An Inverse Method for Parametric Timed Automata

Étienne André, Thomas Chatain, Laurent Fribourg[1]

LSV – ENS de Cachan & CNRS, France

Emmanuelle Encrenaz[2]

LIP6 – Université Pierre et Marie Curie & CNRS, France

Abstract

Given a timed automaton with parametric timings, our objective is to describe a procedure for deriving constraints on the parametric timings in order to ensure that, for each value of parameters satisfying these constraints, the behaviors of the timed automata are time-abstract equivalent. We will exploit a reference valuation of the parameters that is supposed to capture a characteristic proper behavior of the system. The method has been implemented and is illustrated on various examples of asynchronous circuits.

Keywords: Timed automata, parameters, verification, asynchronous circuits.

1 Introduction

This work aims at extending the inverse method described in the restricted framework of "Time Separation of Events" [11]. The problem of time separation can be stated as follows: given a system made of several connected components, each one entailing a local delay known with uncertainty, what is the maximum time for traversing the global system? This problem is useful, e.g. in the domain of digital circuits, for determining the global traversal time of a signal from the knowledge of bounds on the component propagation delays. The inverse problem is the following: we seek intervals for component delays for which the global traversal time is guaranteed to be no greater than a specified maximum. The system is represented in [11] under the form of a *timing constraint graph*, which is an oriented graph whose vertices represent events and directed edges causal dependency between them. We consider here systems modeled by timed automata. The model of timed automata

[⋆] This work is partially supported by the French ANR project VALMEM and FARMAN project SIMOP.

[1] Email: {andre,chatain,fribourg}@lsv.ens-cachan.fr

[2] Email: emmanuelle.encrenaz@lip6.fr

is more general than timing constraint graphs as it allows for composition of systems and choice of actions. The timing bounds involved in the action guards and location invariants of our timed automata are not constants, but *parameters*. Those *parametric timed automata* allow to model various kinds of timed systems, e.g. communication protocols or asynchronous circuits.

We will also assume that we are given an initial set of values for the parameters that form the so-called "reference parameter valuation", which corresponds to values for which the system is known to behave properly. Our goal is to compute a constraint K on the parameters, satisfied by the reference valuation, guaranteeing that, under any valuation satisfying K, the system behaves in the same manner: for any two valuations of the parameters satisfying K, the behaviors of the timed automata are *(time-abstract) equivalent*, i.e., the traces of execution (or runs) viewed as alternating sequences of actions and locations are identical. Our procedure consists in generating growing paths starting from the initial state. When one generates a path ending in a state incompatible with the reference values, the path is discarded by refining appropriately the current constraint K on parameters. The generation procedure is then restarted until a new incompatible state is produced, and so on, iteratively until no incompatible state is generated.

Comparison with Related Work.

The synthesis of constraints has been studied in the context of parametric timed automata or hybrid systems, e.g. in [4], or in [13] where the authors use a prototype extension of Uppaal [14] for linear parametric model checking. Note that [4] is able to infer *non*-linear constraints.

Another interesting related work on parametric timed automata is the 2nd part of [13], which shows decidability results for the verification of a special class, called "L/U automata". This class is somehow restricted: for example, it does not allow for systems with guards of the form $x = p$ (i.e., $x \leq p \wedge p \leq x$), where x is a clock and p a parameter, because parameters must appear either as lower bounds or as upper bounds of clocks, but not both. Furthermore, the way for synthesizing constraints is indirect: one needs to guess a constraint, then check that an appropriate instance of the system is correct under this constraint, from which the general correctness is inferred by an equivalence theoretical result. Two subclasses of L/U automata, called lower-bound and upper-bound parametric timed automata, are also considered in [17], with decidability results.

As pointed out in [12], a major strength of tool HyTech is its ability to perform parametric analysis. One can synthesize constraints on parameters for which a given "bad" state is reachable (see, e.g., Fisher's mutual exclusion protocol in [12]). This is done by computing the set $Post^*(s_{init})$ of reachable states, intersecting with the bad states, and eliminating the non-parameter variables.

The synthesis of constraints has also been studied in the context of asynchronous circuits, mainly by Myers and co-workers on the one hand (see, e.g., [18]), and by Clariso and Cortadella on the other hand (see, e.g., [6,7]). They also proceed by analyzing failure traces and generating timing constraints that prevent the occurrence of such failures. A basic difference between the two works is that Myers *et al*'s approach is not fully parametric, but appeals to a numeric ILP (Integer Linear

Programming) procedure, which tightens the delay bounds according to the generated timing constraints. Myers *et al*'s approach is somehow reminiscent to the technique of timing verification by successive approximation, used by [3] and [15] in the framework of timed automata. In contrast, Clariso-Cortadella's approach, as ours, is fully parametric and handles *linear constraints* on parameters (rather than numeric intervals). Clariso-Cortadella computes the effect of loop iteration, using extrapolation techniques such as "widening" borrowed to the domain of Abstract Interpretation [10], and focuses on a restricted class of linear constraints (called "octahedra"). Our procedure differs on these points, since it does not iterate loops, and handles general linear constraints.

This idea of synthesizing constraints by refinement from counter-examples has often been used in the literature, e.g. using TReX [9], and more generally with CEGAR-based methods (counter-example guided abstraction refinement [8]). Note that, in our work, the notion of bad state corresponds to the notion of state incompatible with the reference valuation.

Overview of the paper

We first introduce Parametric Timed Automata (Sect. 2) and a motivating example (Sect. 3). Then, we present our method of synthesis of constraints (Sect. 4), and apply it to the motivating example, as well as to an example in the case of cyclic traces (Sect. 5). We finally give some remarks and directions for future work (Sect. 6).

2 Parametric Timed Automata

We assume familiarity with standard timed automata [1]. All clock constraints of standard timed automata are boolean combinations of atomic conditions that compare values with nonnegative integer *constants*. Parametric timed automata allow within clock constraints the use of *parameters* in place of constants (see [2]).

2.1 Parameters and Constraints

Throughout this paper, we assume a fixed set of *parameters* $P = \{p_1, \ldots, p_{2R}\}$. We assume that this set is partitioned into a set $P^l = \{p^l_1, \ldots, p^l_R\}$ of "lower bounds" and a set $P^u = \{p^u_1, \ldots, p^u_R\}$ of "upper bounds", satisfying implicitly the set of constraints $0 \leq p^l_i \leq p^u_i$ (for $i = 1, \ldots, R$)[3]. Besides, we assume given a subset $P^=$ of P made of parameters p^l and p^u, satisfying the constraint $p^l = p^u$. In the examples (see Sect. 3), such parameters will be denoted by p without superscript.

