Merge and Conquer: State Merging in Parametric Timed Automata

Étienne André[1], Laurent Fribourg[2], and Romain Soulat[2]

[1] Université Paris 13, Sorbonne Paris Cité, LIPN, F-93430, Villetaneuse, France
[2] LSV, ENS Cachan & CNRS, Cachan, France

Abstract. Parameter synthesis for real-time systems aims at synthesizing dense sets of valuations for the timing requirements, guaranteeing a good behavior. A popular formalism for modeling parameterized real-time systems is parametric timed automata (PTAs). Compacting the state space of PTAs as much as possible is fundamental. We present here a state merging reduction based on convex union, that reduces the state space, but yields an over-approximation of the executable paths. However, we show that it preserves the sets of reachable locations and executable actions. We also show that our merging technique associated with the inverse method, an algorithm for parameter synthesis, preserves locations as well, and outputs larger sets of parameter valuations.

Keywords: Parameter synthesis, state space reduction, real-time systems

1 Introduction

Ensuring the correctness of critical real-time systems, involving concurrent behaviors and timing requirements, is crucial. Formal verification methods may not always be able to verify full size systems, but they provide designers with an important help during the design phase, in order to detect otherwise costly errors. Formalisms for modeling real-time systems, such as time Petri nets or timed automata (TAs), have been extensively used in the past decades, and led to useful and efficient implementations. Parameter synthesis for real-time systems is a set of techniques aiming at synthesizing dense sets of valuations for the timing requirements of the system. We consider the delays as unknown constants, or *parameters*, and synthesize constraints on these parameters guaranteeing the system correctness; of course, the weaker the constraint (i.e., the larger the set of parameter valuations), the more interesting the result. Parameterizing TAs gives parametric timed automata (PTAs) [2].

A fundamental problem in the exploration of the reachability space in PTAs is to compact as much as possible the generated space of symbolic states. Our first contribution is to introduce a state merging technique based on convex union. Roughly speaking, two states are merged when their discrete part is the same, and the union of their respective continuous part (values of the clocks and parameters) is convex. On the one hand, this technique often considerably reduces the state space. On the other hand, exploring the state space using this technique does not reflect the standard semantics of PTAs: the set of possible paths is an over-approximation of the set of paths in the original PTAs semantics. However, we show that the state space computed using the merging reduction preserves the set of reachable locations and executable actions. That is, the sets of reachable locations and executable actions obtained using the merging reduction are the same as those obtained using the classical semantics.

The inverse method *IM* [8] is an algorithm that takes advantage of a known reference parameter valuation, and synthesizes a constraint around the reference valuation guaranteeing the same traces as for the reference valuation, i.e., the same time-abstract (or discrete) behavior. Our second contribution is to show that *IM* equipped with our merging reduction (called IM_{Mrg}) does not preserve traces anymore; however, it preserves locations (i.e., discrete reachability), and outputs a weaker constraint. However, we show that actions are not preserved in the general case. We exhibit a subclass of PTAs, namely backward-deterministic PTA, for which action preservation is guaranteed. Furthermore, we show that IM_{Mrg} outputs a weaker constraint (i.e., a larger set of parameter valuations) than *IM*, which is interesting.

Our third contribution is to define a new version IM'_{Mrg} of IM_{Mrg} that preserves not only locations but actions too, at the cost of a more restrictive constraint than IM_{Mrg}, but still weaker than *IM*. Our work is implemented in IMITATOR [4] and shows large state space reductions in many cases, especially for scheduling problems. Finally, and more surprisingly, the time overhead induced by the convexity test is often not significant in the few case studies where the state space is not reduced.

Related Work. In [19], it is shown that, in a network of TAs, all the successor states can be merged together when all the interleavings of actions are possible. However, this result does not extend to the parametric case. In [13, 14], it is proposed to replace the union of two states by a unique state when the union of their continuous part (viz., the symbolic clock values) is convex, and the discrete part (viz., the location) is identical. This technique is applied to timed constraints represented in the form of Difference Bound Matrices (DBMs). Our merging technique can be seen as an extension of the technique in [13, 14] to the parametric case. This extension is not trivial, and the implementation is necessarily different, since DBMs (in their original form) do not allow the use of parameters. Instead, we implemented our approach in IMITATOR using polyhedra [9].

Remark. This paper is an extension of a "work in progress" paper [5]. In contrast to [5], we formally define the merging operation, and characterize it in the general setting of reachability analysis for PTAs. Furthermore, we rewrite a result from [5] that erroneously stated that the inverse method with merging preserves traces; we show here that it does not, but preserves (at least) the set of locations. We also exhibit a subclass of PTAs for which IM_{Mrg} preserves actions too. We finally define a new version of the inverse method that preserves not only locations but actions as well, for general PTAs.

Outline. We recall preliminaries in Section 2. We define and characterize the merging reduction in Section 3. Section 4 is dedicated to *IM* combined with the merging reduction. We give experiments in Section 5 and conclude in Section 6.

2 Preliminaries

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

2.1 Clocks, Parameters and Constraints

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 $\mathbb{R}_+$. All clocks evolve linearly at the same rate. A *clock valuation* is a function $w : X \to \mathbb{R}_+^H$. We will often identify a valuation w with the point $(w(x_1), \dots, w(x_H))$. Given a constant $d \in \mathbb{R}_+$, we use $X + d$ to denote the set $\{x_1 + d, \dots, x_H + d\}$. Similarly, we write $w + d$ to denote the valuation such that $(w + d)(x) = w(x) + d$ for all $x \in X$.

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

An *inequality* over X and P is $e \prec e'$, where $\prec \in \{<, \leq\}$, and e, e' are two linear terms of the form $\sum_{1 \leq i \leq N} \alpha_i z_i + d$ where $z_i \in X \cup P$, $\alpha_i \in \mathbb{Q}_+$, for $1 \leq i \leq N$, and $d \in \mathbb{Q}_+$. We define similarly inequalities over X (resp. P). A *constraint* is a conjunction of inequalities. Given an inequality J over the parameters of the form $e < e'$ (respectively $e \leq e'$), the *negation* of J, denoted by $\neg J$, is the linear inequality $e' \leq e$ (respectively $e' < e$).

