Merge and Conquer: State Merging in Parametric Timed Automata

Étienne André, Laurent Fribourg, Romain Soulat

June 2013

Research report LSV-13-11 (Version 1)

Laboratoire Spécification & Vérification

École Normale Supérieure de Cachan
61, avenue du Président Wilson
94235 Cachan Cedex France

Merge and Conquer: State Merging in Parametric Timed Automata (Report)*

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

[1] Université Paris 13, Sorbonne Paris Cité, LIPN
F-93430, Villetaneuse, France
{first.last}@univ-paris13.fr
[2] LSV, ENS Cachan & CNRS
Cachan, France
{first.last}@lsv.ens-cachan.fr

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, inverse method, 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,

* This is the extended report version of the paper of the same name accepted for publication at ATVA 2013 [AFS13]. In addition to [AFS13], this paper contains slightly more detailed definitions, and the proofs of all results.

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) [AHV93].

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* [AS13] 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 [AFKS12] 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 [SBM06], 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 [Dav05,Dav06], 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. More precisely, if the union of the continuous part of two states is included into their convex hull, then the two states can be replaced with their hull. 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 [Dav05,Dav06]

to the parametric case. This extension is not trivial (for example, the merging technique of [SBM06] does not extend to the parametric case), 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 [BHZ08].

Remark. This paper is an extension of a "work in progress" paper [AFS12]. In contrast to [AFS12], we formally define the merging operation, and characterize it in the general setting of reachability analysis for PTAs. Furthermore, we rewrite a result from [AFS12] 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, and prove simple properties for PTAs. 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 $X = \{x_1, \ldots, x_H\}$ 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 $w : X \to \mathbb{R}_+^H$ assigning a non-negative real value to each clock variable. We will often identify a valuation w with the point $(w(x_1), \ldots, w(x_H))$. Given a constant $d \in \mathbb{R}_+$, we use $X + d$ to denote the set $\{x_1 + d, \ldots, x_H + d\}$. 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 $P = \{p_1, \ldots, p_M\}$ of *parameters*, i.e., unknown constants. A *parameter valuation* π is a function $\pi : P \to \mathbb{R}_+^M$ assigning a nonnegative real value to each parameter. There is a one-to-one correspondence between valuations and points in $(\mathbb{R}_+)^M$. We will often identify a valuation π with the point $(\pi(p_1), \ldots, \pi(p_M))$.

We define here constraints as a set of linear inequalities. 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 the clocks X, over the parameters P, and over the clocks X and the parameters P respectively. In the sequel, the letter J denotes an inequality over the parameters, the letter $D \in \mathcal{L}(X)$ denotes a constraint over the clocks, the letter $K \in \mathcal{L}(P)$ denotes a constraint over the parameters, and the letter $C \in \mathcal{L}(X \cup P)$ denotes a constraint over the clocks and the parameters. 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. We will use these notions synonymously. In this geometric context, a valuation w satisfying a constraint C is equivalent to the polyhedron C containing the corresponding point w, written as $w \in C$.

Given a constraint D over the clocks and 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 constraint C over the clocks and the parameters anda 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 over the clocks and the parameters, C_1 is said to be *included in* C_2, denoted by $C_1 \subseteq C_2$, if $\forall w, \pi : <w, \pi> \models C_1 \implies <w, \pi> \models C_2$. We have that $C_1 = C_2$ if and only if $C_1 \subseteq C_2$ and $C_2 \subseteq C_1$.

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 two constraints K_1 and K_2 over the parameters, we say that K_1 is *included in* K_2, denoted by $K_1 \subseteq K_2$, if $\forall \pi : \pi \models K_1 \implies \pi \models K_2$. We have that $K_1 = K_2$ if and only if $K_1 \subseteq K_2$ and $K_2 \subseteq K_1$. 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. Formally, $C{\downarrow_P} = \{\pi \mid \exists w : <w, \pi>\}$. (Note that $C{\downarrow_P} = \{\pi \mid \pi \models C\}$.)

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. Note that, of course, only clocks can evolve; parameters are unknown constants and therefore remain constant. 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 [AD94] to the parametric case, where parameters can be used within guards and invariants in place of constants [AHV93].

Syntax.

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 $I(l) \in \mathcal{L}(X \cup P)$, and*
- *$\rightarrow$ is a step relation consisting of elements of the form (l, g, a, ρ, l'), also denoted by $l \overset{g,a,\rho}{\rightarrow} l'$, 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 $\mathcal{A}$ (see semantics in Definition 6). 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, as all parameters are valuated, $\mathcal{A}[\pi]$ is a non-parametric timed automaton. We say that $\mathcal{A}$ is *valuated* with π.

Notation. 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 $\mathcal{A}$.

Semantics. The (symbolic) semantics of PTAs relies on the following notion of state.

Definition 3 (State). *Let $\mathcal{A} = (\Sigma, L, l_0, X, P, K, I, \rightarrow)$ be a PTA. A* state *s of $\mathcal{A}$ is a pair (l, C) where $l \in L$ is a location, and $C \in \mathcal{L}(X \cup P)$ its associated constraint.*

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 $<w, \pi> \models C$.