A *parameter valuation* π is a function $\pi : P \rightarrow \mathbb{R}^{\geq 0}$ assigning a non-negative real value to each parameter. There is a one-to-one correspondence between valuations and points in $(\mathbb{R}^{\geq 0})^{2R}$. We will often identify a valuation π with the point $(\pi(p_1), \ldots, \pi(p_{2R}))$.

A *clock* is a variable x_i with value in $\mathbb{R}_{\geq 0}$. The set of clocks will be denoted by $X = \{x_1, \ldots, x_H\}$. Given a constant $d \in \mathbb{R}_{\geq 0}$, we use $X + d$ to denote the

[3] This definition does not restrict our result, and is only introduced in order to treat the examples in an intuitive way.

set $\{x_1 + d, \ldots, x_H + d\}$. Similarly to the parameter valuation, we define a *clock valuation* as a function $w : X \rightarrow \mathbb{R}^{\geq 0}$ assigning a non-negative real value to each clock. We will often identify a valuation w with the point $(w(x_1), \ldots, w(x_H))$. We will use the notation $\lambda x.0$ for the clock valuation assigning value 0 to each clock.

Definition 2.1 An (X, P)*-atom* is a linear inequality of the form: $\gamma x + \Sigma_i \alpha_i p^l_i \prec \delta y + \Sigma_j \beta_j p^u_j$ where $\prec \in \{<, \leq\}$, $\gamma, \delta \in \{0, 1\}$, $\alpha_i, \beta_j \in \mathbb{N}$, $p^l_i \in P^l, p^u_j \in P^u$, and $x, y \in X$. An (X, P)*-constraint* is a conjunction of (X, P)-atoms.

If C is an (X, P)-constraint and π a parameter valuation, then $C[\pi]$ denotes the X-constraint (constraint on 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 $w(x)$. A clock valuation w *satisfies* X-constraint $C[\pi]$, denoted $w \models C[\pi]$, if $C[\pi][w]$ evaluates to true. The *semantics* of X-constraint $C[\pi]$, denoted $[\![C[\pi]]\!]$ is the set of clock valuations that satisfy $C[\pi]$. We say that $C[\pi]$ is *satisfiable* if $[\![C[\pi]]\!]$ is nonempty.

We say that a parameter valuation π *satisfies* (X, P)-constraint C, denoted $\pi \models C$, if $C[\pi]$ is satisfiable. The semantics of an (X, P)-constraint C, denoted $[\![C]\!]$, is the set of parameter valuations that satisfy C. We say that C is *satisfiable* if $[\![C]\!]$ is nonempty.

Remarks

We will use the notation $<w, \pi> \models C$ to indicate that $C[\pi][w]$ evaluates to true.

Given an (X, P)-constraint C, it is sometimes convenient to 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 clock variables occurring in C.

Sets of parameter valuations will be represented themselves under a constraint form.

Definition 2.2 A P*-atom* is a linear inequality of the form

$$\Sigma_{i \in L} \alpha_i p^l_i \prec \Sigma_{j \in U} \beta_j p^u_j$$

where L, U are subsets of $\{1, \ldots, R\}$, $\prec \in \{<, \leq\}$, $\alpha_i, \beta_j \in \mathbb{N}$, $p^l_i \in P^l$ and $p^u_j \in P^u$.

Let J be a P-atom of the form $\Sigma_{i \in L} \alpha_i p^l_i \leq \Sigma_{j \in U} \beta_j p^u_j$ (resp. $\Sigma_{i \in L} \alpha_i p^l_i < \Sigma_{j \in U} \beta_j p^u_j$). The *negated form of* J, denoted by $\neg J$, is the P-atom defined by:

$$\Sigma_{i \in L} \alpha_i p^l_i > \Sigma_{j \in U} \beta_j p^u_j \text{ (resp. } \Sigma_{i \in L} \alpha_i p^l_i \geq \Sigma_{j \in U} \beta_j p^u_j).$$

A P*-constraint* is a conjunction of P-atoms. Similarly to the semantics of (X, P)-constraints, we say that a parameter valuation π *satisfies* P-constraint K, denoted $\pi \models K$, if $K[\pi]$ is satisfiable, which means that the expression obtained by replacing each parameter p in K with $\pi(p)$ evaluates to true.

We will consider the special atom *True* as a P-atom, corresponding to the set of all possible values for P.

2.2 Definition of Parametric Timed Automata

The following definition is an extension of the class of timed automata (considered in e.g. [16]) to the parametric case. With respect to the classical definition, this class is contrived by the fact that guards and invariants are necessarily in *conjunctive* form, but this is not restrictive in practice.

Definition 2.3 A *parametric timed automaton (PTA)* is a 7-tuple of the form $\mathcal{A}(K) = (\Sigma, Q, q_{init}, X, P, I, \rightarrow)$, where:

- Σ is a finite set of actions (or "step labels"),
- Q is a finite set of locations (or "control states"),
- q_{init} is the initial location,
- X is a finite set of clocks,
- P is a finite set of parameters partitioned as $P = P^l \biguplus P^u$,
- K is a P-constraint on the set of parameters P,
- I is the invariant, assigning to every $q \in Q$ a conjunction $I_q(X)$ of (X, P)-atoms of the form $x \leq p^u$, for some clock variable $x \in X$ and parameter $p^u \in P^u$, and
- $\rightarrow$ is a step (or "transition") relation consisting of elements of the form (q, g, a, ρ, q'), also denoted $q \overset{g,a,\rho}{\rightarrow} q'$, where $q, q' \in Q$, $a \in \Sigma$, $\rho \subseteq X$ is a set of clock variables to be reset by the step, and $g(X)$ (the step guard) is a conjunction of (X, P)-atoms of the form $p^l \prec x$ with $\prec \in \{\leq, <\}$ for some clock variable $x \in X$ and parameter $p^l \in P^l$.

For every parameter valuation $\pi = (\pi_1, \dots, \pi_{2R})$, we denote $\mathcal{A}[\pi]$ the (standard) timed automaton $\mathcal{A}(\bigwedge_{i=1,\dots,2R}(p_i = \pi_i))$, i.e. the automaton obtained by substituting every occurrence of a parameter p_i by the integer π_i in the guards and invariants of $\mathcal{A}$.

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 $x'_i = x_i$ for all the other variables x_i of X.

