Parametric Deadlock-Freeness Checking Timed Automata*

Étienne André

[1] Université Paris 13, Sorbonne Paris Cité, LIPN, CNRS, UMR 7030, F-93430, Villetaneuse, France
[2] École Centrale de Nantes, IRCCyN, CNRS, UMR 6597, France

Abstract. Distributed real-time systems are notoriously difficult to design, and must be verified, *e. g.,* using model checking. In particular, deadlocks must be avoided as they either yield a system subject to potential blocking, or denote an ill-formed model. Timed automata are a powerful formalism to model and verify distributed systems with timing constraints. In this work, we investigate synthesis of timing constants in timed automata for which the model is guaranteed to be deadlock-free.

Keywords: parametric timed automata, deadlock freeness, timelock freeness

1 Introduction

Distributed real-time systems are notoriously difficult to design due to the intricated use of concurrency and timing constraints, and must therefore be verified, *e. g.,* using model checking. Model checking is a set of techniques to formally verify that a system, described by a model, verifies some property, described using formalisms such as reachability properties or more complex properties expressed using, *e. g.,* temporal logics.

Checking the absence of deadlocks in the model of a real-time system is of utmost importance. First, deadlocks can lead the actual system to a blockade when a component is not ready to receive any action (or synchronization label). Second, a specificity of models of distributed systems involving time is that they can be subject to situations where time cannot elapse. This situation denotes an ill-formed model, as this situation of time blocking ("timelock") cannot happen in the actual system due to the uncontrollable nature of time.

Timed automata (TAs) [AD94] are a formalism dedicated to modeling and verifying real-time systems where distributed components communicate via synchronized actions. Despite a certain success in verifying models of actual distributed systems (using *e. g.,* Uppaal [LPY97] or PAT [SLDP09]), TAs reach

* This is the author (and slightly extended) version of the manuscript of the same name published in the proceedings of the 13th International Colloquium on Theoretical Aspects of Computing (ICTAC 2016). The final version is available at https://link.springer.com/chapter/10.1007%2F978-3-319-46750-4_27. This work is partially supported by the ANR national research program PACS (ANR-14-CE28-0002).

some limits when verifying systems only partially specified (typically when the timing constants are not yet known) or when timing constants are known with a limited precision only (although the robust semantics can help tackling some problems, see *e. g.,* [Mar11]). Parametric timed automata (PTAs) [AHV93] leverage these drawbacks by allowing the use of timing parameters, hence allowing for modeling constants unknown or known with some imprecision.

We address here the problem of the deadlock-freeness, *i. e.,* the fact that a discrete transition must always be taken from any state, possibly after elapsing some time. TAs and PTAs are both subject to deadlocks: hence, a property proved correct on the model may not necessarily hold on the actual system if the model is subject to deadlocks. Deadlock checking can be performed on TAs (using *e. g.,* Uppaal); however, if deadlocks are found, then there is often no other choice than manually refining the model in order to remove them.

We recently showed that the existence of a parameter valuation in a PTA for which a run leads to a deadlock is undecidable [AL16]; this result also holds for the subclass of PTAs where parameters only appear as lower or upper bounds (L/U-PTAs [HRSV02]). This result rules out the possibility to perform exact deadlock-freeness synthesis.

In this work, we propose an approach to automatically synthesize parameter valuations in PTAs (in the form of a set of linear constraints) for which the system is deadlock-free. If our procedure terminates, the result is exact. Otherwise, when stopping after some bound (*e. g.,* runtime limit, exploration depth limit), it is an over-approximation of the actual result. In this latter case, we propose a second approach to also synthesize an under-approximation: hence, the designer is provided with a set of valuations that are deadlock-free, a set of valuations for which there exist deadlocks, and an intermediate set of unsure valuations. Our approach is of particular interest when intersected with a set of parameter valuations ensuring some property: one obtains therefore a set of parameter valuations for which that property is valid and the system is deadlock-free.

Outline We briefly recall necessary definitions in Section 2. We introduce our approach in Section 3, extend it to the synthesis of an under-approximated constraint in Section 4, and validate it on benchmarks in Section 5. We discuss future works in Section 6.

2 Preliminaries

2.1 Clocks, Parameters and Constraints

Let $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}_+$ and $\mathbb{R}_+$ denote the sets of non-negative integers, integers, non-negative rational numbers and non-negative real numbers respectively.

