint is reached. Unfortunately, in all but the most simple cases, this is is prohibitively expensive due, in particular, to the brute exploration of the whole parametric state space.

The problem of parameter synthesis for timed automata has been applied to two main domains: telecommunication protocols and asynchronous circuits. For example, concerning telecommunication protocols, the Bounded Retransmission Protocol has been verified in [24] using Uppaal [22] and Spin [17], and the Root

* This work is partially supported by the Agence Nationale de la Recherche, grant ANR-06-ARFU-005.

Contention Protocol in [13] using TReX [6]. Concerning asynchronous circuits, Clarisó and Cortadella have proposed methods with approximations [11].

In [14], the authors propose an extension based on the *counterexample guided abstraction refinement* (CEGAR) [12]. 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.

We propose here an alternative approach. We generate a constraint on the parameters ("tile") for each integer point located within a given rectangle $V_0$. Such a tile is called "behavioral tile" because $\mathcal{A}$ behaves similarly under any parameter valuation corresponding to a point of the tile: the sets of traces coincide [3]. This allows us to decompose the parametric space into behavioral tiles. Then, it is easy to partition the parametric space into a subset of "good" tiles (which correspond to "good behaviors") and a subset of "bad" ones. Often in practice, what is covered is not the *bounded* and *integer* subspace of the parameter rectangle, but two major extensions: first, not only the integer points but all the *real-valued* points of the rectangle is covered by the tiles; second, the tiles are often unbounded and cover most of the parametric space beyond $V_0$.

*Plan of the Paper.* We first recall Parametric Timed Automata in Section 2. We then state the good parameters problem in Section 3 using an example of flip-flop circuit. We then present the behavioral cartography algorithm in Section 4, apply it to the example, and give a sufficient condition to get a full coverage of the parametric space. We present an extension to probabilistic systems in Section 5, summarize experiments in Section 6, and conclude in Section 7.

## 2 Parametric Timed Automata

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 $w : X \rightarrow \mathbb{R}_{\geq 0}$ assigning a non-negative real value to each clock.

Throughout this paper, we assume a fixed set $P = \{p_1, \dots, p_M\}$ of *parameters*. A *parameter valuation* $\pi$ is a function $\pi : P \rightarrow \mathbb{R}_{\geq 0}$ 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 $\pi$ with the point $(\pi(p_1), \dots, \pi(p_M))$.

**Definition 1 (Constraint).** *A* linear inequality on the parameters *$P$ (resp.* linear inequality on the clocks *$X$* and parameters *$P$) is an inequality $e \prec e'$, where $\prec \in \{<, \leq\}$, and $e, e'$ are two linear terms of the form*

$$\Sigma_i \alpha_i p_i + c, \qquad (resp.\ \ \Sigma_i \alpha_i p_i + \Sigma_j \beta_j x_j + c)$$

*where $1 \leq i \leq M, 1 \leq j \leq H$ and $\alpha_i, \beta_j, c \in \mathbb{N}$. A* constraint on the parameters *(resp.* constraint on the clocks and parameters*) is a conjunction of inequalities on $P$ (resp. on $X$ and $P$).*

Given a parameter valuation $\pi$ and a constraint $C$, $C[\pi]$ denotes the constraint 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 $w(x)$. A clock valuation $w$ *satisfies* constraint $C[\pi]$ (denoted by $w \models C[\pi]$) if $C[\pi][w]$ evaluates to true. We say that a parameter valuation $\pi$ *satisfies* a constraint $C$, denoted by $\pi \models C$, if the set of clock valuations that satisfy $C[\pi]$ is nonempty.

Likewise, we say that a parameter valuation $\pi$ *satisfies* a constraint $K$ on the parameters, denoted by $\pi \models K$, if the expression obtained by replacing each parameter $p$ in $K$ with $\pi(p)$ evaluates to true. We consider *True* as a constraint on the parameters, corresponding to the set of all possible values for $P$.

We assume familiarity with timed automata [1]. The following definition is an extension of timed automata to the parametric case. Parametric timed automata allow within guards and invariants the use of parameters in place of constants [2].

**Definition 2 (PTA).** *Given a set of clocks $X$ and a set of parameters $P$, a* parametric timed automaton (PTA) *$\mathcal{A}$ is a 6-tuple of the form $\mathcal{A} = (\Sigma, Q, q_0, K, I, \rightarrow)$, where $\Sigma$ is a finite set of actions, $Q$ is a finite set of locations, $q_0 \in Q$ is the initial location, $K$ is a constraint on the parameters, $I$ is the invariant assigning to every $q \in Q$ a constraint $I_q$ on the clocks and the parameters, and $\rightarrow$ is a step relation consisting in elements of the form $(q, g, a, \rho, q')$ 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 the clocks and the parameters.*

In the sequel, we consider the PTA $\mathcal{A} = (\Sigma, Q, q_0, K, I, \rightarrow)$. We simply denote this PTA by $\mathcal{A}(K)$, in order to emphasize the fact that only $K$ will change in $\mathcal{A}$.