2.3 Concrete Semantics

2.3.1 Concrete States

Given a PTA $\mathcal{A}$ and a valuation π of its set of parameters, a *concrete state* s of $\mathcal{A}[\pi]$ is a pair (q, w) where q is a location and w a valuation of the clock variables.

Definition 2.4 A *labeled transition system (LTS)* over a set of symbols Σ is a triple $\mathcal{L} = (S, S_0, \rightarrow)$, with S a set of *states*, $S_0 \subseteq S$ a set of *initial states*, and $\rightarrow \in S \times \Sigma \times S$ a *transition relation*. We write $s \overset{a}{\rightarrow} s'$ for $(s, a, s') \in \rightarrow$. A *trace*, or *run*, of $\mathcal{L}$ is a finite alternating sequence of states $s_i \in S$ and symbols $a_i \in \Sigma$ 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 $s_0 \in S_0$.

Definition 2.5 Let $\mathcal{A}[\pi] = (\Sigma, Q, q_{init}, X, P, I, \rightarrow)$ be a PTA, where π is a valuation of P. The *concrete semantics of $\mathcal{A}$ under π* is the labeled transition system

$(S, S_0, \Rightarrow)$ over $(\Sigma \cup \mathbb{R}^{\geq 0})$ where

$$S = \{(q, w) \in Q \times (X \to \mathbb{R}^{\geq 0}) \mid <w, \pi> \models I(q)\},$$
$$S_0 = \{(q, w) \in S \mid q = q_{init} \wedge w = \lambda x.0\}$$

and the transition predicate $\Rightarrow$ is specified by the following three rules. For all $(q, w), (q', w') \in S, d \geq 0$ and $a \in \Sigma$,

- $(q, w) \xrightarrow{d} (q', w')$ if $q = q'$ and $w' = w + d$;
- $(q, w) \xrightarrow{a} (q', w')$ if $\exists g, \rho : q \overset{g,a,\rho}{\to} q'$ and $<w, \pi> \models g$ and $w' = \rho(w)$.
- $(q, w) \overset{a}{\Rightarrow} (q', w')$ if $\exists d, q'', w'' : (q, w) \xrightarrow{d} (q'', w'') \xrightarrow{a} (q', w')$.

Note that this LTS has at most one initial state. It has no initial state if the invariant assigned to the initial location q_{init} of $\mathcal{A}$ is unsatisfiable.

2.3.2 Concrete Traces

Definition 2.6 Let $\mathcal{A}$ be a PTA on a set of parameters P, let s_{init} be its initial state, let π be a valuation of P. A *concrete trace*, or *concrete run*, T of $\mathcal{A}[\pi]$ (of length $m - 1$) is a finite alternating sequence of concrete states and actions of the form:

$s_1 \overset{a_1}{\Rightarrow} s_2 \overset{a_2}{\Rightarrow} \cdots s_i \ldots \overset{a_{m-1}}{\Rightarrow} s_m$

such that $s_1 = s_{init}$ and, for all $i = 1, \ldots, m - 1$, $a_i \in \Sigma$ and $s_i \overset{a_i}{\Rightarrow} s_{i+1}$ is a step in the concrete semantics.

The *last state* of a trace of the form $s_1 \overset{a_1}{\Rightarrow} \ldots \overset{a_{m-1}}{\Rightarrow} s_m$ is s_m.

2.4 Symbolic Semantics

2.4.1 Symbolic States

A *symbolic state* s is a pair (q, C) where q is a location and C an (X, P)-constraint. For each valuation π of the parameters P, we may view a symbolic state s as the set of pairs (q, w) where w is a clock valuation such that there exists a parameter valuation π such that $<w, \pi> \models C$.

The *P-constraint associated to a state* $s = (q, C)$ is the constraint D obtained by eliminating the clock variables (i.e. variables of X) in C, for example via Fourier-Motzkin. We have: $D \Leftrightarrow (\exists X : C)$ [4].

Proposition 2.7 *Let C be an (X, P)-constraint on a set of parameters P and a set of clocks X, let D be the P-constraint associated to C, let π be a valuation of P. Then:*

$$\pi \models C \Leftrightarrow \pi \models D$$

Proof. (sketch) From the definition of the P-constraint associated to a state, and the semantics of a P-constraint and an (X, P)-constraint. □

[4] Unlike [13], we introduce no canonical form for D.

Let X_{init} be a subset of X made of clocks initially instantiated to 0 (the other being only constrained to have non-negative initial values). The *initial* (X, P)-*constraint*, denoted by C_{init}, is of the form:

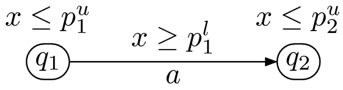
$X_{init} = 0 \wedge \bigwedge_{i=1,...,R} p_i^l \leq p_i^u \wedge \bigwedge_{i \in I^=} p_i^l = p_i^u \wedge K$,

where $I^=$ is the subset of $\{1, ..., R\}$ corresponding to indices of the elements of $P^=$. The initial state, denoted by s_{init}, is (q_{init}, C_{init}).

The semantics of a parametric timed automaton is given in terms of global steps as follows. Given a state $s = (q, C)$, a step (or transition) of the automaton from s is one of the following:

- A *discrete step*: $(q, C) \xrightarrow{a} (q', C')$, which means that, for some step $(q, g, a, \rho, q') \in \rightarrow$, C' is an (X, P)-constraint defined, using the set of (renamed) clock variables X', by:
  $C'(X') = \{\exists X \ \ [C(X) \wedge I_q(X) \wedge g(X) \wedge X' = \rho(X) \wedge I_{q'}(X')]\}$.
- A *time step*: $(q, C) \xrightarrow{d} (q, C')$ where d is a new parameter with values in $\mathbb{R}_{\geq 0}$, which means that C' is given by:
  $C'(X') = \{\exists X \ \ [C(X) \wedge X' = X + d \wedge I_q(X')]\}$.
- A *compound step*: $s \overset{d,a}{\Rightarrow} s'$, which is a time step followed by a discrete step, i.e: $s \xrightarrow{d} s'' \wedge s'' \xrightarrow{a} s'$ for some state s''. In other words: $(q, C) \overset{d,a}{\Rightarrow} (q', C')$, means that, for some step $(q, g, a, \rho, q') \in \rightarrow$, C' is an (X, P)-constraint defined by:
  $C'(X') = \{\exists X \ \ [C(X) \wedge I_q(X + d) \wedge g(X + d) \wedge X' = \rho(X + d) \wedge I_{q'}(X')]\}$.
- A *(general) step*: $s \overset{a}{\Rightarrow} s'$, which means $\exists d \in \mathbb{R}_{\geq 0} : s \overset{d,a}{\Rightarrow} s'$. In other words: $(q, C) \overset{a}{\Rightarrow} (q', C')$, means that, for some step $(q, g, a, \rho, q') \in \rightarrow$, C' is an (X, P)-constraint defined by:
  $C'(X') = \{\exists X, d \ \ [C(X) \wedge I_q(X + d) \wedge g(X + d) \wedge X' = \rho(X + d) \wedge I_{q'}(X')]\}$.[5]