We denote by $\mathcal{L}(X)$, $\mathcal{L}(P)$ and $\mathcal{L}(X \cup P)$ the set of all constraints over X, over P, and over X and P respectively. In the sequel, J denotes an inequality over the parameters, $D \in \mathcal{L}(X)$, $K \in \mathcal{L}(P)$, and $C \in \mathcal{L}(X \cup P)$. A constraint over X and P can be interpreted as a set of points in the space $\mathbb{R}^{M+H}$, more precisely as a convex polyhedron.

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

Given two constraints C_1 and C_2, C_1 is said to be *included in* C_2, denoted by $C_1 \subseteq C_2$, if $\forall w, \pi : <w, \pi> \models C_1 \implies <w, \pi> \models C_2$.

A parameter valuation π *satisfies* a constraint K over the parameters, denoted by $\pi \models K$, if the expression obtained by replacing each parameter p in K with $\pi(p)$ evaluates to true. Given K_1 and K_2, K_1 is *included in* K_2, denoted by $K_1 \subseteq K_2$, if $\forall \pi : \pi \models K_1 \Longrightarrow \pi \models K_2$. We consider true as a constraint over the parameters, corresponding to the set of all possible values for P.

We denote by $C{\downarrow_P}$ the constraint over the parameters obtained by projecting C onto the set of parameters, that is after elimination of the clock variables.

Sometimes we will refer to a variable domain X', which is obtained by renaming the variables in X. Explicit renaming of variables is denoted by the substitution operation. Given a constraint C over the clocks and the parameters, we denote by $C_{[X \leftarrow X']}$ the constraint obtained by replacing in C the variables of X with the variables of X'. We sometime write $C(X)$ or $C(X')$ to denote the set of clocks used within C.

We define the *time elapsing* of C, denoted by $C^{\uparrow}$, as the constraint over X and P obtained from C by delaying an arbitrary amount of time. Formally:

$$C^{\uparrow} = \Big((C \wedge X' = X + d){\downarrow_{X' \cup P}} \Big)_{[X' \leftarrow X]}$$

where d is a new parameter with values in $\mathbb{R}_+$, and X' is a renamed set of clocks. The inner part of the expression adds the same delay d to all clocks; the projection onto $X' \cup P$ eliminates the original set of clocks X, as well as the variable d; the outer part of the expression renames clocks X' with X.

2.2 Labeled Transition Systems

We introduce below labeled transition systems, which will be used later in this section to define the semantics of PTAs.

Definition 1. *A* labeled transition system *is a quadruple* $\mathcal{LTS} = (\Sigma, S, S_0, \Rightarrow)$*, with* Σ *a set of symbols,* S *a set of* states*,* $S_0 \subset S$ *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* run *(of length* m*) of* $\mathcal{LTS}$ *is an alternating sequence of states* $s_i \in S$ *and symbols* $a_i \in \Sigma$ *of the form* $s_0 \overset{a_0}{\Rightarrow} s_1 \overset{a_1}{\Rightarrow} \cdots \overset{a_{m-1}}{\Rightarrow} s_m$*, where* $s_0 \in S_0$*. A state* s_i *is* reachable *if it belongs to some run* r*.*

2.3 Parametric Timed Automata

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

Definition 2 (Parametric Timed Automaton). *A* parametric timed automaton (PTA) $\mathcal{A}$ *is a 8-tuple of the form* $\mathcal{A} = (\Sigma, L, l_0, X, P, K, I, \rightarrow)$*, where*

- Σ *is a finite set of actions,*
- L *is a finite set of locations,* $l_0 \in L$ *is the initial location,*
- X *is a set of clocks,* P *is a set of parameters,*
- $K \in \mathcal{L}(P)$ *is the initial constraint,*
- I *is the invariant, assigning to every* $l \in L$ *a constraint* $I(l) \in \mathcal{L}(X \cup P)$*,*

– $\rightarrow$ *is a step relation consisting of elements of the form* (l, g, a, ρ, l') *where* $l, l' \in L$ *are the source and destination locations,* $a \in \Sigma$, $\rho \subseteq X$ *is a set of clocks to be reset by the step, and* $g \in \mathcal{L}(X \cup P)$ *is the step guard.*

The constraint K corresponds to the *initial* constraint over the parameters, i.e., a constraint that will be true in all the states of $\mathcal{A}$. For example, in a PTA with two parameters *min* and *max*, we may want to constrain *min* to be always smaller or equal to *max*, in which case K is defined as $min \leq max$.

Given a PTA $\mathcal{A} = (\Sigma, L, l_0, X, P, K, I, \rightarrow)$, for every parameter valuation π, $\mathcal{A}[\pi]$ denotes the PTA $(\Sigma, L, l_0, X, P, K_\pi, I, \rightarrow)$, where $K_\pi = K \wedge \bigwedge_{i=1}^{M} p_i = \pi(p_i)$. This corresponds to the PTA obtained from $\mathcal{A}$ by substituting every occurrence of a parameter p_i by constant $\pi(p_i)$ in the guards and invariants. Note that $\mathcal{A}[\pi]$ is a non-parametric timed automaton.

In the following, given a PTA $\mathcal{A} = (\Sigma, L, l_0, X, P, K, I, \rightarrow)$ and when clear from the context, we will often denote this PTA by $\mathcal{A}(K)$, in order to emphasize the value of K in $\mathcal{A}$.

The (symbolic) semantics of PTAs relies on the notion of state, i.e., a pair (l, C) where $l \in L$ is a location, and $C \in \mathcal{L}(X \cup P)$ its associated constraint. For each valuation π of P, we may view a state s as the set of pairs (l, w) where w is a clock valuation such that $<w, \pi> \models C$.

A state $s = (l, C)$ of a PTA $\mathcal{A}$ is π*-compatible* if $\pi \models C$. We say that a set of states S_1 is *included* into a set of states S_2, denoted by $S_1 \sqsubseteq S_2$, if $\forall s : s \in S_1 \Longrightarrow s \in S_2$.

The *initial state* of $\mathcal{A}(K)$ is $s_0 = (l_0, C_0)$, where $C_0 = K \wedge I(l_0) \wedge \bigwedge_{i=1}^{H-1} x_i = x_{i+1}$. In this expression, K is the initial constraint over the parameters, $I(l_0)$ is the invariant of the initial state, and the rest of the expression lets clocks evolve from the same initial value.