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

We assume a set $P = \{p_1, \ldots, p_M\}$ of *parameters*, *i. e.,* unknown constants. A parameter *valuation* v is a function $v : P \to \mathbb{Q}_+$. We identify a valuation v with the *point* $(v(p_1), \ldots, v(p_M))$. In the following, we assume $\bowtie \in \{<, \leq, \geq, >\}$. A *constraint* C (*i. e.,* a convex polyhedron) over $X \cup P$ is a conjunction of inequalities of the form $lt \bowtie 0$, where lt denotes a linear term over $X \cup P$ of the form $\sum_{1 \leq i \leq H} \alpha_i x_i + \sum_{1 \leq j \leq M} \beta_j p_j + d$, with $x_i \in X$, $p_i \in P$, and $\alpha_i, \beta_j, d \in \mathbb{Z}$. Given a parameter valuation v, $v(C)$ denotes the constraint over X obtained by replacing each parameter p in C with $v(p)$. Likewise, given a clock valuation w, $w(v(C))$ denotes the expression obtained by replacing each clock x in $v(C)$ with $w(x)$. We say that v *satisfies* C, denoted by $v \models C$, if the set of clock valuations satisfying $v(C)$ is nonempty. We say that C is *satisfiable* if $\exists w, v$ s.t. $w(v(C))$ evaluates to true. We define the *time elapsing* of C, denoted by $C^{\nearrow}$, as the constraint over X and P obtained from C by delaying all clocks by an arbitrary amount of time. We define the *past* of C, denoted by $C^{\swarrow}$, as the constraint over X and P obtained from C by letting time pass backward by an arbitrary amount of time (see *e. g.,* [JLR15]). 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. We denote by $C{\downarrow_P}$ the projection of C onto P, *i. e.,* obtained by eliminating the clock variables (*e. g.,* using Fourier-Motzkin [Sch86]).

A *guard* g is a constraint over $X \cup P$ defined by inequalities of the form $x \bowtie z$, where z is either a parameter or a constant in $\mathbb{Z}$.

A *parametric zone* is a polyhedron over $X \cup P$ in which all constraints on variables are of the form $x \bowtie plt$ (parametric rectangular constraints) or $x_i - x_j \bowtie plt$ (parametric diagonal constraints), where $x_i \in X$, $x_j \in X$ and plt is a parametric linear term over P, *i. e.,* a linear term without clocks ($\alpha_i = 0$ for all i). Given a parameter constraint K, $\neg K$ denotes the (possibly non-convex) negation of K. We extend the notation $v \models K$ to possibly non-convex constraints in a natural manner. $\top$ (resp. $\bot$) denotes the constraint corresponding to the set of all (resp. no) parameter valuations.

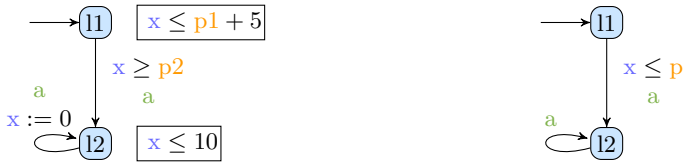
2.2 Parametric Timed Automata

Syntax

Definition 1. *A PTA $\mathcal{A}$ is a tuple $\mathcal{A} = (\Sigma, L, l_0, X, P, I, E)$, where: i) Σ is a finite set of actions, ii) L is a finite set of locations, iii) $l_0 \in L$ is the initial location, iv) X is a set of clocks, v) P is a set of parameters, vi) I is the invariant, assigning to every $l \in L$ a guard $I(l)$, vii) E is a set of edges $e = (l, g, a, R, l')$ where $l, l' \in L$ are the source and target locations, $a \in \Sigma$, $R \subseteq X$ is a set of clocks to be reset, and g is a guard.*

Given a parameter valuation v, we denote by $v(\mathcal{A})$ the non-parametric timed automaton where all occurrences of a parameter p_i have been replaced by $v(p_i)$.