Example. Consider the following PTA, with 2 states and one clock ($X = \{x\}$):



Starting from q_1, we consider a time step and a discrete a-transition. Thus, we have $(q_1, C_1(X)) \xrightarrow{d} (q_1, C_1'(X)) \xrightarrow{a} (q_2, C_2(X))$. We have $C_1(X) = (x \leq p_1^u)$. From the definition of the semantics of a time step, we have $C_1'(X) = (x \leq p_1^u \wedge x + d \leq p_1^u)$. And from the definition of the semantics of a general step, we have:
$C_2(X) = \exists d : (x \leq p_1^u \wedge x + d \leq p_1^u \wedge x + d \geq p_1^l \wedge x + d \leq p_2^u)$
and after elimination of d: $C_2(X) = x \leq p_1^u \wedge p_1^l \leq p_1^u \wedge p_1^l \leq p_2^u$, which gives us in particular the new P-constraint $p_1^l \leq p_2^u$.

Remark. The presence of $P^=$ and the associated equalities will allow us to express guards of the form $x \geq p$ on transitions exiting from locations having $x \leq p$ as invariants, where p $(= p^l = p^u)$ belongs to $P^=$. This feature is not expressible with L/U automata (see [13]), where parameters can be used as either lower bounds of clocks or upper bounds, but not both. We thus consider here a superclass of L/U automata.

[5] It can be shown that $C'(X')$ can be put under the form of an (X, P)-constraint.

Given a set of states S, the *posterior set of states* of S, denoted by $Post_{\mathcal{A}(K)}(S)$, is the set of states reachable in at most one general step from a state of S, i.e.:

$Post_{\mathcal{A}(K)}(S) = S \cup \{s' \mid s \overset{a}{\Rightarrow} s',$ for some $s \in S$ and $a \in \Sigma\}$.
As usual, one defines $Post^i_{\mathcal{A}(K)}(S)$ as the set of states reachable from S in at most i steps, and $Post^*_{\mathcal{A}(K)}(S) = \bigcup_{i \geq 0} Post^i_{\mathcal{A}(K)}(S)$. Note that, if $Post^{i+1}_{\mathcal{A}(K)}(S) = Post^i_{\mathcal{A}(K)}(S)$ for some i, then $Post^*_{\mathcal{A}(K)}(S) = Post^i_{\mathcal{A}(K)}(S)$. In the sequel, we will be interested in computing the set $Post^*_{\mathcal{A}(K)}(s_{init})$ where K is a given P-constraint.

2.4.2 Symbolic Traces

Definition 2.8 Given a PTA $\mathcal{A}(K)$ of initial state s_{init}, a *symbolic trace* T of $\mathcal{A}(K)$ (of length $m-1$) is a finite alternating sequence of symbolic states and actions of the form:

$s_1 \overset{a_1}{\Rightarrow} s_2 \overset{a_2}{\Rightarrow} \cdots s_i \ldots \overset{a_{m-1}}{\Rightarrow} s_m$
such that $s_1 = s_{init}$ and, for all $i = 1, \ldots, m-1$, $a_i \in \Sigma$ and $s_i \overset{a_i}{\Rightarrow} s_{i+1}$ is a step.

We say that a symbolic trace is *acyclic* if it never passes twice by the same location.

We say that a symbolic trace is *trivially cyclic* if, when entering for the second time in a previously visited location, the constraint associated to the second visit is stronger than the first time. More formally:

Definition 2.9 Let $\mathcal{A}(K)$ be a PTA. A symbolic trace T of $\mathcal{A}(K)$ of the form $(q_1, C_1) \overset{a_1}{\Rightarrow} \cdots \overset{a_{m-1}}{\Rightarrow} (q_m, C_m)$ is *trivially cyclic* iff:

$\exists i < m : q_i = q_m \wedge C_m \subseteq C_i$ and $\forall j < m : i \neq j \Rightarrow q_i \neq q_j$

By extension, any symbolic trace containing as a prefix a trivially cyclic trace is also considered as a trivially cyclic trace.

The *significant length*, or simply *length*, of a trivially cyclic trace T is defined as the length of the smallest prefix of T being a trivially cyclic trace.

We now define both notions of symbolic trace simulated by a concrete trace, and concrete trace simulated by a symbolic trace.

Definition 2.10 Let $\mathcal{A}$ be a PTA on a set of parameters P, let K be a P-constraint on P, let π be a valuation of P.

A symbolic trace of $\mathcal{A}(K)$ of the form $(q_1, C_1) \overset{a_1}{\Rightarrow} \cdots \overset{a_{m-1}}{\Rightarrow} (q_m, C_m)$ is *simulated by a concrete trace* of $\mathcal{A}[\pi]$ of the form $(q_1, w_1) \overset{a_1}{\Rightarrow} \cdots \overset{a_{m-1}}{\Rightarrow} (q_m, w_m)$, where $C_1, \ldots, C_m$ are (X, P)-constraints and $w_1, \ldots, w_m$ are clock valuations, iff

$\forall i, 1 \leq i \leq m : w_i \models C_i[\pi]$.

Definition 2.11 Let $\mathcal{A}$ be a PTA on a set of parameters P, let K be a P-constraint on P, let π be a valuation of P.

A concrete trace of $\mathcal{A}[\pi]$ of the form $(q_1, w_1) \overset{a_1}{\Rightarrow} \cdots \overset{a_{m-1}}{\Rightarrow} (q_m, w_m)$ is *simulated by a symbolic trace* of $\mathcal{A}(K)$ of the form $(q_1, C_1) \overset{a_1}{\Rightarrow} \cdots \overset{a_{m-1}}{\Rightarrow} (q_m, C_m)$, where $C_1, \ldots, C_m$ are (X, P)-constraints and $w_1, \ldots, w_m$ are clock valuations, iff

$\forall i, 1 \leq i \leq m : w_i \models C_i[\pi]$.