We recall below the notion of π-compatibility [AS13].

Definition 4 (π-compatibility). *A state $s = (l, C)$ of a PTA $\mathcal{A}$ is said to be* compatible with *π or, more simply,* π-compatible *if $\pi \models C$.*

We now define the inclusion of a set of states in another one. Observe that this notion does not refer to the inclusion of each state of the first set into a state of the second set, but to the *equality* of each state of the first set with a state of the second set.

Definition 5 (Set inclusion). *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 $\forall s : s \in S_1 \implies s \in S_2$.*

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 6 (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*

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

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

- *discrete transitions* $(l, C) \xrightarrow{a} (l', C')$ *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* $(l, C) \xrightarrow{d} (l, C')$ *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 $s_i \overset{a_i}{\Rightarrow} s_{i+1} \in \Rightarrow(\mathcal{A})$. Formally:

Definition 7 (Path). *Let* $\mathcal{A}$ *be a PTA. Let* $s_0 \overset{a_0}{\Rightarrow} \ldots \overset{a_{n-1}}{\Rightarrow} s_n$*, such that* $s_i \overset{a_i}{\Rightarrow} s_{i+1} \in \Rightarrow(\mathcal{A})$*, for all* $0 \leq i \leq n-1$*, and* $S_0 = \{s_0\}$.

Then $s_0 \overset{a_0}{\Rightarrow} \ldots \overset{a_{n-1}}{\Rightarrow} s_n$ *is said to be a* path *of* $\mathcal{A}$*. The set of all paths of* $\mathcal{A}$ *is denoted by* $Paths(\mathcal{A})$.

We define *traces* as time-abstract paths.

Definition 8 (Trace). *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* $Traces(\mathcal{A})$.

The *Post* operation computes the successors of a state. Formally, $Post_{\mathcal{A}}(s) = \{s' | \exists a \in \Sigma : (s \overset{a}{\Rightarrow} s') \in \Rightarrow(\mathcal{A})\}$. We define $Post^i_{\mathcal{A}}(s)$ as the set of states reachable from a state s_0 in exactly i steps. Formally:

$$Post^i_{\mathcal{A}}(s_0) = \{s_i | \exists s_1, \ldots, s_{i-1}, \exists a_0, \ldots, a_{i-1} \in \Sigma : \forall 1 \leq k \leq i, (s_{k-1} \overset{a_{k-1}}{\Rightarrow} s_k) \in \Rightarrow(\mathcal{A})\}.$$

The *Post* operation extends to a set S of states as follows: $Post_{\mathcal{A}}(S) = \bigcup_{s \in S} Post_{\mathcal{A}}(s)$. And similarly for $Post^i_{\mathcal{A}}(S)$. We write $Post^*_{\mathcal{A}}(S) = \bigcup_{i \geq 0} Post^i_{\mathcal{A}}(S)$; hence, we denote by $Post^*_{\mathcal{A}}(S)$ the set of all states reachable from S in $\mathcal{A}$.

We will be in particular interested here in computing the set $Post^*_{\mathcal{A}}(\{s_0\})$, where s_0 is the initial state of $\mathcal{A}$. Note that if $Post^{i+1}_{\mathcal{A}}(\{s_0\}) = \emptyset$ (or, more generally, if $Post^{i+1}_{\mathcal{A}}(\{s_0\}) \sqsubseteq \bigcup_{j=0}^{i} Post^j_{\mathcal{A}}(\{s_0\})$), then $Post^*_{\mathcal{A}}(\{s_0\}) = \bigcup_{j=0}^{i} Post^j_{\mathcal{A}}(\{s_0\})$.

Given a PTA $\mathcal{A}$ of initial state s_0, we write $Reach^i(\mathcal{A})$ (resp. $Reach^*(\mathcal{A})$) for $Post^i_{\mathcal{A}}(\{s_0\})$ (resp. $Post^*_{\mathcal{A}}(\{s_0\})$). We also define $Locations(\mathcal{A})$ (resp. $Actions(\mathcal{A})$) as the set of locations (resp. actions) reachable (resp. executable) from the initial state of $\mathcal{A}$. Formally: $Locations(\mathcal{A}) = \bigcup_{(l,C) \in Reach^*(\mathcal{A})} l$. And similarly for actions. 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., [HRSV02,AS13]), and it is clear that the sets $Locations(\mathcal{A})$ and $Actions(\mathcal{A})$ are the same for both the symbolic and concrete semantics. □

2.4 General Results for Parametric Timed Automata

We prove below several general results on constraints and PTAs, that will be used in the forthcoming proofs.

We first prove below a simple result on variable elimination.