For every parameter valuation $\pi = (\pi_1, \ldots, \pi_M)$, $\mathcal{A}[\pi]$ denotes the PTA $\mathcal{A}(K)$, where $K$ is $\bigwedge_{i=1}^{M} p_i = \pi_i$. This corresponds to the PTA obtained from $\mathcal{A}$ by substituting every occurrence of a parameter $p_i$ by constant $\pi_i$ in the guards and invariants. We say that $p_i$ is *instantiated* with $\pi_i$. Note that, as all parameters are instantiated, $\mathcal{A}[\pi]$ is a standard timed automaton. (Strictly speaking, $\mathcal{A}[\pi]$ is only a timed automaton if $\pi$ assigns an integer to each parameter.)

Also recall that the composition of several PTAs (Network of Parametric Timed Automata, or NPTA) results in a PTA (see, e.g., [3]).

**Definition 3 (State).** *A* (symbolic) state *$s$ of $\mathcal{A}(K)$ is a couple $(q, C)$ where $q$ is a location, and $C$ a constraint on the clocks and the parameters.*

For each valuation $\pi$ of the parameters $P$, we may view a state $s$ as the set of pairs $(q, w)$ where $w$ is a clock valuation such that $w \models C[\pi]$. The *initial state* of $\mathcal{A}(K)$ is a state $s_0$ of the form $(q_0, C_0)$, where $C_0 = K \wedge I_{q_0} \wedge \bigwedge_{i=1}^{H-1} x_i = x_{i+1}$. $K$ is the initial constraint, $I_{q_0}$ is the invariant of the initial state, and the rest of the expression lets clocks evolve from the same initial value.

The symbolic semantics of a PTA is given in the following. Given a constant $d \in \mathbb{R}_{\geq 0}$, we use $X+d$ to denote the set $\{x_1+d, \ldots, x_H+d\}$. Given a constraint $C$, we rename the set of variables $X = \{x_1, \ldots, x_H\}$ as $X' = \{x'_1, \ldots, x'_H\}$. We use the notation $C(X)$ (resp. $C(X')$) to indicate that $X$ (resp. $X'$) is the set of clocks occurring in $C$. We use $X' = \rho(X)$, where $X'$ is a renaming of $X$, to denote the conjunction of equalities $x'_i = 0$ for all $x_i \in \rho$, and $x'_i = x_i$ otherwise. Given a state $s = (q, C)$, a step of the automaton from $s$ is defined below:

- $(q, C) \overset{a}{\to} (q', C')$ if $(q, g, a, \rho, q') \in \to$, and $C'$ is a constraint on the clocks and parameters defined, using the set of (renamed) clocks $X'$, by:
  $C'(X') = (\exists X : \ (C(X) \wedge g(X) \wedge X' = \rho(X) \wedge I_{q'}(X')))$.
- $(q, C) \overset{d}{\to} (q, C')$, 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 $(q, C) \overset{a}{\to} (q', C'')$ and $(q', C'') \overset{d}{\to} (q', C')$, i.e., $C'$ is a constraint on the clocks and the parameters obtained by removing $X$ and $d$ from the following expression:
  $C'(X') = (\exists X, d : \ (C(X) \wedge g(X) \wedge X' = \rho(X) + d \wedge I_{q'}(X' - d) \wedge I_{q'}(X')))$.
  It can be shown that $C'$ can be put under the form of a constraint on the clocks and the parameters.

**Definition 4 (Run).** *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 $\mathcal{A}(K)$.*

**Definition 5 (Trace associated to a run).** *Given a PTA $\mathcal{A}$ and a run $R$ of $\mathcal{A}$ of the form $(q_0, C_0) \overset{a_0}{\Rightarrow} \cdots \overset{a_{m-1}}{\Rightarrow} (q_m, C_m)$, the* trace associated to $R$ *is the alternating sequence of locations and actions $q_0 \overset{a_0}{\Rightarrow} \cdots \overset{a_{m-1}}{\Rightarrow} q_m$.*

The *trace set of $\mathcal{A}$* refers to the set of traces associated to the runs of $\mathcal{A}$.

In the following, we are interested in verifying properties on the trace set of $\mathcal{A}$. For example, given a predefined set of "bad locations", a reachability property is satisfied by a trace if this trace never contains a bad location; such a trace is "good" w.r.t. this reachability property. A trace can also be said to be "good" if a given action always occurs before another one within the trace (see example in Section 3). Actually, the good behaviors that can be captured with trace sets are relevant to *linear-time properties* [7], which can express properties more general than reachability properties.

Formally, given a property on traces, we say that a trace is *good* if it satisfies the property, and *bad* otherwise. Likewise, we say that a trace set is *good* if all its traces are good, and bad otherwise.

## 3 The Good Parameters Problem

We consider an example of asynchronous "D flip-flop" circuit described in [11] and depicted in Figure 1 left. It is composed of 4 gates ($G_1$, $G_2$, $G_3$ and $G_4$) interconnected in a cyclic way, and an environment involving two input signals $D$ and $CK$. The global output signal is $Q$. Each gate $G_i$ has a delay in the parametric interval $[\delta_i^-, \delta_i^+]$, with $\delta_i^- \leq \delta_i^+$. There are 4 other parameters (viz., $T_{HI}, T_{LO}, T_{setup}$, and $T_{Hold}$) used to model the environment. The output signal

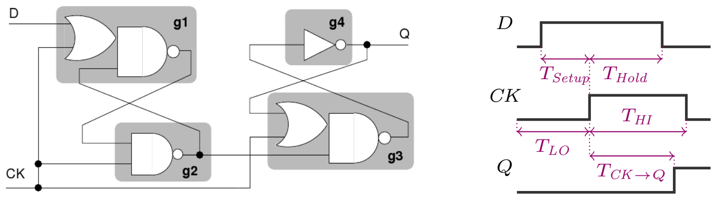

**Fig. 1.** Flip-flop circuit (left) and its environment (right)

of a gate $G_i$ is named $g_i$ (note that $g_4 = Q$). The rising (resp. falling) edge of signal $D$ is denoted by $D^{\uparrow}$ (resp. $D^{\downarrow}$) and similarly for signals $CK, Q, g_1, \ldots, g_4$.