Thus, we can define the notion of equivalence between set of traces.

Definition 2.12 Let $\mathcal{A}$ be a PTA on a set of parameters P, let K be a P-constraint on P, let π be a valuation of P.

We say that a set S of symbolic traces of $\mathcal{A}(K)$ and a set S' of concrete traces of $\mathcal{A}[\pi]$ are *equivalent* iff:

- any trace in S is simulated by a trace in S';
- any trace in S' is simulated by a trace in S.

2.5 Networks of Parametric Timed Automata

This presentation is adapted from [16] to the parametric framework. For each automaton $\mathcal{A}_i(K_i)$, where K_i is a constraint on a local set of parameters $P_i = P_i^l \uplus P_i^u$, let Σ_i be its local alphabet, i.e., the set of step labels it uses.

Definition 2.13 Let $P = \biguplus_{i=1}^{N} P_i$ be a set of parameters where P_i are mutually disjoint sets of local parameters, let $K = \bigwedge_{i=1,\ldots,N} K_i$ be a constraint on P. A *network of parametric timed automata (NPTA)* is $\mathcal{A}(K) = \mathcal{A}_1(K_1)\| \cdots \|\mathcal{A}_N(K_N)$, where $\|$ is the standard operator for parallel composition, and each automaton is of the form $\mathcal{A}_i = (\Sigma_i, Q_i, q_{init_i}, X_i, P_i, I_i, \rightarrow_i)$. The sets of locations and clocks are mutually disjoint.

The global automaton obtained from the network of parametric timed automata is $\mathcal{A}(K) = (\Sigma, Q, q_{init}, X, P, I, \rightarrow)$, where $Q = \Pi_{i=1}^{N} Q_i$, $q_{init} = (q_{init_1}, \ldots, q_{init_N})$, $X = \biguplus_{i=1}^{N} X_i$, $P = \biguplus_{i=1}^{N} P_i$ and $\Sigma = \bigcup_{i=1}^{N} \Sigma_i$. We write global locations as $\overline{q} = (q_1, \ldots, q_N) \in Q$ and global (X, P)-constraints as $C = C_1 \wedge \ldots \wedge C_N$.

The symbolic semantics of the NPTA is given in terms of compound and general steps as follows:

- A compound step: $(\overline{q}, C) \overset{d,a}{\Rightarrow} (\overline{q}', C')$ is such that, for every i either $a \in \Sigma_i$ and $(q_i, C_i) \overset{d,a}{\Rightarrow} (q_i', C_i')$ is a step of $\mathcal{A}_i$, or $a \notin \Sigma_i$ and $(q_i, C_i) \xrightarrow{d} (q_i', C_i')$ is a step of $\mathcal{A}_i$, and $C' = C_1' \wedge \ldots \wedge C_N'$.
- A (general) step: $(\overline{q}, C) \overset{a}{\Rightarrow} (\overline{q}', C')$ means that $(\overline{q}, C) \overset{d,a}{\Rightarrow} (\overline{q}', C')$ for some $d \in \mathbb{R}_{\geq 0}$.

The concrete semantics is similarly defined in a natural way.

3 An Illustrating Example

Consider an asynchronous "D flip-flop" circuit described in [6] and depicted on Fig. 1. It is composed of 4 gates (g_1, g_2, g_3 and g_4) which are interconnected in a cyclic way, and an environment which corresponds to 2 input signals D and CK.

Each gate g_i has a symbolic delay in the interval $[p_{g_i}^l, p_{g_i}^u]$. Each gate is modeled by a timed automaton, plus a timed automaton modeling the environment. There are 12 timing parameters, among which 4 belong to $P^=$ (viz., $T_{HI}, T_{LO}, T_{Setup}$, and T_{Hold}). We set the following parameters as follows:

$$\begin{array}{llll}
T_{HI} = 20 & T_{LO} = 15 & T_{Setup} = 10 & T_{Hold} = 15 \\
p_{g_1}^u = 1 & p_{g_1}^l = 1 & p_{g_2}^u = 6 & p_{g_2}^l = 5 \\
p_{g_3}^u = 10 & p_{g_3}^l = 8 & p_{g_4}^u = 5 & p_{g_4}^l = 3
\end{array}$$

For these values, the set of traces of execution of the system is depicted under

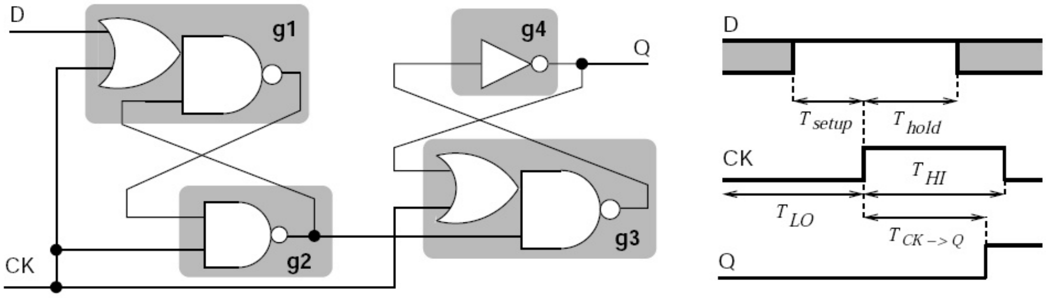


Fig. 1. Flip-flop circuit

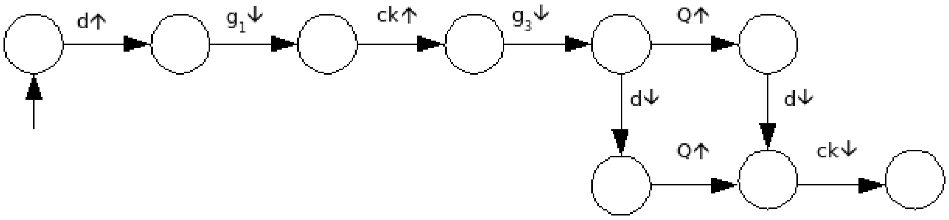


Fig. 2. Reachability graph of the flip-flop circuit

the compact form of a (time-abstract) reachability graph, given in Fig. 2. Note that this example is non deterministic, since we have several possible traces for the same set of values.

We are interested in finding a constraint K on the parameters, such that, for any valuation of P satisfying K, the reachability graph will be the same. This amounts to say:

> Consider a PTA $\mathcal{A}$ with a valuation π of the parameters. Find a constraint K with $\pi \models K$, such that the set of traces of $\mathcal{A}(K)$ and the set of traces of $\mathcal{A}[\pi]$ are equivalent.