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

Definition 3 (Semantics of PTAs). *Let* $\mathcal{A} = (\Sigma, L, l_0, X, P, K, I, \rightarrow)$ *be a PTA. The* semantics of $\mathcal{A}$ *is* $\mathcal{LTS}(\mathcal{A}) = (\Sigma, S, S_0, \Rightarrow)$ *where*

$$\begin{aligned} S &= \{(l, C) \in L \times \mathcal{L}(X \cup P) \mid C \subseteq I(l)\}, \\ S_0 &= \{s_0\} \end{aligned}$$

and a transition $(l, C) \overset{a}{\Rightarrow} (l', C')$ *belongs to* $\Rightarrow$ *if* $\exists C'' : (l, C) \overset{a}{\rightarrow} (l', C'') \overset{d}{\rightarrow} (l', C')$*, with*

– *discrete transitions* $(l, C) \overset{a}{\rightarrow} (l', C')$ *if there exists* $(l, g, a, \rho, l') \in \rightarrow$ *and*

$$C' = \Big(\big(C(X) \wedge g(X) \wedge X' = \rho(X) \big) \downarrow_{X' \cup P} \wedge I(l')(X') \Big)_{[X' \leftarrow X]} \text{ and}$$

– *delay transitions* $(l, C) \overset{d}{\rightarrow} (l, C')$ *with* $C' = C^{\uparrow} \wedge I(l)(X)$.

Let $\mathcal{LTS}(\mathcal{A}) = (\Sigma, S, S_0, \Rightarrow)$. When clear from the context, given $(s_1, a, s_2) \in \Rightarrow$, we write $(s_1 \overset{a}{\Rightarrow} s_2) \in \Rightarrow(\mathcal{A})$.

A *path* of $\mathcal{A}$ is an 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 $s_i \overset{a_i}{\Rightarrow} s_{i+1} \in \Rightarrow(\mathcal{A})$. The set of all paths of $\mathcal{A}$ is denoted by $\mathit{Paths}(\mathcal{A})$. We define *traces* as time-abstract paths. Given a path $(l_0, C_0) \overset{a_0}{\Rightarrow} (l_1, C_1) \overset{a_1}{\Rightarrow} \cdots \overset{a_{m-1}}{\Rightarrow} (l_m, C_m)$, the corresponding trace is $l_0 \overset{a_0}{\Rightarrow} l_1 \overset{a_1}{\Rightarrow} \cdots \overset{a_{m-1}}{\Rightarrow} l_m$. The set of all traces of $\mathcal{A}$ (or *trace set*) is denoted by $\mathit{Traces}(\mathcal{A})$.

The *Post* operation computes the successors of a state. Formally, $\mathit{Post}_{\mathcal{A}}(s) = \{s' \mid \exists a \in \Sigma : (s \overset{a}{\Rightarrow} s') \in \Rightarrow(\mathcal{A})\}$. We define $\mathit{Post}^i_{\mathcal{A}}(s)$ as the set of states reachable from a state s_0 in exactly i steps. The *Post* operation extends to a set S of states: $\mathit{Post}_{\mathcal{A}}(S) = \bigcup_{s \in S} \mathit{Post}_{\mathcal{A}}(s)$. And similarly for $\mathit{Post}^i_{\mathcal{A}}(S)$. We write $\mathit{Post}^*_{\mathcal{A}}(S) = \bigcup_{i \geq 0} \mathit{Post}^i_{\mathcal{A}}(S)$.

Given a PTA $\mathcal{A}$ of initial state s_0, we write $\mathit{Reach}^i(\mathcal{A})$ (resp. $\mathit{Reach}^*(\mathcal{A})$) for $\mathit{Post}^i_{\mathcal{A}}(\{s_0\})$ (resp. $\mathit{Post}^*_{\mathcal{A}}(\{s_0\})$). We also define $\mathit{Locations}(\mathcal{A})$ (resp. $\mathit{Actions}(\mathcal{A})$) as the set of locations (resp. actions) reachable (resp. executable) from the initial state of $\mathcal{A}$. We will often use these notations with $\mathcal{A}(K)$ in place of $\mathcal{A}$.

Remark 1. For sake of conciseness, we do not recall the concrete semantics of PTAs here. Our symbolic semantics is commonly used (see, e.g., [17, 8]), and it is clear that the sets $\mathit{Locations}(\mathcal{A})$ and $\mathit{Actions}(\mathcal{A})$ are the same for both the symbolic and concrete semantics. □

2.4 The Inverse Method

The inverse method *IM* is a semi-algorithm (i.e., if it terminates, its result is correct) that takes as input a PTA $\mathcal{A}$ and a reference parameter valuation π, and synthesizes a constraint K over the parameters such that, for all $\pi' \models K$, $\mathcal{A}[\pi]$ and $\mathcal{A}[\pi']$ have the same trace sets [8].

Algorithm 1: Inverse method $\mathit{IM}(\mathcal{A}, \pi)$

```
input : PTA $\mathcal{A}$ of initial state $s_0$, parameter valuation $\pi$
output: Constraint $K$ over the parameters
1 $i \leftarrow 0$ ; $K_c \leftarrow$ true ; $S_{new} \leftarrow \{s_0\}$ ; $S \leftarrow \{\}$
2 while true do
3     while there are $\pi$-incompatible states in $S_{new}$ do
4         Select a $\pi$-incompatible state $(l, C)$ of $S_{new}$ (i.e., s.t. $\pi \not\models C$) ;
5         Select a $\pi$-incompatible $J$ in $C{\downarrow_P}$ (i.e., s.t. $\pi \not\models J$) ;
6         $K_c \leftarrow K_c \wedge \neg J$ ; $S \leftarrow \bigcup_{j=0}^{i-1} \mathit{Post}^j_{\mathcal{A}(K_c)}(\{s_0\})$ ; $S_{new} \leftarrow \mathit{Post}_{\mathcal{A}(K_c)}(S)$ ;
7     if $S_{new} \sqsubseteq S$ then return $K \leftarrow \bigcap_{(l,C) \in S} C{\downarrow_P}$
8     $i \leftarrow i + 1$ ; $S \leftarrow S \cup S_{new}$ ; $S_{new} \leftarrow \mathit{Post}_{\mathcal{A}(K_c)}(S)$
```