We consider an environment starting from $D = CK = Q = 0$ and $g_1 = g_2 = g_3 = 1$, with the following ordered sequence of actions for inputs $D$ and $CK$: $D^{\uparrow}$, $CK^{\uparrow}$, $D^{\downarrow}$, $CK^{\downarrow}$, as depicted in Figure 1 right. Therefore, we have the implicit constraint $T_{setup} \leq T_{LO} \wedge T_{Hold} \leq T_{HI}$. Each gate is modeled by a PTA, as well as the environment. We consider a bi-bounded inertial model for gates (see [8, 23]), where any change of the input may lead to a change of the output (after some delay). The PTA $\mathcal{A}$ modeling the system results from the composition of those 5 PTAs. The initial location $q_0$ corresponds to the initial levels of the signals according to the environment. The initial constraint $C_0$ (regardless of the equality between the clock variables, see Section 2) is:

$$T_{setup} \leq T_{LO} \wedge T_{Hold} \leq T_{HI} \wedge \bigwedge_{i=1,..,4} \delta_i^- \leq \delta_i^+$$

We consider that the circuit has a *good behavior* if it verifies the following property $Prop_1$: "every trace contains both $Q^{\uparrow}$ and $CK^{\downarrow}$, and $Q^{\uparrow}$ occurs before $CK^{\downarrow}$". We are now interested in identifying parameter valuations for which the system has such a good behavior.

More generally, the *good parameters problem* can be stated as follows [14]:

Given a PTA $\mathcal{A}$ and a rectangular real-valued parameter domain $V_0$, what is the largest set of parameters values within $V_0$ for which $\mathcal{A}$ behaves well?

## 4 The Behavioral Cartography Algorithm

### 4.1 The Inverse Method

We first recall the inverse method algorithm, as defined in [3]. Given a PTA $\mathcal{A}$ and a valuation $\pi$ of the parameters, the inverse method $IM(\mathcal{A}, \pi)$ generates a constraint $K$ on the parameters, such that:

1. $\pi \models K$, and
2. For all $\pi_1, \pi_2 \models K$, the trace sets of $\mathcal{A}[\pi_1]$ and $\mathcal{A}[\pi_2]$ are equal.

We informally describe the algorithm $IM$ in the following. Starting with $K = True$, we iteratively compute a growing set of reachable states. When a $\pi$-*incompatible* state $(q, C)$ is encountered (i.e., when $\pi \not\models C$), $K$ is refined as follows: a $\pi$-incompatible inequality $J$ (i.e., such that $\pi \not\models J$) 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 no new state is computed. We finally return the intersection of the projection onto the parameters of all the constraints associated to the reachable states.

A more detailed version of the inverse method is given in Algorithm 1. Given a linear inequality $J$ of the form $e < e'$ (resp. $e \leq e'$), the expression $\neg J$ denotes the negation of $J$ and corresponds to the linear inequality $e' \leq e$ (resp. $e' < e$). Given a constraint $C$ on the clocks and the parameters, the expression $\exists X : C$ denotes the constraint on the parameters obtained from $C$ after elimination of the clocks, i.e., $\{\pi \mid \pi \models C\}$. We define $Post^i_{\mathcal{A}(K)}(S)$ as the set of states reachable from $S$ 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., $Post^*_{\mathcal{A}(K)}(S) = \bigcup_{i \geq 0} Post^i_{\mathcal{A}(K)}(S)$). Given two sets of states $S$ and $S'$, we write $S \sqsubseteq S'$ iff $\forall s \in S, \exists s' \in S'$ s.t. $s = s'$.

**Algorithm 1:** *IM*$(\mathcal{A}, \pi)$

```
   input : A PTA $\mathcal{A}$ of initial state $s_0 = (q_0, C_0)$
   input : Valuation $\pi$ of the parameters
   output: Constraint $K$ on the parameters
 1 $i \leftarrow 0$;  $K \leftarrow True$;  $S \leftarrow \{s_0\}$
 2 while $True$ do
 3     while there are $\pi$-incompatible states in $S$ do
 4         Select a $\pi$-incompatible state $(q, C)$ of $S$ (i.e., s.t. $\pi \not\models C$) ;
 5         Select a $\pi$-incompatible $J$ in $(\exists X : C)$ (i.e., s.t. $\pi \not\models J$) ;
 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 return $K \leftarrow \bigcap_{(q,C) \in S} (\exists X : C)$
 9     $i \leftarrow i + 1$ ;
10     $S \leftarrow S \cup Post_{\mathcal{A}(K)}(S)$ ;          // $S = \bigcup_{j=0}^{i} Post^j_{\mathcal{A}(K)}(\{s_0\})$
```

The termination of *IM* is not guaranteed in general. However, we provide in [3] sufficient condition for termination; in particular, *IM* is guaranteed to terminate for a form of acyclic automata.