The simple idea of the procedure that we are going to describe is to start with $K = True$, enumerate all the states and remove the states incompatible with π by iteratively refining K.

4 Synthesis of P-Constraints

We first define the notion of π-compatible state.

Definition 4.1 A symbolic state $s = (q, C)$ is said to be π-*compatible* iff $\pi \models D$, where D is the P-constraint associated to s.

A state is said to be π-incompatible if it is not π-compatible.

In [13], Prop. 3.17 states that each symbolic trace is simulated by a concrete trace. The formalism in [13] uses PDBM, whereas we use a first-order form of (X, P)-constraints. However, this has no incidence on the result used here. The following proposition is a reformulation of Prop. 3.17 in our formalism.

Proposition 4.2 *Let $\mathcal{A}$ be a PTA on a set of parameters P, let π be a valuation of*

P, let K be a P-constraint. For each parameter valuation π and clock valuation w, if there is a symbolic trace of $\mathcal{A}(K)$ reaching state (q, C), with $<w, \pi> \models C$, then this trace is simulated by a concrete trace of $\mathcal{A}[\pi]$ reaching state (q, w).

Proof. Cf. [13]. □

Conversely, we state that, given $\pi \models K$, each concrete trace of $\mathcal{A}[\pi]$ is simulated by a symbolic trace of $\mathcal{A}(K)$.

Proposition 4.3 *Let $\mathcal{A}$ be a PTA on a set of parameters P, let π be a valuation of P and K be a P-constraint such that $\pi \models K$. For each clock valuation w, if there is a concrete trace of $\mathcal{A}[\pi]$ reaching state (q, w), then this trace is simulated by a symbolic trace of $\mathcal{A}(K)$ reaching a state (q, C), such that $<w, \pi> \models C$.*

Proof. By adapting the proof of Prop. 3.18 of [13] to take into account the fact that $\pi \models K$. □

4.1 Algorithm SYNTHESIS

Prop. 4.3 gives us a hint for the synthesis procedure: in order to find a generalization of π, for which the set of traces mimics the set of π, it suffices to prevent the generation of states traces which are π-incompatible.

Given an NPTA $\mathcal{A}(K)$, an initial state s_{init} and a reference valuation π of the parameters, we are going to describe a procedure finding by refinement a constraint K on the parameters that preclude the generation of any π-incompatible state. The idea of the procedure for synthesizing K is as follows, starting from $K := \mathit{True}$:

(i) Generate $Post^i_{\mathcal{A}(K)}(s_{init})$ for $i = 1, 2, \dots$ until a π-incompatible state is generated;

(ii) strengthen K in order to prevent the generation of this state;

(iii) go to (i).

Step (i) can incorporate a fixpoint test for the set of generated states, and step (ii) a satisfiability test for K with the reference valuation. Note that, if the P-constraint associated to a state does not satisfy π, then it necessarily contains an atom J such that $\pi \models K \wedge \neg J$ (because any P-constraint is stronger than K). Note also that $\pi \models K$ is an invariant.

We present the procedure *SYNTHESIS* on Fig. 3, where K is the current P-constraint, and $\mathcal{S}$ the current set of generated states.

The pair $(K, \mathcal{S})$ can be seen as a *successful answer* given by the procedure, where K is satisfied by π, and $\mathcal{S}$ is a fixpoint of *Post* (i.e.: $\mathcal{S} = Post^*_{\mathcal{A}(K)}(s_{init})$). This answer is produced nondeterministically for each selection of π-incompatible state, and each selection of atom J.

4.2 Correction

We now state the correction of the algorithm.

Theorem 4.4 *Let $\mathcal{A}$ be a PTA on a set of parameters P, let s_{init} be its initial state, let π be a valuation of the parameters P, let K be the result of SYNTHESIS*

```
Input: $(\mathcal{A}, s_{init}, \pi)$
Output: $(K, \mathcal{S})$
Variables:
    $i$: current step
    $\mathcal{S}_{prev}$: set of reachable states at previous step

Algorithm SYNTHESIS:
Initially:  $K := True$;  $\mathcal{S} := \{s_{init}\}$;  $i := 1$.
DO
    $\mathcal{S}_{prev} := \mathcal{S}$
    $\mathcal{S} := Post^{i}_{\mathcal{A}(K)}(s_{init})$
    if $\mathcal{S} = \mathcal{S}_{prev}$ then return $(K, \mathcal{S})$ fi
    DO until there is no $\pi$-incompatible state in $\mathcal{S}$ :
        Select:
          – a $\pi$-incompatible state $s$ in $\mathcal{S}$ ($\pi \not\models D$, where $D$ is the
                                    $P$-constraint associated to $s$)
          – an inequality $J$ in $D$ such that $\pi \models K \wedge \neg J$
        $K := K \wedge \neg J$
        $\mathcal{S} := Post^{i}_{\mathcal{A}(K)}(s_{init})$
    OD
    $i := i + 1$
OD
```

Fig. 3. Algorithm *SYNTHESIS*

applied to $(\mathcal{A}, s_{init}, \pi)$. *Then* $\pi \models K$, *and the set of traces of* $\mathcal{A}(K)$ *and the set of traces of* $\mathcal{A}[\pi]$ *are equivalent.*

Proof. By construction of K, we have $\pi \models K$.

Let us consider a symbolic trace T of $\mathcal{A}(K)$, whose last state is (q_m, C_m). By construction of K, all the symbolic states of T are π-compatible, which means $C_m[\pi]$ is satisfiable. Thus, there exists a clock valuation w such that $w \models C_m[\pi]$. Thus, we have a trace in the symbolic semantics reaching state (q_m, C_m), with $<w, \pi> \models C_m$. Which means, applying Prop. 4.2, that T is simulated by a trace of $\mathcal{A}[\pi]$.

Conversely, we prove that any concrete trace of $\mathcal{A}[\pi]$ is simulated by a symbolic trace of $\mathcal{A}(K)$ applying Prop. 4.3. □

It follows that, for any two valuations π_1 and π_2 of P satisfying K, the set of traces of $\mathcal{A}[\pi_1]$ and $\mathcal{A}[\pi_2]$ are equivalent.

4.3 Termination

Reachability analysis is known to be undecidable in the framework of PTA [2], and computations performed with tools dealing with PTA (such as HyTech [12]) might not terminate. However, we give sufficient condition for ensuring termination of our method.