IM, recalled in Algorithm 1, uses 4 variables: an integer i measuring the depth of the state space exploration, the current constraint K_c, the set S of states explored at previous iterations, and a set S_{new} of states explored at the current iteration i. Starting from the initial state s_0, *IM* iteratively computes reachable states. When a π-incompatible state is found, an incompatible inequality is non-deterministically selected within the projection onto P of the constraint (line 5); its negation is then added to K_c (line 6). The set of reachable states is then updated. When all successor states have already been reached (line 7), *IM* returns the intersection K of the projection onto P of the constraints associated with all the reachable states. Otherwise, the exploration goes one step further (line 8). Recall from [8] that *IM* is non-deterministic, and hence its result may be non-complete, i.e., the resulting constraint may not be the weakest constraint guaranteeing the preservation of trace sets.

3 Merging States in Parametric Timed Automata

3.1 Principle

We extend here the notion of merging from [13] to the parametric case.

Definition 4. *Two states* $s_1 = (l_1, C_1)$ *and* (l_2, C_2) *are* mergeable *if* $l_1 = l_2$ *and* $C_1 \cup C_2$ *is convex; then,* $(l_1, C_1 \cup C_2)$ *is their* merging *denoted by* $\mathit{merge}(s_1, s_2)$.

Given a set S of states, $\mathit{Merge}(S)$ denotes the result of applying iteratively the merging of a pair of states of S until no further merging applies, as given in Algorithm 2.

Algorithm 2: Merging a set of states

```
input  : Set $S$ of states
output: Merged set $S$ of states
1 $Q \leftarrow S$ ;
2 while $\exists (l, C_1), (l, C_2) \in Q$ such that $C_1 \neq C_2$ and $C_1 \cup C_2$ is convex do
3     $Q \leftarrow Q \setminus \{(l, C_1), (l, C_2)\} \cup \{\mathit{merge}((l, C_1), (l, C_2))\}$ ;
4 return $Q$
```

Remark. This process is not deterministic, i.e., the result depends on the order of the iterative merging operations of pairs of states. Consider three states $(l, C_1), (l, C_2), (l, C_3)$ such that $C_1 \cup C_2$ and $C_2 \cup C_3$ are convex, but $C_1 \cup C_3$ is not. This situation is depicted in Fig. 1 with 2 parameter dimensions. In that case, two possible sets of states can result from an application of the merging to these 3 states. That is, either $\{(l, C_1), (l, C_2 \cup C_3)\}$ or $\{(l, C_1 \cup C_2), (l, C_3)\}$.

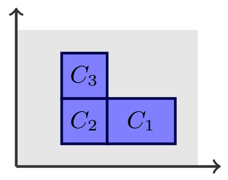


Fig. 1: Non-determinism

3.2 Merging and Reachability

We define below the semantics of PTAs with merging.

Definition 5. *Let $\mathcal{A} = (\Sigma, L, l_0, X, P, K, I, \rightarrow)$ be a PTA. The* semantics of $\mathcal{A}$ with merging *is $\mathcal{LTS}_{Mrg}(\mathcal{A}) = (\Sigma, S, S_0, \Rightarrow_{Mrg})$ where*

$$S = \{(l, C) \in L \times \mathcal{L}(X \cup P) \mid C \subseteq I(l)\},$$
$$S_0 = \{(l_0, K \wedge I(l_0) \wedge \textstyle\bigwedge_{i=1}^{H-1} x_i = x_{i+1})\}$$

and a transition $(l, C) \overset{a}{\Rightarrow} (l', C')$ belongs to $\Rightarrow_{Mrg}$ if there exists $n \in \mathbb{N}$ such that $(l, C) \in ReachM^n$, and $(l', C') \in ReachM^{n+1}$, where $ReachM^n$ is inductively defined as follows:

- *$ReachM^0 = S_0$, and*
- *$ReachM^{i+1} = Merge\big(Post_{\mathcal{A}}(ReachM^i)\big)$ for all $i \in \mathbb{N}$.*

Recall that *Post* is defined using the $\Rightarrow$ relation of $\mathcal{A}$ without merging. Hence the semantics of PTAs with merging iteratively computes states (using the standard transition relation), and merges the new states at each iteration.

Then we define $\Rightarrow^i_{Mrg}$, *PostM*, $ReachM^*$, *PathsM*, *TracesM*, *LocationsM* and *ActionsM* the same way as $\Rightarrow^i$, *Post*, $Reach^*$, *Paths*, *Traces*, *Locations* and *Actions*, respectively, by replacing within their respective definition $\Rightarrow$ with $\Rightarrow_{Mrg}$. Observe that, from the definition of $\Rightarrow_{Mrg}$ in Definition 5, *PostM* can be defined as *Post* followed by *Merge*, i.e., $PostM = Merge \circ Post$.

3.3 Characterization of the Merging Reduction

The following lemma states that the initial state of any path (hence, including of length 0) of $\mathcal{A}$ without merging is the same for $\mathcal{A}$ with merging.

Lemma 1. *Let $\mathcal{A}$ be a PTA. Then $Reach^0(\mathcal{A}) = ReachM^0(\mathcal{A})$.*

Proof. From Definitions 3 and 5. ■

The main property preserved by merging states while generating the reachability graph is the preservation of each time-abstract transition, i.e., taken one by one. In other words, for each time-abstract transition $l_1 \overset{a}{\Rightarrow} l_2$ in the graph obtained without merging, there is a corresponding time-abstract transition $l_1 \overset{a}{\Rightarrow} l_2$ in the graph obtained with merging. However, this does not extend to traces.

The characterization of merging will be stated in Theorem 1. This result relies[1] on the two forthcoming lemmas 2 and 3.