The output $K$ of *IM* is a *behavioral tile* in the following sense: A constraint $K$ is said to be a *behavioral tile* (or more simply a *tile*), if for all $\pi_1, \pi_2 \in K$, the trace sets of $\mathcal{A}[\pi_1]$ and $\mathcal{A}[\pi_2]$ are equal. Note that a tile corresponds to a convex and dense set of real-valued points. Given a tile $K$, the trace set of $\mathcal{A}(K)$ will be simply referred to as "the trace set of $K$". Note that such a trace set is a (possibly infinite) set of finite traces.

Given a tile $K$ and a trace property *Prop*, we say that $K$ is *good* if its trace set is good. From the inverse method [3], in order to decide whether $K$ is good or bad, it is sufficient to select any $\pi \models K$ and check the truth of *Prop* for $\mathcal{A}[\pi]$.

## 4.2 The Behavioral Cartography Algorithm

*Principle.* By iterating the above inverse method *IM* over all the *integer* points of a rectangle[1] $V_0$ (of which there are a finite number), one is able to decompose (most of) the parametric space included into $V_0$ into behavioral tiles. Formally:

**Algorithm 2:** Behavioral Cartography Algorithm $BC(\mathcal{A}, V_0)$

```
    input : A PTA $\mathcal{A}$, a finite rectangle $V_0 \subseteq \mathbb{R}^M_{\geq 0}$
    output: $Tiling$: list of tiles (initially empty)
1   repeat
2       select an integer point $\pi \in V_0$;
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 points of $V_0$;
```

Note that two tiles with distinct trace sets are necessarily disjoint. On the other hand, two tiles with the same trace sets may overlap.

In many cases, all the real-valued space of $V_0$ is covered by *Tiling* (see Section 6). Besides, the space covered by *Tiling* often largely exceeds the limits of $V_0$ (see Section 4.4 for a sufficient condition of full coverage of the parametric space).

*Partition Between Good and Bad Tiles.* If now a decidable trace property is given then one can check which tiles are good (i.e., the tiles whose trace set satisfies the property), and which ones are bad. One can thus partition the rectangle $V_0$ into a good (resp. bad) subspace, i.e., a union of good (resp. bad) tiles.

*Advantages.* First, the cartography itself does not depend on the property one wants to check. Only the partition between good and bad tiles involves the considered property. Moreover, the algorithm is interesting because one does not need to compute the set of all the reachable states. On the contrary, each call to the inverse method algorithm quickly reduces the state space by removing the incompatible states. This allows us to overcome the state space explosion problem, which prevents other methods, such as the computation of the whole set of reachable states (and then the intersection with the bad states) [15], to terminate in practice. Finally note that the algorithm could easily be parallelized, e.g., by performing different calls to the inverse method in parallel, which is not possible in general when computing the set of all reachable states.

## 4.3 Application to the Flip-Flop Example

We are interested in studying the correctness of the flip-flop described in Section 3. For the sake of simplicity, we consider a model with only 2 parameters,

[1] Actually, $V_0$ can be a convex set containing a finite number of integer points.

with the following $V_0$: $\delta_3^+ \in [8, 30]$ and $\delta_4^+ \in [3, 30]$. The other parameters are instantiated as follows:

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

We compute the cartography of the flip-flop circuit according to $\delta_3^+$ and $\delta_4^+$, depicted in Figure 2. The dashed rectangle corresponds to $V_0$.

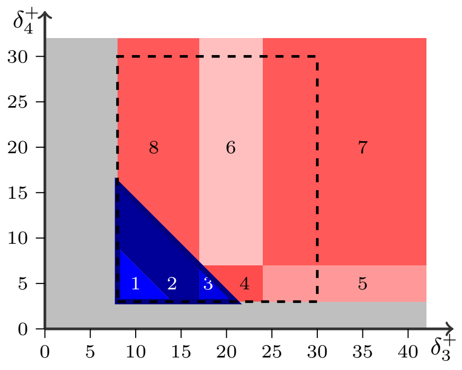

**Fig. 2.** Behavioral cartography of the flip-flop according to $\delta_3^+$ and $\delta_4^+$

First note that the whole (real-valued) $V_0$ is covered. Note also that tiles 5 to 8 are unbounded. Actually, this cartography covers the whole[2] real-valued parametric space $\mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0}$. According to the nature of the trace sets, we can easily partition the tiles into good and bad tiles w.r.t. property $Prop_1$ (see Section 3).

For example, the trace set of tile 3 (corresponding to the constraint $\delta_3^+ + \delta_4^+ < 24 \wedge \delta_3^+ \geq 17 \wedge \delta_4^+ \geq 3$) is given in Figure 3. This tile is a *good* tile because $Q^\uparrow$ occurs before $CK^\downarrow$ for all traces.