Concrete Semantics



(a) PTA deadlocked for some valuations (b) PTA deadlocked for all valuations

Fig. 1: Examples of PTAs with potential deadlocks

Definition 2 (Semantics of a TA). *Given a PTA* $\mathcal{A} = (\Sigma, L, l_0, X, P, I, E)$*, and a parameter valuation* v*, the concrete semantics of* $v(\mathcal{A})$ *is given by the timed transition system* $(S, s_0, \to)$*, with*

- $S = \{(l, w) \in L \times \mathbb{R}_+^H \mid w(v(I(l)))$ *evaluates to true*$\}$, $s_0 = (l_0, \mathbf{0})$
- $\to$ *consists of the discrete and (continuous) delay transition relations:*
  - *discrete transitions:* $(l, w) \xrightarrow{e} (l', w')$*, if* $(l, w), (l', w') \in S$*, there exists* $e = (l, g, a, R, l') \in E$*,* $\forall x \in X : w'(x) = 0$ *if* $x \in R$ *and* $w'(x) = w(x)$ *otherwise, and* $w(v(g))$ *evaluates to true.*
  - *delay transitions:* $(l, w) \xrightarrow{d} (l, w + d)$*, with* $d \in \mathbb{R}_+$*, if* $\forall d' \in [0, d], (l, w + d') \in S$*.*

Moreover we write $(l, w) \overset{e}{\mapsto} (l', w')$ for a sequence of delay and discrete transitions where $((l, w), e, (l', w')) \in \mapsto$ if $\exists d, w'' : (l, w) \xrightarrow{d} (l, w'') \xrightarrow{e} (l', w')$. Given a TA $v(\mathcal{A})$ with concrete semantics $(S, s_0, \to)$, we refer to the states of S as the *concrete states* of $v(\mathcal{A})$. A *concrete run* (or simply a *run*) of $v(\mathcal{A})$ is an alternating sequence of concrete states of $v(\mathcal{A})$ and edges starting from the initial concrete state s_0 of the form $s_0 \overset{e_0}{\mapsto} s_1 \overset{e_1}{\mapsto} \cdots \overset{e_{m-1}}{\mapsto} s_m$, such that for all $i = 0, \ldots, m-1$, $e_i \in E$, and $(s_i, e_i, s_{i+1}) \in \mapsto$. Given a state $s = (l, w)$, we say that this state has no successor (or is deadlocked) if, in the concrete semantics of $v(\mathcal{A})$, there exists no discrete transition from s of from a successor of s obtained by taking exclusively continuous transition(s) from s. If no state of $v(\mathcal{A})$ is deadlocked, then $v(\mathcal{A})$ is deadlock-free.

Example 1. Consider the PTA in Fig. 1a (invariants are boxed): deadlocks can occur if the guard of the transition from l_1 to l_2 cannot be satisfied (when $p_2 > p_1 + 5$) or if the invariant of l_2 is not compatible with the guard (when $p_2 > 10$). In Fig. 1b, the system may risk a deadlock for any parameter valuation as, if the guard is "missed" (if a run chooses to spend more than p time units in l_1), then no transition can be taken from l_1.

3 Parametric Deadlock-Freeness Checking

Symbolic Semantics Let us first recall the symbolic semantics of PTAs (from, *e. g.,* [JLR15]). A symbolic state is a pair (l, C) where $l \in L$ is a location,

and C its associated parametric zone. The initial symbolic state of $\mathcal{A}$ is $\mathbf{s}_0^{\mathcal{A}} = (l_0, (\bigwedge_{1 \leq i \leq H} x_i = 0)^{\nearrow} \wedge I(l_0))$.

The symbolic semantics relies on the Succ operation. Given a symbolic state $\mathbf{s} = (l, C)$ and an edge $e = (l, g, a, R, l')$, the successor of $\mathbf{s}$ via e is the symbolic state $\mathsf{Succ}(\mathbf{s}, e) = (l', C')$, with $C' = \big([(C \wedge g)]_R\big)^{\nearrow} \cap I(l')$. We write $\mathsf{Succ}(\mathbf{s})$ for $\cup_{e \in E} \mathsf{Succ}(\mathbf{s}, e)$. We write Pred for Succ^{-1}. Given a set $\mathbf{S}$ of states, we write $\mathsf{Pred}(\mathbf{S})$ for $\bigcup_{\mathbf{s} \in \mathbf{S}} \mathsf{Pred}(\mathbf{s})$.