The following lemma, used to prove termination, states that, for any i, the number of states of a PTA symbolically reachable in i iterations is finite. This, of course, does not mean that the number of all reachable states ($Post^*$) is finite.

Lemma 4.5 *Let $\mathcal{A}$ be a PTA on a set of parameters P, let s_{init} be its initial state, let K be a P-constraint on P, let i be a positive integer. Then, the number of states in $Post^i_{\mathcal{A}(K)}(s_{init})$ is finite.*

Proof. (sketch) Based on the fact that there is a finite number of branching at each iteration. □

4.3.1 Acyclic Case

Let us first assume that all symbolic traces of $\mathcal{A}[\pi]$ are acyclic.

Theorem 4.6 *Let $\mathcal{A}$ be a PTA on a set of parameters P, let π be a valuation of P. If all symbolic traces of $\mathcal{A}[\pi]$ are acyclic, then algorithm SYNTHESIS terminates in n iterations, where n is the length of the longest symbolic trace of $\mathcal{A}[\pi]$.*

Proof. Let us first consider the inner *DO* loop.
At each iteration, we select an inequality J in the P-constraint associated to a state s, we negate it and add it to K. Thus, s can not be reached anymore. Moreover, no new state can be reached, as K was strengthened with the addition of $\neg J$. As i remains constant in this loop, and as $Post^i_{\mathcal{A}(K)}(s_{init})$ contains a finite number of states (Lemma 4.5), then the number of states in $\mathcal{S}$ strictly decreases. Thus, the number of π-incompatible states strictly decreases and a finite number of iterations of the inner *DO* loop is performed.

Let us now consider the outer *DO* loop.

- Let us suppose the algorithm terminates when $i < n$. Then, as the instantiated traces are acyclic, we can find an instantiated trace which is strictly bigger than any parametric trace. Which is not possible because of the correction of *SYNTHESIS* (Theorem 4.4).
- Let us suppose we are still in the outer *DO* loop when $i = n + 1$. Then we reached new states which were not reachable at step n. Consider one state s of those new states. If s is π-compatible, then it should have been encountered in the instantiated trace, which is not possible because it has length n and is acyclic. If s is π-incompatible, then it will be cut by negation of an inequality in its associated P-constraint, and the set of states will finally be the same one as at iteration n, thus the fixpoint and the termination.

□

4.3.2 Cyclic Case

Let us now consider that the symbolic traces of $\mathcal{A}[\pi]$ may be trivially cyclic. We conjecture that our algorithm also terminates in the case where all symbolic traces of $\mathcal{A}[\pi]$ are either of finite length, or trivially cyclic. This conjecture was observed to be true on all the examples we treated.

Conjecture 4.7 *Let $\mathcal{A}$ be a PTA on a set of parameters P, let π be a valuation of P. If all symbolic traces of $\mathcal{A}[\pi]$ are either acyclic or trivially cyclic, then algorithm SYNTHESIS terminates in n iterations, where n is the longest length of a symbolic trace of $\mathcal{A}[\pi]$.*

An example of cyclic system is given in Sect. 5.2.

4.4 Complexity and Optimizations

An elementary complexity analysis can show that our procedure is exponential with the number of parameters P, exponential with the number of locations of $\mathcal{A}$, and doubly exponential with the number of clocks X. However, in practice, we have successfully applied *SYNTHESIS* to various examples containing up to 10 parameters, 10 clocks and several thousands of potentially reachable locations. One reason for which it behaves well in practice, apart from the optimizations (see below), is that the procedure quickly reduces the number of reachable states, by iteratively refining K.

Optimization

The first $Post^i$ (in the outer *DO* loop) can actually be replaced with a simple *Post* computation: $Post_{\mathcal{A}(K)}(\mathcal{S})$. Indeed, K did not change since the last computation and, as i was since incremented, the $Post^i$ newly computed is only the posterior set of states of the $\mathcal{S}$.

Another optimization is the test of equality $\mathcal{S} = \mathcal{S}_{prev}$. Its computation is theoretically exponential, but we use a method in negligible time to perform it. Indeed, in the case of our algorithm, it is sufficient in practice to consider the number of reachable states ordered by location in order to know whether we reached the fixpoint or not. Correction of this optimization can be verified by computing $Post^*_{\mathcal{A}(K)}$ at the end of the algorithm and checking that it is equal to $\mathcal{S}$.

5 Examples

The procedure *SYNTHESIS* described in Sect. 4, has been implemented under the form of a script program written in Python and calling HyTech [12] for computing the *Post* operator on the current set of states. The selection of the atom J to be negated is performed randomly. The experiments were conducted on a 3.20 GHz Intel Xeon with 2 GB memory. The two examples are asynchronous circuits modeled using the bi-bounded inertial delay (see [5,15]).

5.1 Flip-flop Example

Applied to the NPTA modeling the flip-flop circuit depicted with the associated set of parameter valuations given in Section 3, the program generates the following set K of inequalities $\neg J$ in 6 seconds:

(1) $T_{LO} > T_{Setup}$

(2) $T_{Setup} > p^u_{g_1}$

(3) $T_{Hold} > p^u_{g_3}$

(4) $T_{Hold} > p^u_{g_3} + p^u_{g_4}$

A minimal constraint K' is obtained from K by removing the 3rd inequality ($T_{Hold} > p^u_{g_3}$). The set of reachable states is exactly the same as the one depicted on Fig. 2. Surprisingly, we noticed that, whatever the algorithm randomly chooses

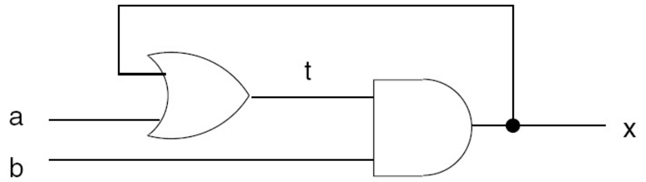


Fig. 4. And–Or Component

at each iteration for the atom $\neg J$, the constraint finally computed always remains equivalently the same.