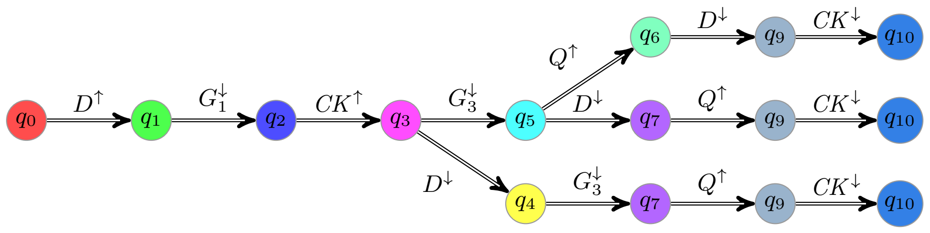

**Fig. 3.** Trace set of tile 3 for the flip-flop case study

[2] Apart from the irrelevant zone originating from the model ($\delta_3^+ < 8$ or $\delta_4^+ < 3$).

Likewise, the trace set of tile 7 (corresponding to the constraint $\delta_3^+ \geq 24 \wedge \delta_4^+ \geq 7$) is given in Figure 4. This is a *bad* tile because there exist traces where $Q^\uparrow$ occurs after $CK^\downarrow$.

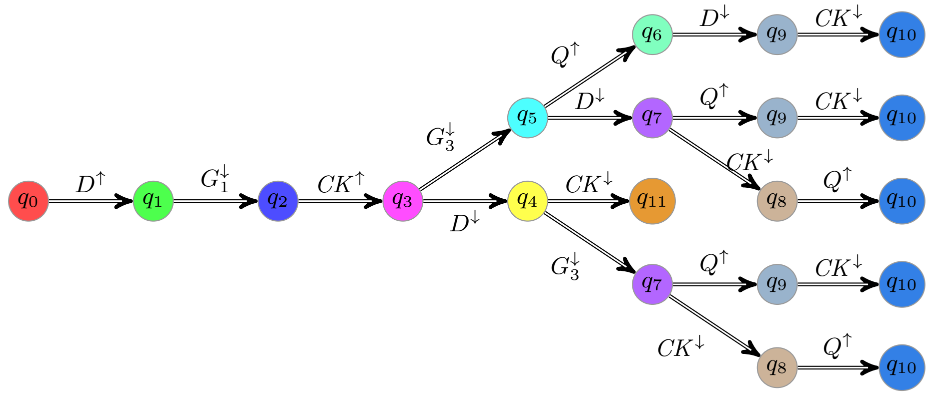

**Fig. 4.** Trace set of tile 7 for the flip-flop case study

One sees more generally that tiles 1 to 3 are good while tiles 4 to 8 are bad. From this partition into good and bad tiles, we infer the following constraint:

$$\delta_3^+ + \delta_4^+ \leq 24 \quad \wedge \quad \delta_3^+ \geq 8 \quad \wedge \quad \delta_4^+ \geq 3$$

which gives the *maximal* set of good parameters, thus solving the good parameters problem for this example.

*Comparison with other methods.* By computing in a brute manner the whole set of reachable states for all possible valuations of the parameters, and performing the intersection with the set of bad locations, we get the same constraint ensuring the good behavior of the system. Note that this comparison is possible because this example is rather simple; for bigger examples, such a computation would be impossible because of the state space explosion problem (see the Root Contention Protocol in Section 5.3). In [11], a constraint guaranteeing a good behavior is given. The projection of this constraint onto $\delta_3^+$ and $\delta_4^+$ gives $\delta_3^+ < 11 \wedge \delta_3^+ + \delta_4^+ < 18 \wedge \delta_3^+ \geq 8 \wedge \delta_4^+ \geq 3$, which is strictly included in our constraint[3].

## 4.4 A Sufficient Condition for Full Coverage

In this section, we show that for "acyclic" automata, a variant of the cartography algorithm allows us to cover the whole real-valued space of parameters within $V_0$.

[3] Actually, the comparison is not completely fair, because the two models are slightly different.

The graphical representation of a PTA $\mathcal{A}$ is an oriented graph where vertices correspond to locations, and edges correspond to actions of $\mathcal{A}$. We say that a PTA is *graphically acyclic* (or, more simply, *acyclic*) if its graph is acyclic.

**Lemma 1 (Termination).** *Given an acyclic PTA $\mathcal{A}$ and a rectangle $V_0$, the algorithm $BC(\mathcal{A}, V_0)$ always terminates.*

*Proof.* Based on the termination of the inverse method (see Proposition 23 in [3]) and the finite number of integer points in $V_0$.

Note that the acyclicity of the PTA is a sufficient, but non-necessary, termination condition of $BC$. See Section 5.3 for an example of non acyclic PTA for which the cartography algorithm terminates.