A symbolic run of a PTA is an alternating sequence of symbolic states and edges starting from the initial symbolic state, of the form $\mathbf{s}_0^{\mathcal{A}} \overset{e_0}{\Rightarrow} \mathbf{s}_1 \overset{e_1}{\Rightarrow} \cdots \overset{e_{m-1}}{\Rightarrow} \mathbf{s}_m$, such that for all $i = 0, \ldots, m-1$, $e_i \in E$, and $\mathbf{s}_{i+1} = \mathsf{Succ}(\mathbf{s}_i, e_i)$. The symbolic states with the Succ relation form a *state space*, *i. e.,* a (possibly infinite) directed graph, the nodes of which are the symbolic states, and there exists an edge from $\mathbf{s}_i$ to $\mathbf{s}_j$ labeled with e_i iff $\mathbf{s}_j = \mathsf{Succ}(\mathbf{s}_i, e_i)$. Given a concrete (respectively symbolic) run $(l_0, \mathbf{0}) \overset{e_0}{\mapsto} (l_1, w_1) \overset{e_1}{\mapsto} \cdots \overset{e_{m-1}}{\mapsto} (l_m, w_m)$ (respectively $(l_0, C_0) \overset{e_0}{\Rightarrow} (l_1, C_1) \overset{e_1}{\Rightarrow} \cdots \overset{e_{m-1}}{\Rightarrow} (l_m, C_m)$), its corresponding *discrete sequence* is $l_0 \overset{e_0}{\Rightarrow} l_1 \overset{e_1}{\Rightarrow} \cdots \overset{e_{m-1}}{\Rightarrow} l_m$. Two runs (concrete or symbolic) are said to be *equivalent* if their associated discrete sequences are equal.

Deadlock-freeness synthesis We now introduce below our procedure PDFC, that makes use of an intermediate, recursive procedure DSynth. Both are written in a functional form in the spirit of, *e. g.,* the reachability and unavoidability synthesis algorithms in [JLR15]. Given $\mathbf{s} = (l, C)$, we use $\mathbf{s}_C$ to denote C. The notation $g(\mathbf{s}, \mathbf{s}')$ denotes the guard of the edge from $\mathbf{s}$ to $\mathbf{s}'$.

$$\mathsf{DSynth}(\mathbf{s}, \mathsf{Passed}) = \begin{cases} \bot & \text{if } \mathbf{s} \in \mathsf{Passed} \\ \big(\bigcup_{\mathbf{s}' \in \mathsf{Succ}(\mathbf{s})} \mathsf{DSynth}(\mathbf{s}', \mathsf{Passed} \cup \{\mathbf{s}\})\big) & \\ \cup \Big(\mathbf{s}_C \setminus \big(\bigcup_{\mathbf{s}' \in \mathsf{Succ}(\mathbf{s})} \big(\mathbf{s}_C \wedge g(\mathbf{s}, \mathbf{s}')\big)^{\swarrow} \wedge \mathbf{s}'_C{\downarrow_P}\big)\Big){\downarrow_P} & \text{otherwise} \end{cases}$$

$\mathsf{PDFC}(\mathcal{A}) = \neg \mathsf{DSynth}(s_0^{\mathcal{A}}, \emptyset)$

First, we use a function $\mathsf{DSynth}(\mathbf{s}, \mathsf{Passed})$ to recursively synthesize the parameter valuations for which a deadlock may occur. This function takes as argument the current state $\mathbf{s}$ together with the list Passed of passed states. If $\mathbf{s}$ belongs to Passed (*i. e.,* $\mathbf{s}$ was already met), then no parameter valuation is returned. Otherwise, the first part of the second case computes the union over all successors of $\mathbf{s}$ of DSynth recursively called over these successors; the second part computes all parameter valuations for which a deadlock may occur, *i. e.,* the constraint characterizing $\mathbf{s}$ minus all clock and parameter valuations that allow to exit $\mathbf{s}$ to some successor $\mathbf{s}'$, all this expression being eventually projected onto P.

Finally, PDFC ("parametric deadlock-freeness checking") returns the negation of the result of DSynth called with the initial state of $\mathcal{A}$ and an empty list of passed states.

We show below that PDFC is sound and complete. Note however that, in the general case, the algorithm may not terminate, as DSynth explores the set of symbolic states, of which there may be an infinite number.

Proposition 1. *Assume* PDFC($\mathcal{A}$) *terminates with result* K*. Let* $v \models K$*. Then* $v(\mathcal{A})$ *is deadlock-free.*

Proof (sketch). Consider a run of $v(\mathcal{A})$, and assume it reaches a state $s = (l, w)$ with no discrete successor. From [HRSV02], there exists an equivalent symbolic run in $\mathcal{A}$ reaching a state (l, C). As DSynth explores all symbolic states, (l, C) was explored in DSynth too; since s has no successor, then w belongs to the second part of the second line of DSynth. Hence, the projection onto P was added to the result of DSynth, and hence does not belong to the negation returned by PDFC. Hence $v \not\models K$, which contradicts the initial assumption.