Lemma 2 (Merging and reachability ($\Longrightarrow$)). *Let $\mathcal{A}$ be a PTA. Let $(l_0, C_0) \overset{a_0}{\Rightarrow} \dots \overset{a_{n-1}}{\Rightarrow} (l_n, C_n) \in Paths(\mathcal{A})$. Then there exist $C'_1, \dots, C'_n$ such that:*

1. *$(l_0, C_0) \overset{a_0}{\Rightarrow}_{Mrg} (l_0, C'_1) \overset{a_1}{\Rightarrow}_{Mrg} \dots \overset{a_{n-1}}{\Rightarrow}_{Mrg} (l_n, C'_n) \in PathsM(\mathcal{A})$, and*

[1] The proofs of all results can be found in [6].

2. $C_i \subseteq C'_i$, *for all* $1 \leq i \leq n$.

We show in Lemma 3 that the constraint associated to each state in the merged graph is the union of several constraints in the non-merged graph.

Lemma 3 (Merging and reachability ($\Longleftarrow$)). *Let $\mathcal{A}$ be a PTA. For all $n \in \mathbb{N}$, for all $(l, C) \in ReachM^n(\mathcal{A})$, there exist $m \in \mathbb{N}$ and $(l, C_1), \dots, (l, C_m) \in Reach^*(\mathcal{A})$ such that*

$$C = \bigcup_{1 \leq i \leq m} C_i.$$

We can finally characterize the merging in the following theorem.

Theorem 1 (Merging states in PTAs). *Let $\mathcal{A}$ be a PTA. Then:*

1. *For all $(l_0, C_0) \overset{a_0}{\Rightarrow} \dots \overset{a_{n-1}}{\Rightarrow} (l_n, C_n) \in Paths(\mathcal{A})$, there exist $C'_1, \dots, C'_n$ such that:*
   (a) $(l_0, C_0) \overset{a_0}{\Rightarrow}_{Mrg} (l_0, C'_1) \overset{a_1}{\Rightarrow}_{Mrg} \dots \overset{a_{n-1}}{\Rightarrow}_{Mrg} (l_n, C'_n) \in PathsM(\mathcal{A})$, and
   (b) $C_i \subseteq C'_i$, for all $1 \leq i \leq n$.
2. *For all $(l, C) \in ReachM^*(\mathcal{A})$ there exist $m \in \mathbb{N}$ and $(l, C_1), \dots, (l, C_m) \in Reach^*(\mathcal{A})$ such that $C = \bigcup_{1 \leq i \leq m} C_i$.*

Proof. From Lemmas 2 and 3. ■

We can derive several results from Theorem 1.

First, each trace in the non-merged graph exists in the merged graph. (Note that the converse statement does not hold.) Hence, *TracesM*$(\mathcal{A})$ is an over-approximation of *Traces*$(\mathcal{A})$.

Corollary 1 (Inclusion of traces). *$Traces(\mathcal{A}) \subseteq TracesM(\mathcal{A})$.*

We state below that each timed-abstract transition in the non-merged graph exists in the merged graph, and vice versa. (Note that this cannot be generalized to complete traces.)

Corollary 2 (Preservation of time-abstract transitions). *Let $\mathcal{A}$ be a PTA.*

1. *Let $l \overset{a}{\Rightarrow} l' \in Traces(\mathcal{A})$. Then $l \overset{a}{\Rightarrow}_{Mrg} l' \in TracesM(\mathcal{A})$.*
2. *Let $l \overset{a}{\Rightarrow}_{Mrg} l' \in TracesM(\mathcal{A})$. Then $l \overset{a}{\Rightarrow} l' \in Traces(\mathcal{A})$.*

Finally, locations and actions are preserved by the merging reduction.

Corollary 3 (Preservation of locations and actions). *Let $\mathcal{A}$ be a PTA. Then: $Locations(\mathcal{A}) = LocationsM(\mathcal{A})$ and $Actions(\mathcal{A}) = ActionsM(\mathcal{A})$.*

To summarize, computing the set of reachable states using the merging reduction yields an over-approximation of the set of paths. In the original semantics, each trace of $\mathcal{A}(K)$ exists in $\mathcal{A}[\pi]$ for at least one valuation $\pi \models K$; this is not the case anymore with the use of merging, where some traces in $\mathcal{A}(K)$ may not exist for any $\pi \models K$. Nevertheless, both the set of reachable locations and the set of actions are identical to those computed using the original semantics. As a consequence, the merging reduction can be safely used to verify the reachability or the non-reachability of a (set of) location(s), but not to verify more complex properties such as properties on traces (linear-time formulas).

4 The Inverse Method with Merging

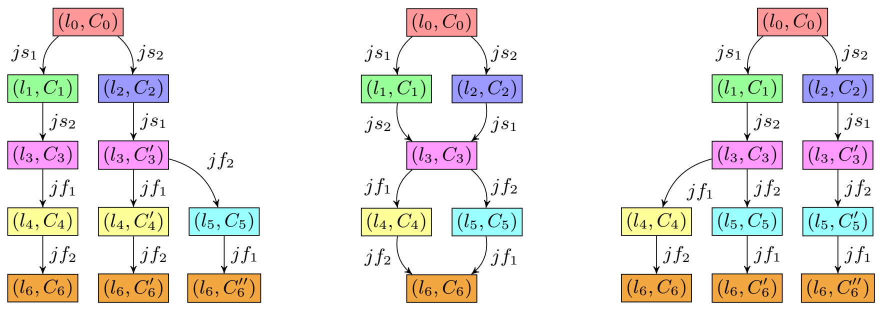
4.1 Principle

We extend *IM* with the merging operation, by merging states within the algorithm, i.e., by replacing within Algorithm 1 all occurrences of *Post* with *PostM*. (The extension IM_{Mrg} is given in [6].)

Remark 2. In IM_{Mrg}, states are merged *before* the π-compatibility test. Hence, some π-incompatible states may possibly be merged, and hence become π-compatible. As a consequence, less inequalities will be negated and added to K_c, thus giving a weaker output constraint K_{Mrg}. Also note that the addition of merging to *IM* adds a new reason for non-confluence since the merging process is itself non-deterministic. □

We will see that, in contrast to *IM*, IM_{Mrg} does not preserve traces. That is, given $\pi, \pi' \models K_{Mrg}$, a trace in $\mathcal{A}[\pi]$ may not exist in $\mathcal{A}[\pi']$, and vice versa.