The algorithm $BC$ guarantees to cover the *integer* points within $V_0$. 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*. In order to fill these holes, one can refine the algorithm in a simple way. This variant, say $BC'$, is obtained from $BC$ by repeatedly generating at the end of $BC$ new tiles of the form $IM(\mathcal{A}, \pi)$, where $\pi$ is a *rational* (instead of integer) point selected within the holes. In the case of acyclic PTAs, the termination of $BC'$ is guaranteed. This is due to the *finiteness* of the number of different tiles which can be output by $IM(\mathcal{A}, \pi)$, for any *rational* point $\pi$ of $V_0$. Formally:

**Proposition 1.** *Let $\mathcal{A}$ be an acyclic PTA. The set of tiles $\{IM(\mathcal{A}, \pi) \mid \pi \in V_0 \cap \mathbb{Q}_{\geq 0}\}$ is finite.*

Moreover, one can show that $BC$ covers the whole parametric space beyond $V_0$, for a "sufficiently large" $V_0$. Formally:

**Proposition 2.** *Let $\mathcal{A}$ be an acyclic PTA. Then there exists a rectangle $V_0$ such that $BC(\mathcal{A}, V_0)$ covers the whole real-valued parametric space outside $V_0$.*

## 5 Application to the Probabilistic Framework

### 5.1 Extending the Inverse Method to Probabilistic Systems

Probabilistic Timed Automata are an extension of Timed Automata to the probabilistic case [19]. Parametric Probabilistic Timed Automata (PPTAs) are an extension of those Probabilistic Timed Automata to the parametric case [4]. In this framework, the discrete actions are *distributions* of actions. Roughly speaking, instead of going from a location to another location, one goes from a location to a distributions of locations. A *scheduler* is a mapping which associates to every state *one* output distribution. For each scheduler $\sigma$, one can define a probability space for a given probabilistic timed automaton $\mathcal{A}[\pi]$. In particular, one can define the *probability of reaching a given location* for $\mathcal{A}[\pi]$ under a given $\sigma$. Such probabilities can be computed using the Prism model-checker [16].

Given a PPTA $\mathcal{A}$, one considers the non-probabilistic version $\mathcal{A}^*$ of $\mathcal{A}$ [4]: this is done roughly speaking by replacing each distribution of actions by a set of standard non-deterministic actions. We have shown in [4] that the minimum (resp. maximum) probability *prob* of reaching a given location in $\mathcal{A}[\pi]$ is uniquely determined by the trace set of $\mathcal{A}^*[\pi]$. Hence, in order to determine *prob* for $\mathcal{A}[\pi]$, it is sufficient to proceed as follows:

1. Compute $K = IM(\mathcal{A}^*, \pi)$;
2. Compute *prob* (using, e.g., PRISM) for $\mathcal{A}[\pi']$, for some $\pi' \in K$.

One advantage of this method is that one can take $\pi'$ small enough in order to make the computation of PRISM easier, because the performance of PRISM depends on the size of the state space of the model used as input, which in turn depends on the size of the constants used in the probabilistic timed automata.

### 5.2 Extending the Cartography to the Probabilistic Framework

Using the cartography described in Section 4 and the result of [4], we can construct a cartography of a probabilistic system. We get a set of tiles such that, for any point in a given tile, the minimum (resp. maximum) probability of reaching a given location is the same. Formally, given a PPTA $\mathcal{A}$, a rectangle $V_0$ and a reachability property *rp*:

1. Compute $\mathit{Tiling} = BC(\mathcal{A}^*, V_0)$;
2. For each tile $K \in \mathit{Tiling}$, select $\pi \models K$, and compute the minimum (resp. maximum) probability of satisfying *rp* in $\mathcal{A}[\pi]$ (using, e.g., PRISM).

Note that, if one wants to consider another reachability property $rp'$, one can keep *Tiling* as computed in step 1, and only needs to redo step 2.

This cartography method is useful for finding appropriate timing parameters, e.g., in randomized protocols. To our knowledge, no other method allows the synthesis of constraints on the parameters within which the values of reachability probabilities are preserved.

### 5.3 Example: Root Contention Protocol

This case study concerns the Root Contention Protocol of the IEEE 1394 ("FireWire") High Performance Serial Bus, considered in the parametric framework in [20]. We consider the following valuation $\pi_0$ of the parameters given in [20]: $\mathit{rc\_fast\_min} = 76$, $\mathit{rc\_fast\_max} = 85$, $\mathit{rc\_slow\_min} = 159$, $\mathit{rc\_slow\_max} = 167$, and $\mathit{delay} = 30$. We are interested in computing the minimum probability $\mathit{prob}_1$ of satisfying the following property $\mathit{rp}_1$: "a leader is elected after three rounds or less". Using PRISM, it is possible to determine that $\mathit{prob}_1 = 0.75$ for $\pi_0$. To study this probability for other points around $\pi_0$, we compute a cartography with the following $V_0$: $\mathit{rc\_slow\_min} \in [140, 200]$, $\mathit{rc\_slow\_max} \in [140, 200]$ and $\mathit{delay} \in [1, 50]$. The two other parameters remain constant, as in $\pi_0$.

The cartography is given in Figure 5. For the sake of clarity, we project onto *delay* and *rc_slow_min*. In each tile, the parameter *rc_slow_max* is only bound by the implicit constraint $\mathit{rc\_slow\_min} \leq \mathit{rc\_slow\_max}$.