Proposition 2. *Assume* PDFC($\mathcal{A}$) *terminates with result* K*. Consider a valuation* v *such that* $v(\mathcal{A})$ *is deadlock-free. Then* $v \models K$*.*

Proof (sketch). Following a reasoning dual to Proposition 1.

Finally, we show that PDFC outputs an over-approximation of the parameter set when stopped before termination.

Proposition 3. *Fix a maximum number of recursive calls in* DSynth*. Then* PDFC *terminates and its result is an over-approximation of the set of parameter valuations for which the system is deadlock-free.*

Proof. Observe that, in DSynth, the deeper the algorithm goes in the state space (*i. e.,* the more recursive calls are performed), the more valuations it synthesizes. Hence bounding the number of recursive calls yields an under-approximation of its expected result. As PDFC returns the negation of DSynth, this yields an over-approximation.

4 Under-approximated Synthesis

A limitation of PDFC is that either the result is exact, or it is an over-approximation when stopped earlier than the actual fixpoint (from Proposition 3). In the latter case, the result is not entirely satisfactory: if deadlocks represent an undesired behavior, then an over-approximation may also contain unsafe parameter valuations. More valuable would be an under-approximation, as this result (although potentially incomplete) firmly guarantees the absence of deadlocks in the model.

Our idea is as follows: after exploring a part of the state space in PDFC, we obtain an over-approximation. In order to get an under-approximation, we can consider that any unexplored state is unsafe, *i. e.,* may lead to deadlocks. Therefore, we first need to negate the parametric constraint associated with any state that has unexplored successors. But this may not be sufficient: by removing

Algorithm 1: BwUS$(K, \mathcal{G})$

input : result K of DSynth, parametric state space $\mathcal{G}$
output: Constraint over the parameters guaranteeing deadlock-freeness

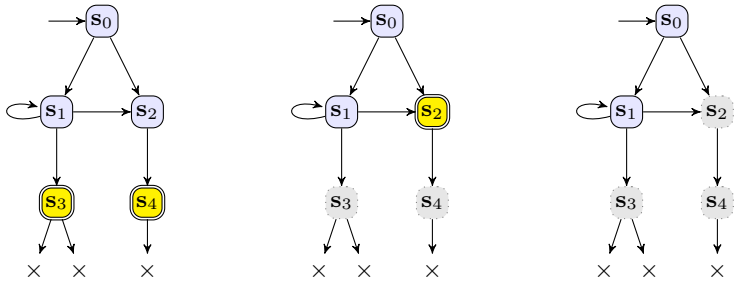
1 $K^+ \leftarrow K$
2 Marked $\leftarrow \{\mathbf{s} \mid \mathbf{s}$ has unexplored successors in $\mathcal{G}\}$
3 Disabled $\leftarrow \emptyset$
4 **while** Marked $\neq \emptyset$ **do**
5 | **foreach** $(l, C) \in$ Marked **do** $K^+ \leftarrow K^+ \cup C{\downarrow_P}$;
6 | preds $\leftarrow$ Pred(Marked) $\setminus$ Disabled
7 | Marked$' \leftarrow \emptyset$
8 | **foreach** $\mathbf{s} \in$ preds **do**
9 | | $K^+ \leftarrow K^+ \cup \Big(\mathbf{s}_C \setminus \big(\bigcup_{\mathbf{s}' \in \mathsf{Succ}(\mathbf{s})} \big(\mathbf{s}_C \wedge g(\mathbf{s}, \mathbf{s}')\big)^{\swarrow} \wedge \mathbf{s}'_C{\downarrow_P}\big)\Big){\downarrow_P}$
10 | | **if** $\mathbf{s}_C{\downarrow_P} \subseteq K^+$ **then** Marked$' \leftarrow$ Marked$' \cup \{\mathbf{s}\}$;
11 | Disabled $\leftarrow$ Disabled $\cup$ Marked ; Marked $\leftarrow$ Marked$' \setminus$ Disabled
12 **return** $\neg K^+$

those unsafe states, their predecessors can themselves become deadlocked, and so on. Hence, we will perform a backward-exploration of the state space by iteratively removing unsafe states, until a fixpoint is reached (or the constraint becomes false).