Example 1. We use here a jobshop example in the setting of parametric schedulability [16], in order to show that the traces are no longer preserved with IM_{Mrg}. This system (modeled by a PTA $\mathcal{A}$) contains 2 machines on which 2 jobs should be performed. The system parameters are d_i (for $i = 1, 2$) that encode the duration of each job. The system actions are js_1 (job 1 starting), jf_1 (job 1 finishing) and similarly for job 2.



(a) Trace set of $\mathcal{A}(\pi)$ (b) For $IM_{Mrg}(\mathcal{A}, \pi)$ (c) Trace set of $\mathcal{A}(\pi)$

Fig. 2: Trace sets of $\mathcal{A}$

Consider $\pi = \{d_1 := 1, d_2 := 2\}$. The trace set of $\mathcal{A}[\pi]$ using the standard semantics (Definition 3) is given in Fig. 2a (in the form of a graph). Applying *IM* to $\mathcal{A}$ and π gives $K = d_2 > d_1$. From the correctness of *IM* [8], the trace set of $\mathcal{A}[\pi']$, for all $\pi' \models K$, is the same as for $\mathcal{A}[\pi]$. Now, applying IM_{Mrg} to $\mathcal{A}$ and π gives K_{Mrg} = true; the merged trace set is given in Fig. 2b. Then, let $\pi' = \{d_1 := 2, d_2 := 1\}$ be a valuation in K_{Mrg} but outside of K. The trace set of $\mathcal{A}[\pi']$ (using the standard semantics) is given in Fig. 2c. The trace sets of $\mathcal{A}[\pi]$

and $\mathcal{A}[\pi']$ are different: the trace $l_0 \overset{js_2}{\Rightarrow} l_2 \overset{js_1}{\Rightarrow} l_3 \overset{jf_1}{\Rightarrow} l_4 \overset{jf_2}{\Rightarrow} l_6$ exists in $\mathcal{A}[\pi]$ but not in $\mathcal{A}[\pi']$; the trace $l_0 \overset{js_1}{\Rightarrow} l_1 \overset{js_2}{\Rightarrow} l_3 \overset{jf_2}{\Rightarrow} l_5 \overset{jf_1}{\Rightarrow} l_6$ exists in $\mathcal{A}[\pi']$ but not in $\mathcal{A}[\pi]$. However, note that the reachable locations and executable actions are the same in these two trace sets. □

4.2 Preservation of Locations

We will show in Theorem 2 that IM_{Mrg} preserves locations. This result relies on the forthcoming lemma.

Lemma 4. *Suppose $IM_{Mrg}(\mathcal{A}, \pi)$ terminates with output K_{Mrg}. Then $\pi \models K_{Mrg}$.*

Proof. At the end of IM_{Mrg}, all merged states in S are π-compatible by construction. That is, for all $(l, C) \in S$, we have $\pi \models C{\downarrow}_P$. Since $K_{Mrg} = \bigcap_{(l,C)\in S} C{\downarrow}_P$, then $\pi \models K_{Mrg}$. ■

Theorem 2. *Suppose $IM_{Mrg}(\mathcal{A}, \pi)$ terminates with output K_{Mrg}. Then, for all $\pi' \models K_{Mrg}$, $Locations(\mathcal{A}[\pi]) = Locations(\mathcal{A}[\pi'])$.*

Hence, although the trace set is not preserved by IM_{Mrg}, the set of locations is. As a consequence, the reachability and safety properties (based on locations) that are true in $\mathcal{A}[\pi]$ are also true in $\mathcal{A}[\pi']$.

4.3 Preserving Actions

General Case Although the set of locations is preserved by IM_{Mrg}, the set of actions is not preserved in the general case (in contrast to the reachability analysis with merging). A counterexample is given in [6].

Proposition 1 (Non-preservation of actions). *There exist $\mathcal{A}$, π and π' such that (1) $IM_{Mrg}(\mathcal{A}, \pi)$ terminates with output K_{Mrg}, (2) $\pi' \models K_{Mrg}$, and (3) $Actions(\mathcal{A}[\pi]) \neq Actions(\mathcal{A}[\pi'])$.*

Not all properties are based on actions. Hence IM_{Mrg} is suitable for systems the correctness of which is expressed using the reachability or the non-reachability of locations. Nevertheless, to be able to handle as well systems the correctness of which is expressed using the (non-)reachability of actions, the rest of this section will be devoted to identifying techniques to preserve actions too.

Backward-Deterministic Parametric Timed Automata We identify here a subclass of PTAs for which IM_{Mrg} preserves the set of actions. We restrict the model so that, for any location, at most one action is used on its incoming edges. This restriction can be checked syntactically.

Definition 6 (Backward-determinism). *A PTA is* backward-deterministic *if for all $(l_1, g, a, \rho, l_2), (l_1', g', a', \rho', l_2') \in \rightarrow$, then $l_2 = l_2' \Longrightarrow a = a'$.*

In a backward-deterministic PTA, if a location is reachable, then its incoming action is executed too. Hence the preservation of the locations by IM_{Mrg} implies the preservation of the actions too.

Proposition 2 (Action preservation). *Let $\mathcal{A}$ be a backward-deterministic PTA. Suppose $IM_{Mrg}(\mathcal{A}, \pi)$ terminates with output K_{Mrg}. Then, for all $\pi' \models K_{Mrg}$, $Actions(\mathcal{A}[\pi]) = Actions(\mathcal{A}[\pi'])$.*

Proof. From Theorem 2 and Definition 6. ■

This restriction of backward-determinism may be seen as quite strong in practice. Hence, in the following, in order to preserve the set of actions, we propose to modify the algorithm itself rather than restricting the model.

Improvement of the Inverse Method The non-preservation of the actions by IM_{Mrg} comes from the fact that the states are first merged, and then tested against π-compatibility (see Remark 2). In order to guarantee the action preservation, we propose to first test newly generated states against π-compatibility, and then merge them. Although this modification is only a subtle inversion of two operations in the algorithm, it has consequences on the properties preserved.