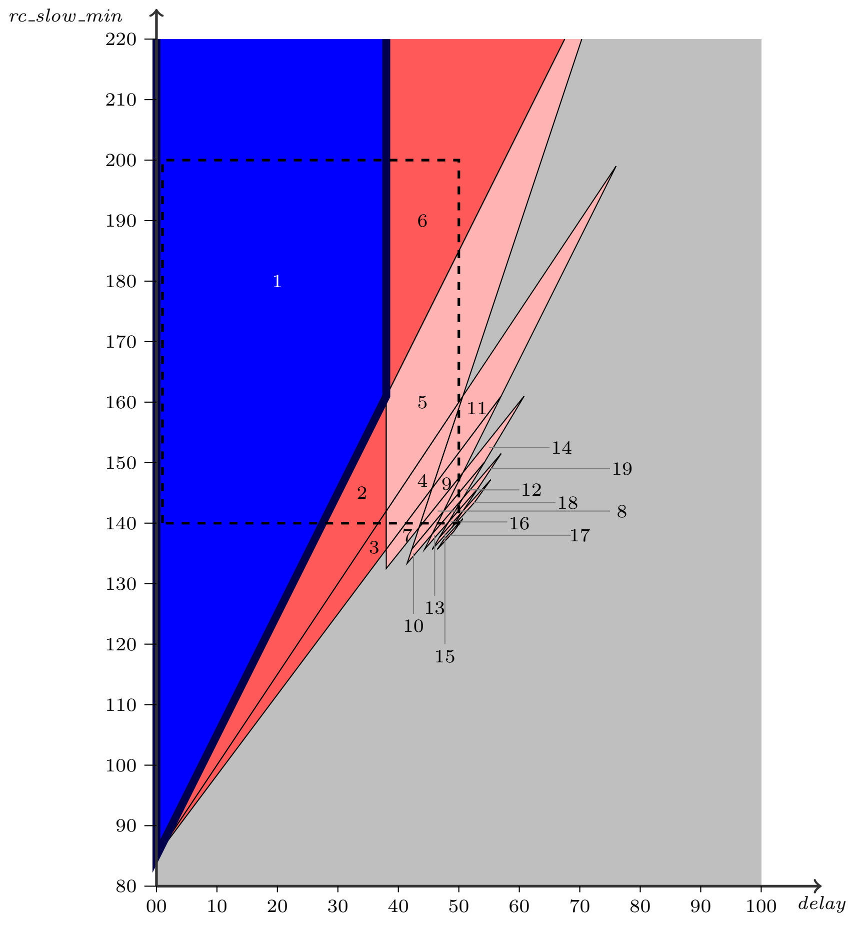

**Fig. 5.** Behavioral cartography of the Root Contention Protocol according to *delay* and *rc_slow_min*

Note that tiles 1 and 6 are infinite towards dimension *rc_slow_min*, and all tiles are infinite towards dimension *rc_slow_max*. Moreover, although all the integer points within $V_0$ are covered (from the algorithm), note that the real-valued part of $V_0$ is not fully covered, because there are some "holes" (real-valued zones without integer points) in the lower right corner. It would not be possible to fill completely those holes, using the refinement of the algorithm *BC* given in Section 4.4, because Proposition 1 does not hold any longer here. Note that, nevertheless, our method is still capable of giving valuable information by partitioning most of the parametric space within $V_0$ into good and bad tiles. Finally note that the computation of the whole set of reachable states (see, e.g., [15]) does not terminate in this example, due to the infinite number of generated traces (with incomparable constraints).

*Partition into good and bad subspaces.* Applying PRISM to one point of each tile, we find $prob_1 = 0.75$ for tile 1. For tiles 2, 3 and 6, we have $prob_1 = 0.625$. For the other tiles, $prob_1 = 0.5$. Let us suppose that a tile is good when the probability $prob_1$ is greater than 0.7, and bad otherwise. In this case, only tile 1 is a good tile, and the others are bad tiles.

An advantage of the cartography algorithm is that, if one considers another property than $rp_1$, there is no need to re-compute the cartography again. Various other properties have been considered (e.g., the election of a leader after *five* rounds or less), leading to different partitions into good and bad subspaces.

## 6 Case Studies

An implementation of the behavioral cartography algorithm has been made, called IMITATOR II. This program is a complete new version, written in OCaml, of the prototype IMITATOR [5]. The execution of IMITATOR II is fully automated, from the source file to the generation of the behavioral tiles and the corresponding trace sets under a graphical form. IMITATOR II makes use of the library APRON for the manipulation of constraints [18].

Results are presented in the table below. The input rectangle $V_0$ in each case study was chosen for containing the reference valuation $\pi_0$ of the model, corresponding to a reference behavior (see, e.g., [3]). We give from left to right the name of the example, the number of PTAs composing the global system $\mathcal{A}$, the lower and upper bounds on the number of locations per PTA, the number of clocks, of non-instantiated parameters, of integer points within $V_0$, of tiles computed, the average number per tile of states and transitions of the trace set, and the computation time in seconds.

| Example | PTAs | loc./PTA | $\|X\|$ | $\|P\|$ | $\|V_0\|$ | tiles | states | trans. | Time |
|---|---|---|---|---|---|---|---|---|---|
| SR-latch | 3 | $[3,8]$ | 3 | 3 | 1331 | 6 | 5 | 4 | 0.3 |
| Flip-flop [11] | 5 | $[4,16]$ | 5 | 2 | 644 | 8 | 15 | 14 | 3 |
| SPSMALL [9] | 10 | $[3,8]$ | 10 | 2 | 3149 | 259 | 60 | 61 | 1194 |
| CSMA/CD [21] | 3 | $[3,8]$ | 3 | 3 | 2000 | 140 | 349 | 545 | 269 |
| RCP [20] | 5 | $[6,11]$ | 6 | 3 | 186050 | 19 | 5688 | 9312 | 7018 |

Note that the version of the algorithm that we used in IMITATOR II is the classical algorithm (viz., $BC$, and not $BC'$). Also note that only the SR-latch case study is modeled with an acyclic PTA (see Section 4.4).