We give our procedure BwUS (backward under-approximated synthesis) in Algorithm 1. BwUS takes as input *1*) the (under-approximated) result of DSynth, *i. e.,* a set of parameter valuations for which the system contains a deadlocked run, and *2*) the part of the symbolic state space explored while running DSynth.

The algorithm maintains several variables. Marked denotes the states that are potentially deadlocked, and the predecessors of which must be considered iteratively. Disabled denotes the states marked in the past, which avoids to consider several times the same state. K^+ is an over-approximated constraint for which there are deadlocks; since the negation of K^+ is returned (line 12), then the algorithm returns an under-approximation. Initially, K^+ is set to the (under-approximated) result of DSynth, and all states that have unexplored successors in the state space (due to the early termination) are marked.

While there are marked states (line 4), we remove the marked states by adding to K^+ the negation of the constraint associated with these states (line 5). Then, we compute the predecessors of these states, except those already disabled (line 6). By definition, all these predecessors have at least one marked (and hence potentially deadlocked) successor; as a consequence, we have to recompute the constraint leading to a deadlock from each of these predecessors. This is the purpose of line 9, where K^+ is enriched with the recomputed valuations for which a deadlock may occur from a given predecessor $\mathbf{s}$, using the same computation as in DSynth. Then, if the constraint associated with the current predecessor $\mathbf{s}$ is included in K^+, this means that $\mathbf{s}$ is unreachable for valuations in $\neg K^+$ (recall that we will return the negation of K^+), and therefore $\mathbf{s}$ should be marked



(a) Partial state space (b) Iteration 1 (c) Iteration 2

Fig. 2: Application of Algorithm 1

at the next iteration, denoted by the local variable Marked' (line 10). Finally, all currently marked states become disabled, and the new marked states are all the marked predecessors of the currently marked states with the exception of the disabled states to ensure termination (line 11). The algorithm returns eventually the negation of the over-approximated K^+ (line 12), which yields an under-approximation.

Example 2. Let us apply Algorithm 1 to a (fictional) example of a partial state space, given in Fig. 2a. We only focus on the backward exploration, and rule out the constraint update (constraints are not represented in Fig. 2a anyway). $\mathbf{s}_3$ and $\mathbf{s}_4$ have unexplored successors (denoted by $\times$), and both states are hence unsafe as they might lead to deadlocks along these unexplored branches.

Initially, $\mathsf{Marked} = \{\mathbf{s}_3, \mathbf{s}_4\}$ (depicted in yellow with a double circle in Fig. 2a), and no states are disabled. First, we add $\mathbf{s}_{3C}{\downarrow_P} \cup \mathbf{s}_{4C}{\downarrow_P}$ to K^+. Then, preds is set to $\{\mathbf{s}_1, \mathbf{s}_2\}$. We recompute the deadlock constraint for both states (using line 9 in Algorithm 1). For $\mathbf{s}_2$, it now has no successors anymore, and clearly we will have $\mathbf{s}_{2C}{\downarrow_P} \subseteq K^+$, hence $\mathbf{s}_2$ is marked. For $\mathbf{s}_1$, it depends on the actual constraints; let us assume in this example that $\mathbf{s}_1$ is still not deadlocked for some valuations, and $\mathbf{s}_1$ remains unmarked. At the end of this iteration, $\mathsf{Marked} = \{\mathbf{s}_2\}$ and $\mathsf{Disabled} = \{\mathbf{s}_3, \mathbf{s}_4\}$.

For the second iteration, we assume here (it actually depends on the constraints) that $\mathbf{s}_1$ will not be marked, leading to a fixpoint where $\mathbf{s}_2, \mathbf{s}_3, \mathbf{s}_4$ are disabled, and the constraint $\neg K^+$ therefore characterizes the deadlock-free runs in Fig. 2c. (Alternatively, if $\mathbf{s}_1$ was marked, then $\mathbf{s}_0$ would be eventually marked too, and the result would be $\bot$.)

First note that BwUS necessarily terminates as it iterates on marked states, and no state can be marked twice thanks to the set $\mathsf{Disabled}$. In addition, the result is an under-approximation of the valuation set yielding deadlock-freeness: indeed, it only explores a part of state space, and considers states with unexplored successors as deadlocked by default, yielding a possibly too strong, hence under-approximated, constraint.