Besides, by construction on the environment (signals D, CK and Q), we have the implicit additional constraint: $T_{HI} > T_{Hold}$. By comparison, the authors of [6] find (besides $T_{HI} > T_{Hold}$):

$$\begin{aligned} T_{LO} &> T_{Setup} \\ T_{HI} &> p^u_{g_3} + p^u_{g_4} + p^u_{g_2} \\ T_{Setup} + p^l_{g_2} &> p^u_{g_1} + p^u_{g_2} \\ T_{Hold} &> p^u_{g_2} + p^u_{g_3} \\ p^l_{g_1} &> p^u_{g_2} \end{aligned}$$

Note that the authors of [6] generated the previous constraint in order to prevent bad system behaviors. The bad state is defined as the case where $CK^{\downarrow}$ occurs before $Q^{\uparrow}$. The time-abstraction of the two sets of traces (modeled under the form of a reachability graph in Fig. 2) coincide. As we chose values for the parameters having the same good behavior as in [6], and thus avoiding the bad state, our reachability graph avoids the bad state. Therefore, our constraint also prevents bad system behaviors.

Our constraint strictly includes the one of [6], which can be easily checked with HyTech. In other terms, our constraint allows a strictly bigger set of behaviors (in terms of NPTA zone inclusion) than the one in [6].

5.2 An Example of Cyclic Trace

This example deals with an "And–Or" circuit described in [7] and depicted on Fig. 4. It is composed of 2 gates (one And-gate and one Or-gate) which are interconnected in a cyclic way, and the environment which corresponds to 2 input signals a and b, with cyclic alternating rising edges and falling edges. Each rising (resp. falling) edge of signal $a, b, \ldots$, is denoted by $a^{\uparrow}, \ldots$ (resp. $a^{\downarrow}, \ldots$) The delay between the rising and the falling edge of $a^{\uparrow}$ (resp. $a^{\downarrow}$) and $a^{\downarrow}$ (resp. $a^{\uparrow}$) is in $[p^l_{a+}, p^u_{a+}]$ (resp. $[p^l_{a-}, p^u_{a-}]$), and similarly for b. The traversal of the gate Or takes also a delay in $[p^l_{Or}, p^u_{Or}]$, and likewise for gate And. There are 12 timing parameters.

A bad state expresses that the rising edge of output signal x occurs before the rising edge of a within the same cycle. We set the parameters to the following values, ensuring that the bad state is not reachable:

$$\begin{array}{llll} p^u_{a+} = 20 & p^l_{a+} = 19 & p^u_{a-} = 18 & p^l_{a-} = 16 \\ p^u_{b+} = 8 & p^l_{b+} = 7 & p^u_{b-} = 21 & p^l_{b-} = 20 \\ p^u_{And} = 10 & p^l_{And} = 9 & p^u_{Or} = 5 & p^l_{Or} = 4 \end{array}$$

Using those values, we get the reachability graph on Fig. 5. This ensures that the bad state is not reached, i.e., the rising edges and falling edges of a, b, x alternate properly. Using our program, the following set K of inequalities $\neg J$ is computed in

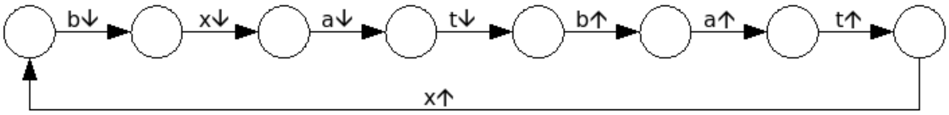


Fig. 5. Reachability graph for the instantiated And–Or component

11 seconds:

$$\begin{array}{lr}
(1) & p^l_{a+} > p^u_{And} + p^u_{b+} \\
(2) & p^l_{b+} + p^l_{b-} > p^u_{Or} + p^u_{a+} \\
(4) & p^l_{b+} > p^u_{Or} \\
(4) & p^l_{And} + p^l_{Or} > p^u_{b+}
\end{array}$$

The set of traces, expressed under the compact form of a reachability graph, is depicted on Fig. 5 and, by construction, does not reach any bad state. As in the case of the flip-flop example, we noticed that, whatever the algorithm randomly chooses at each iteration for the atom $\neg J$, the constraint finally computed always remains equivalently the same. In [7], the set of generated inequalities is not given.

6 Final Remarks

We presented an algorithm *SYNTHESIS* allowing to synthesize a constraint ensuring to get the same set of traces than the set of traces of a given valuation of the parameters. Our method suits particularly well in the framework of asynchronous circuits. It is in particular experimented at a larger scale in the framework of French ANR project VALMEM, for synthesizing timing constraints of memory circuits designed by ST-Microelectronics. Using the implementation of the algorithm *SYNTHESIS*, we successfully treated in less than 2 minutes the case of an asynchronous circuit containing several dozens of gates. One of the objectives of this project is to treat even larger circuits, containing several hundreds of gates.

We are currently working on proving Conjecture 4.7, allowing to state that our algorithm terminates, provided all symbolic traces of $\mathcal{A}[\pi]$ are either of finite length, or trivially cyclic. This conjecture was observed to be true on all the examples we treated.

Moreover, as noticed in the two examples, the random selection of the atom $\neg J$ in the algorithm always gives an (equivalently) identical final constraint. This phenomenon was also observed on much larger examples. It would be interesting to see under which condition this phenomenon holds.

We presented in this paper a method based on a reference valuation, leading to totally ordered traces. We are also interested in extending the method to the case of partially ordered traces.

References

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

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

[3] R. Alur, A. Itai, R. P. Kurshan, and M. Yannakakis. Timing verification by successive approximation. In *CAV '92*. Springer-Verlag, 1993.

[4] 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.

[5] J. A. Brzozowski and C. J. Seger. *Asynchronous Circuits*. Springer-Verlag, 1995.

[6] R. Clarisó and J. Cortadella. The octahedron abstract domain. In *SAS '04*, 2004.

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

[8] 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.

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

[10] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *POPL '78*, pages 84–96. ACM, 1978.

[11] E. Encrenaz and L. Fribourg. Time separation of events: An inverse method. In *Proceedings of the LIX Colloquium '06*, volume 209 of *ENTCS*, Palaiseau, France, 2008. Elsevier Science Publishers.

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

[13] T. Hune, J. Romijn, M. Stoelinga, and F. W. Vaandrager. Linear parametric model checking of timed automata. In *TACAS '01*, pages 189–203. Springer-Verlag, 2001.

[14] 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.

[15] O. Maler and A. Pnueli. Timing analysis of asynchronous circuits using timed automata. In *CHARME '95*, pages 189–205. Springer-Verlag, 1995.

[16] R. Ben Salah, M. Bozga, and O. Maler. On interleaving in timed automata. In *CONCUR '06*, volume 4137 of *LNCS*, pages 465–476. Springer, 2006.

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

[18] 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.