We introduce an improved version IM'_{Mrg} of IM_{Mrg} in Algorithm 3, where states are merged after the π-compatibility tests. Technically, the differences with IM_{Mrg} (highlighted using a non-white background) are as follows: (1) the operation to compute the states at the current deepest level i is *Post* instead of *PostM* (lines 9 and 6), and (2) the states are merged *after* the end of the π-incompatibility tests (addition of line 7).

Algorithm 3: Inverse method with merging (variant) $IM'_{Mrg}(\mathcal{A}, \pi)$

```
input  : PTA $\mathcal{A}$ of initial state $s_0$, parameter valuation $\pi$
output: Constraint $K'_{Mrg}$ over the parameters
1 $i \leftarrow 0$ ; $K_c \leftarrow$ true ; $S_{new} \leftarrow \{s_0\}$ ; $S \leftarrow \{\}$
2 while true do
3     while there are $\pi$-incompatible states in $S_{new}$ do
4         Select a $\pi$-incompatible state $(l, C)$ of $S_{new}$ (i.e., s.t. $\pi \not\models C$) ;
5         Select a $\pi$-incompatible $J$ in $C{\downarrow_P}$ (i.e., s.t. $\pi \not\models J$) ;
6         $K_c \leftarrow K_c \wedge \neg J$ ; $S \leftarrow \bigcup_{j=0}^{i-1} PostM^{j}_{\mathcal{A}(K_c)}(\{s_0\})$ ; $S_{new} \leftarrow Post_{\mathcal{A}(K_c)}(S)$ ;
7     $S_{new} \leftarrow Merge(S_{new})$
8     if $S_{new} \sqsubseteq S$ then return $K'_{Mrg} \leftarrow \bigcap_{(l,C) \in S} C{\downarrow_P}$
9     $i \leftarrow i + 1$ ; $S \leftarrow S \cup S_{new}$ ; $S_{new} \leftarrow Post_{\mathcal{A}(K_c)}(S)$
```

Proposition 3. *Suppose $IM(\mathcal{A}, \pi)$, $IM_{Mrg}(\mathcal{A}, \pi)$ and $IM'_{Mrg}(\mathcal{A}, \pi)$ terminate in a deterministic manner with an output K, K_{Mrg} and K'_{Mrg}, respectively.*

Then, $K \subseteq K'_{Mrg} \subseteq K_{Mrg}$

Note that IM'_{Mrg} still does not preserve traces; the situation in Fig. 2 is exactly the same for IM'_{Mrg} as for IM_{Mrg}.

Theorem 3. *Suppose $IM'_{Mrg}(\mathcal{A}, \pi)$ terminates with output K'_{Mrg}. Then, for all $\pi' \models K'_{Mrg}$:*

1. *$Locations(\mathcal{A}[\pi]) = Locations(\mathcal{A}[\pi'])$, and*
2. *$Actions(\mathcal{A}[\pi]) = Actions(\mathcal{A}[\pi'])$.*

Proof. Preservation of locations follows the same reasoning as for Theorem 2. Preservation of actions is guaranteed by construction of IM'_{Mrg} together with the preservation of locations. ■

5 Experimental Validation

We implemented IM'_{Mrg} in IMITATOR [4], in addition to the classical IM. In [13], the main technique for merging two timed constraints C, C' consists in comparing their convex hull H with their union. If the hull and the union are equal (or alternatively, if $(H \setminus C) \setminus C' = \emptyset$, where $\setminus$ is the operation of *convex difference*), then C and C' are mergeable into H. In [13, 14], this technique is specialized to the case where the timed constraints are represented as DBMs. DBMs are not suitable to represent the state space of PTAs; in IMITATOR, polyhedra are used. We implemented the mergeability test using the (costly) operation of convex merging from the Parma Polyhedra Library (PPL) [9].

Table 1 describes experiments comparing the performances and results of IM and IM'_{Mrg}. Column $|X|$ (resp. $|P|$) denotes the number of clocks (resp. parameters) of the PTA. For each algorithm, columns States, Trans., t and Cpl denote the number of states, of transitions the computation time in seconds, and whether the resulting constraint is complete[2], respectively. In the last 3 columns, we compare the results: first, we divide the number of states in IM by the number of states in IM'_{Mrg} and multiply by 100 (hence, a number smaller than 100 denotes an improvement of IM'_{Mrg}); second, we perform the same comparison for the computation time; the last column indicates whether $K = K'_{Mrg}$ or $K \subsetneq K'_{Mrg}$. Experiments were performed on a KUbuntu 12.10 64 bits system running on an Intel Core i7 CPU 2.67GHz with 4 GiB of RAM.

The first 4 models are asynchronous circuits [11, 8]. The SIMOP case study is an industrial networked automation system [8]. The next 5 models are common protocols [12, 17, 8]. The other models are scheduling problems (e.g., [1, 10, 18]). All models are described and available (with sources and binaries of IMITATOR) on IMITATOR's Web page[3].

From Table 1, we see that IM'_{Mrg} has the following advantages. First, the state space is often reduced (actually, in all but 4 models) compared to IM.

[2] Whereas IM and IM'_{Mrg} may be non-complete in general, IMITATOR exploits a sufficient (but non-necessary) condition to detect completeness, when possible.

[3] http://www.lsv.ens-cachan.fr/Software/imitator/merging/

			IM				IM'_{Mrg}				Comparison		
Example	$\|X\|$	$\|P\|$	States	Trans.	t	Cpl	States	Trans.	t	Cpl	States	t	K
AndOr	4	12	11	11	0.052	$\surd$	9	9	0.056	$\surd$	82	108	=
Flip-Flop	5	12	11	10	0.060	$\surd$	9	9	0.057	$\surd$	82	108	=
Latch	8	13	18	17	0.083	?	12	12	0.069	?	67	83	=
SPSMALL	10	26	31	30	0.618	?	31	30	0.613	?	100	99	=
SIMOP	8	7	-	-	OoM	-	172	262	2.52	?	0	0	-
BRP	7	6	429	474	3.50	$\surd$	426	473	4.30	$\surd$	99	123	=
CSMA/CD	3	3	301	462	0.514	$\surd$	300	461	0.574	$\surd$	100	112	=
CSMA/CD’	3	3	13,365	14,271	18.3	$\surd$	13,365	14,271	25.4	$\surd$	100	139	=
RCP	5	6	327	518	0.748	$\surd$	115	186	0.684	$\surd$	35	91	=
WLAN	2	8	-	-	OoM	-	8,430	15,152	2,137	$\surd$	0	0	-
ABT	7	7	63	62	0.344	?	63	62	0.335	?	100	97	=
AM02	3	4	182	215	0.369	$\surd$	53	70	0.112	$\surd$	29	30	$\subsetneq$
BB04	6	7	806	827	28.0	?	141	145	3.15	?	17	11	=
CTC	15	21	1,364	1,363	88.9	$\surd$	215	264	17.6	$\surd$	16	20	=
LA02	3	5	6,290	8,023	751	?	383	533	17.7	$\surd$	6.0	2.4	$\subsetneq$
LPPRC10	4	7	78	102	0.39	?	31	40	0.251	?	40	64	=
M2.4	3	8	1,497	1,844	8.89	$\surd$	119	181	0.374	$\surd$	7.9	4.2	$\subsetneq$