Case study	$\lvert\mathcal{A}\rvert$	$\lvert X\rvert$	$\lvert P\rvert$	States	PDFC	BwUS	K	Soundness
Fig. 1a	1	1	2	3	0.012	-	nncc	exact
Fig. 1b	1	1	1	2	0.005	-	$\bot$	exact
and–or circuit	4	4	4	5,265	TO	171	$[\text{nncc}^-, \text{nncc}^+]$	under/over-app
coffee machine 1	1	2	3	9,042	TO	8.4	$[\text{nncc}^-, \text{nncc}^+]$	under/over-app
coffee machine 2	2	3	3	51	0.198	-	nncc	exact
CSMA/CD protocol	3	3	3	38	0.105	-	$\bot$	exact
flip-flop circuit	6	5	2	20	0.093	-	$\bot$	exact
nuclear plant	1	2	4	13	0.014	-	nncc	exact
RCP protocol	5	6	5	2,091	10.63	-	$\bot$	exact
SIMOP	5	8	2	22,894	TO	121	nncc	over-app
Train controller	1	2	3	11	0.025	-	nncc	exact
WFAS	3	4	2	14,614	TO	69.1	$[\text{nncc}^-, \text{nncc}^+]$	under/over-app

Table 1: Synthesizing parameter valuations ensuring deadlock-freeness

5 Experiments

We implemented PDFC in IMITATOR [AFKS12] (which relies on PPL [BHZ08] for polyhedra operations), and synthesized constraints for which a set of models of distributed systems are deadlock-free.[3] Our benchmarks come from teaching examples (coffee machines, nuclear plant, train controller), communication protocols (CSMA/CD [KNSW07], RCP [CS01]), asynchronous circuits (and–or [CC05], flip-flop [CC04]), a distributed networked automation system (SIMOP [ACD+09]) and a Wireless Fire Alarm System (WFAS) [BBLS15].

If an experiment has not finished within 300 s, the result is still a valid over-approximation according to Proposition 3; in addition, IMITATOR then runs Algorithm 1 to also obtain an under-approximation.

We give in Table 1 from left to right the numbers of PTA components[4], of clocks, of parameters, and of symbolic states explored, the computation time in seconds (TO denotes no termination within 300 s) for PDFC and BwUS (when necessary), the type of constraint (nncc denotes a non-necessarily convex constraint different from $\top$ or $\bot$) and an evaluation of the result soundness.

Analyzing the experiments, several situations occur: the most interesting result is when an nncc is derived and is exact; for example; the constraint synthesized by IMITATOR for Fig. 1a is $p_1 + 5 \geq p_2 \wedge p_2 \leq 10$, which is exactly the valuation set ensuring the absence of deadlocks. In several cases, the synthesized constraint is $\bot$, meaning that no parameter valuation is deadlock-free; this may not always denote an ill-formed model, as some case studies are "finite" (no infinite behavior), typically some of the hardware case studies (*e. g.,* flip-flop);

[3] Experiments were conducted on Linux Mint 17 64 bits, running on a Dell Intel Core i7 CPU 2.67GHz with 4 GiB. Binaries, models and results are available at `www.imitator.fr/static/ICTAC16/`.

[4] The synchronous product of several PTA components (using synchronized actions) yields a PTA. IMITATOR performs this composition on-the-fly.

this may also denote a modeling process purposely blocking the system (to limit the state space explosion) after some property (typically reachability) is proved correct or violated. When no exact result could be synthesized, our second procedure BwUS allows to get both an under-approximated and an over-approximated constraint (denoted by $[\text{nncc}^-, \text{nncc}^+]$). This is a valuable result, as it contains valuations guaranteed to be deadlock-free, others guaranteed to be deadlocked, and a third unsure set. An exception is SIMOP, where BwUS derives $\bot$, leaving the designer with only an over-approximation. This result remains valuable as the parameter valuations not belonging to the synthesized constraint necessarily lead to deadlocks, an information that will help the designer to refine its model, or to rule out these valuations.

Concerning the performances of BwUS, its overhead is significant, and depends on the number of dimensions (clocks and parameters) as well as the number states in the state space. However, it still remains smaller than the forward exploration (300 s) in all case studies, which therefore remains reasonable to some extent. It seems the most expensive operation is the computation of the deadlock constraint (line 9 in Algorithm 1); this has been implemented in a straightforward manner, but could benefit from optimizations (*e. g.,* only recompute the part corresponding to successor states that were disabled at the previous iteration of BwUS).