For all those examples, the cartography covers 100 % of the real-valued space of $V_0$, except for the Root Contention Protocol, where "only" 99,99 % of $V_0$ is covered (see Section 5.3). Moreover, a significant part of the real-valued space outside $V_0$ is also covered. Those examples, as well as other case studies, can be found on IMITATOR II's Web page[4].

Finally note that it is possible to find examples (such as the "And–Or" circuit considered in [10, 3]) for which the algorithm $BC$ does not terminate for some $V_0$, because the algorithm $IM$ does not terminate for some $\pi \in V_0$.

[4] `http://www.lsv.ens-cachan.fr/~andre/IMITATOR2/`

## 7 Final Remarks

In this paper, we presented a cartography algorithm, which covers most of the parametric space with *tiles*, for which the behavior is uniform. This gives a new approach for solving the *good parameters problem*. Our algorithm has been successfully applied to various examples of asynchronous circuits and protocols. Our cartography algorithm often covers the whole real-valued space of $V_0$ as well as a significant part of the space beyond $V_0$.

This method extends naturally to probabilistic systems. This allows us to decompose the parametric space into tiles which are uniform w.r.t. probabilistic reachability properties. The tiles generated by the cartography are always the same, whatever the considered probabilistic property is. Only the partition into good and bad subspaces changes.

Our approach has the following limitation: the equivalence relation on parameters that leads to "tiles" as equivalence classes is strong (because of the equality of trace sets). This may lead to a big (even infinite) number of small equivalence classes (as shown in Section 5.3). It would be interesting to consider a more general inverse method in order to weaken the equivalence relation.

As suggested in Section 4.4, it is interesting to consider variants of $BC$ with a strategy of dynamic point selection for $IM$: instead of starting from the set of all integer points of $V_0$, one starts from a sparse subset of points, and fill incrementally the uncovered zones by selecting (non-necessarily integer) points in the "holes".

Finally, it would be interesting to extend the method to hybrid systems, where clocks evolve at different rates.

*Acknowledgment.* We thank the anonymous referees for their helpful comments.

## References

1. R. Alur and D. L. Dill. A theory of timed automata. *TCS*, 126(2):183–235, 1994.
2. R. Alur, T.A. Henzinger, and M. Y. Vardi. Parametric real-time reasoning. In *STOC '93*, pages 592–601. ACM, 1993.
3. 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.
4. 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.
5. Étienne André. IMITATOR: A tool for synthesizing constraints on timing bounds of timed automata. In *ICTAC'09*, volume 5684 of *LNCS*, pages 336–342. Springer, 2009.
6. 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.
7. C. Baier and J.-P. Katoen. *Principles of Model Checking*. The MIT Press, 2008.
8. J. A. Brzozowski and C. J. Seger. *Asynchronous Circuits*. Springer-Verlag, 1995.

9. R. Chevallier, E. Encrenaz, L. Fribourg, and W. Xu. Timed verification of the generic architecture of a memory circuit using parametric timed automata. *Formal Methods in System Design*, 34(1):59–81, 2009.
10. R. Clarisó and J. Cortadella. Verification of concurrent systems with parametric delays using octahedra. In *ACSD '05*. IEEE Computer Society, 2005.
11. R. Clarisó and J. Cortadella. The octahedron abstract domain. *Sci. Comput. Program.*, 64(1):115–139, 2007.
12. 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.
13. A. Collomb–Annichini and M. Sighireanu. Parameterized reachability analysis of the IEEE 1394 Root Contention Protocol using TReX. In *RT-TOOLS '01*, 2001.
14. 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.
15. T.A. Henzinger and H. Wong-Toi. Using HyTech to synthesize control parameters for a steam boiler. In *Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control, LNCS 1165*. Springer-Verlag, 1996.
16. A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. PRISM: A tool for automatic verification of probabilistic systems. In *TACAS'06*, volume 3920 of *LNCS*, pages 441–444. Springer, 2006.
17. Gerard Holzmann. *Spin model checker, the: primer and reference manual.* Addison-Wesley Professional, 2003.
18. B. Jeannet and A. Miné. Apron: A library of numerical abstract domains for static analysis. In *CAV '09*, volume 5643 of *LNCS*, pages 661–667. Springer, 2009.
19. M. Kwiatkowska, G. Norman, R. Segala, and J. Sproston. Automatic verification of real-time systems with discrete probability distributions. *TCS*, 282:101–150, 2002.
20. M. Kwiatkowska, G. Norman, and J. Sproston. Probabilistic model checking of deadline properties in the IEEE 1394 FireWire root contention protocol. *Formal Aspects of Computing*, 14(3):295–318, 2003.
21. M. Kwiatkowska, G. Norman, J. Sproston, and F. Wang. Symbolic model checking for probabilistic timed automata. *Information and Computation*, 205(7):1027–1077, 2007.
22. 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.
23. O. Maler and A. Pnueli. Timing analysis of asynchronous circuits using timed automata. In *CHARME '95*, pages 189–205. Springer-Verlag, 1995.
24. 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.