Table 1: Comparison between *IM* and IM'_{Mrg}

This is particularly interesting for the scheduling problems, with a division of the number of states by a factor of up to 16 (LA02). Also note that two case studies could not even be verified without the merging reduction, due to memory exhaustion (“OoM”). Second, the computation time is almost always reduced when the merging reduction indeed reduces the state space, by a factor of up to 42 (LA02). Third, and more surprisingly (considering the cost of the mergeability test), the overhead induced by the mergeability test often does not yield a significant augmentation of the computation time, even when the merging reduction does not reduce the state space at all; the worst case is +39 % (CSMA/CD’), which remains reasonable. Finally, the constraint output by IM'_{Mrg} is weaker (i.e., corresponds to a larger set of valuations) than *IM* for some case studies.

6 Final Remarks

We have shown in this paper that (1) a general technique of state merging in PTAs preserves both the reachability and the non-reachability of actions and locations, (2) the integration of this technique into *IM* often synthesizes a weaker (hence, better) constraint while reducing the computation space, and preserves locations (but neither traces nor actions), and (3) an improved version of IM_{Mrg} preserves not only locations but actions. Experiments with Imitator show that the improved procedure IM'_{Mrg} does not only reduce the state space, but is also often faster than the original procedure *IM*.

As future work, we plan to study the combined integration into *IM* of the general technique of state merging with variants [7] and optimizations [3] of *IM*. Regarding the implementation in Imitator, we aim at studying the replacement of polyhedra with parametric DBMs [17]; furthermore, the (costly) mergeability test should be optimized so as to improve performance. Finally, we also plan to generalize the merging technique to the hybrid setting [15].

References

1. Y. Adbeddaïm and O. Maler. Preemptive job-shop scheduling using stopwatch automata. In *TACAS*, volume 2280 of *LNCS*, pages 113–126. Springer, 2002.
2. R. Alur, T. A. Henzinger, and M. Y. Vardi. Parametric real-time reasoning. In *STOC*, pages 592–601. ACM, 1993.
3. É. André. Dynamic clock elimination in parametric timed automata. In *FSFMA*, volume 31 of *OpenAccess Series in Informatics*, pages 18–31. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, 2013.
4. É. André, L. Fribourg, U. Kühne, and R. Soulat. IMITATOR 2.5: A tool for analyzing robustness in scheduling problems. In *FM*, volume 7436 of *LNCS*, pages 33–36. Springer, 2012.
5. É. André, L. Fribourg, and R. Soulat. Enhancing the inverse method with state merging. In *NFM*, volume 7226 of *LNCS*, pages 100–105. Springer, 2012.
6. É. André, L. Fribourg, and R. Soulat. Merge and conquer: State merging in parametric timed automata (report). Research Report LSV-13-11, Laboratoire Spécification et Vérification, ENS Cachan, France, July 2013. Available at http://www.lsv.ens-cachan.fr/Publis/RAPPORTS_LSV/PDF/rr-lsv-2013-11.pdf.
7. É. André and R. Soulat. Synthesis of timing parameters satisfying safety properties. In *RP*, volume 6945 of *LNCS*, pages 31–44. Springer, 2011.
8. É. André and R. Soulat. *The Inverse Method.* FOCUS Series in Computer Engineering and Information Technology. ISTE Ltd and John Wiley & Sons Inc., 2013.
9. R. Bagnara, P. M. Hill, and E. 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.
10. E. Bini and G. C. Buttazzo. Schedulability analysis of periodic fixed priority systems. *IEEE Transactions on Computers*, 53(11):1462–1473, 2004.
11. R. Clarisó and J. Cortadella. The octahedron abstract domain. *Science of Computer Programming*, 64(1):115–139, 2007.
12. P. R. D'Argenio, J.-P. Katoen, T. C. Ruys, and J. Tretmans. The bounded retransmission protocol must be on time! In *TACAS*, volume 1217 of *LNCS*, pages 416–431. Springer, 1997.
13. A. David. Merging DBMs efficiently. In *NWPT*, pages 54–56. DIKU, University of Copenhagen, 2005.
14. A. David. Uppaal DBM library programmer's reference. http://people.cs.aau.dk/ adavid/UDBM/manual-061023.pdf, 2006.
15. L. Fribourg and U. Kühne. Parametric verification and test coverage for hybrid automata using the inverse method. *International Journal of Foundations of Computer Science*, 24(2):233–249, 2013.
16. L. Fribourg, D. Lesens, P. Moro, and R. Soulat. Robustness analysis for scheduling problems using the inverse method. In *TIME*, pages 73–80. IEEE Computer Society Press, 2012.
17. T. Hune, J. Romijn, M. Stoelinga, and F. W. Vaandrager. Linear parametric model checking of timed automata. *Journal of Logic and Algebraic Programming*, 52-53:183–220, 2002.
18. T. T. H. Le, L. Palopoli, R. Passerone, Y. Ramadian, and A. Cimatti. Parametric analysis of distributed firm real-time systems: A case study. In *ETFA*, pages 1–8. IEEE, 2010.
19. R. B. Salah, M. Bozga, and O. Maler. On interleaving in timed automata. In *CONCUR*, volume 4137 of *LNCS*, pages 465–476. Springer, 2006.