6 Perspectives

We proposed here a procedure to synthesize timing parameter valuations ensuring the absence of deadlocks in a real-time system; we implemented it in IMITATOR and we have run experiments on a set of benchmarks. When terminating, our procedure yields an exact result. Otherwise, thanks to a second procedure, we get both an under- and an over-approximation of the valuations for which the system is deadlock-free.

Our definition of deadlock-freeness addresses discrete transitions; however, in case of Zeno behaviors (an infinite number of discrete transition within a finite time), a deadlock-free system can still correspond to an ill-formed model. Hence, performing parametric Zeno-freeness checking is also on our agenda. Moreover, we are very interested in proposing distributed procedures for deadlock-freeness synthesis so as to take advantage of the power of clusters.

Finally, we believe our backward algorithm BwUS could be adapted to obtained under-approximated results for other problems such as the unavoidability synthesis in [JLR15].

References

ACD+09. Étienne André, Thomas Chatain, Olivier De Smet, Laurent Fribourg, and Silvain Ruel. Synthèse de contraintes temporisées pour une architecture d'automatisation en réseau. In Didier Lime and Olivier H. Roux, editors, *Actes du 7ème Colloque sur la Modélisation des Systèmes Réactifs*

(MSR'09), volume 43 of *Journal Européen des Systèmes Automatisés*, pages 1049–1064. Hermès, November 2009. 9

AD94. Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994. 1

AFKS12. Étienne André, Laurent Fribourg, Ulrich Kühne, and Romain Soulat. IMITATOR 2.5: A tool for analyzing robustness in scheduling problems. In *FM*, volume 7436 of *Lecture Notes in Computer Science*, pages 33–36. Springer, 2012. 9

AHV93. Rajeev Alur, Thomas A. Henzinger, and Moshe Y. Vardi. Parametric real-time reasoning. In *STOC*, pages 592–601, 1993. 2

AL16. Étienne André and Didier Lime. Liveness in L/U-parametric timed automata. Submitted. https://hal.archives-ouvertes.fr/hal-01304232, 2016. 2

BBLS15. Nikola Beneš, Peter Bezděk, Kim G. Larsen, and Jiří Srba. Language emptiness of continuous-time parametric timed automata. In *ICALP, Part II*, volume 9135 of *Lecture Notes in Computer Science*, pages 69–81. Springer, 2015. 9

BHZ08. Roberto Bagnara, Patricia M. Hill, and Enea Zaffanella. The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. *Science of Computer Programming*, 72(1–2):3–21, 2008. 9

CC04. Robert Clarisó and Jordi Cortadella. Verification of timed circuits with symbolic delays. In *ASP-DAC'04*, pages 628–633, Piscataway, NJ, USA, 2004. IEEE Press. 9

CC05. Robert Clarisó and Jordi Cortadella. Verification of concurrent systems with parametric delays using octahedra. In *ACSD*, pages 122–131. IEEE Computer Society, 2005. 9

CS01. Aurore Collomb–Annichini and Mihaela Sighireanu. Parameterized reachability analysis of the IEEE 1394 Root Contention Protocol using TReX. In *Proceedings of the Real-Time Tools Workshop (RT-TOOLS'01)*, 2001. 9

HRSV02. Thomas Hune, Judi Romijn, Mariëlle Stoelinga, and Frits W. Vaandrager. Linear parametric model checking of timed automata. *Journal of Logic and Algebraic Programming*, 52-53:183–220, 2002. 2, 6

JLR15. Aleksandra Jovanović, Didier Lime, and Olivier H. Roux. Integer parameter synthesis for timed automata. *IEEE Transactions on Software Engineering*, 41(5):445–461, 2015. 3, 4, 5, 10

KNSW07. Marta Z. Kwiatkowska, Gethin Norman, Jeremy Sproston, and Fuzhi Wang. Symbolic model checking for probabilistic timed automata. *Information and Computation*, 205(7):1027–1077, 2007. 9

LPY97. Kim Guldstrand Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1-2):134–152, 1997. 1

Mar11. Nicolas Markey. Robustness in real-time systems. In *SIES*, pages 28–34. IEEE Computer Society Press, 2011. 2

Sch86. Alexander Schrijver. *Theory of linear and integer programming.* John Wiley & Sons, Inc., New York, NY, USA, 1986. 3

SLDP09. Jun Sun, Yang Liu, Jin Song Dong, and Jun Pang. PAT: Towards flexible verification under fairness. In *CAV*, volume 5643 of *Lecture Notes in Computer Science*, pages 709–714. Springer, 2009. 1