Lemma 1 (Distributivity of $\downarrow$ over $\cup$). *Let $C_1, C_2 \in \mathcal{L}(X \cup P)$. Then:*

$$(C_1 \cup C_2)\downarrow_P = C_1\downarrow_P \cup C_2\downarrow_P$$

Proof. $(C_1 \cup C_2)\downarrow_P = \{\pi \mid \pi \models C_1 \cup C_2\}$
$= \{\pi \mid \pi \models C_1 \vee \pi \models C_2\}$
$= \{\pi \mid \pi \models C_1\} \cup \{\pi \mid \pi \models C_2\}$
$= C_1\downarrow_P \cup C_2\downarrow_P$ ■

Lemma 2 (Preservation of inclusion). *Let $\mathcal{A}$ be a PTA, and $(l_1, C_1), (l_2, C_2) \in Reach^*(\mathcal{A})$ with $(l_1, C_1) \stackrel{a}{\Rightarrow} (l_2, C_2)$. Let C_1' be such that $C_1 \subseteq C_1'$.*

Then $\exists C_2'$ such that $C_2 \subseteq C_2'$ and $(l_1, C_1') \stackrel{a}{\Rightarrow} (l_2, C_2')$.

Proof. From the semantics of PTAs, we have that

$$C_2 = \left(\Big(\big(C_1(X) \wedge g(X) \wedge X' = \rho(X)\big)\downarrow_{X' \cup P} \wedge I(l')(X')\Big)_{[X' \leftarrow X]}\right)^{\uparrow} \wedge I(l')(X)$$

and

$$C_2' = \left(\Big(\big(C_1'(X) \wedge g(X) \wedge X' = \rho(X)\big)\downarrow_{X' \cup P} \wedge I(l')(X')\Big)_{[X' \leftarrow X]}\right)^{\uparrow} \wedge I(l')(X)$$

We have $C_1 \subseteq C_1'$; then C_1' (resp. C_2') is obtained from C_1 (resp. C_2) by adding other constraints, which are the same in each expression, hence this preserves inclusion. Then a projection is performed onto $X' \cup P$, which preserves inclusion. Then the expressions are intersected with another constraint ($I(l')(X')$), which preserves the inclusion too. Then the expressions have their (same) set of clocks renamed, and are intersected with the same destination constraint $I(l')(X)$, which again preserves inclusion. Hence $C_2 \subseteq C_2'$. ■

Lemma 3 (Disjunction and Post). *Let $\mathcal{A}$ be a PTA, and $(l, C_1), (l, C_2) \in Reach^*(\mathcal{A})$. Suppose $(l, C_1) \stackrel{a}{\Rightarrow} (l', C_1')$, $(l, C_2) \stackrel{a}{\Rightarrow} (l', C_2')$ and $(l, C) \stackrel{a}{\Rightarrow} (l', C')$. Then:*

$$C = C_1 \cup C_2 \Longrightarrow C' = C_1' \cup C_2'.$$

Proof. The proof follows from the semantics of PTAs.

$C = C_1 \cup C_2$
$\Longrightarrow \big(C \wedge g(X) \wedge X' = \rho(X)\big) = \Big(\big(C_1 \wedge g(X) \wedge X' = \rho(X)\big) \cup \big(C_2 \wedge g(X) \wedge X' = \rho(X)\big)\Big)$ (distributivity of $\wedge$ over $\cup$)

$\Longrightarrow \big(C \wedge g(X) \wedge X' = \rho(X)\big)\downarrow_{X' \cup P} = \Big(\big(C_1 \wedge g(X) \wedge X' = \rho(X)\big)\downarrow_{X' \cup P} \cup \big(C_2 \wedge g(X) \wedge X' = \rho(X)\big)\downarrow_{X' \cup P}\Big)$ (Lemma 1)

$\Longrightarrow \Big(\big(C \wedge g(X) \wedge X' = \rho(X)\big)\downarrow_{X' \cup P} \wedge I(l')(X')\Big) = \Bigg(\Big(\big(C_1 \wedge g(X) \wedge X' = \rho(X)\big)\downarrow_{X' \cup P} \wedge I(l')(X')\Big) \cup \Big(\big(C_2 \wedge g(X) \wedge X' = \rho(X)\big)\downarrow_{X' \cup P} \wedge I(l')(X')\Big)\Bigg)$ (distributivity of $\wedge$ over $\cup$)

$\Longrightarrow \Big(\big(C \wedge g(X) \wedge X' = \rho(X)\big)\downarrow_{X' \cup P} \wedge I(l')(X')\Big)_{[X' \leftarrow X]} = \Bigg(\Big(\big(C_1 \wedge g(X) \wedge X' = \rho(X)\big)\downarrow_{X' \cup P} \wedge I(l')(X')\Big)_{[X' \leftarrow X]} \cup \Big(\big(C_2 \wedge g(X) \wedge X' = \rho(X)\big)\downarrow_{X' \cup P} \wedge I(l')(X')\Big)_{[X' \leftarrow X]}\Bigg)$ (variables renaming)

$\Longrightarrow \Bigg(\Big(\big(C \wedge g(X) \wedge X' = \rho(X)\big)\downarrow_{X' \cup P} \wedge I(l')(X')\Big)_{[X' \leftarrow X]}\Bigg)^{\uparrow} = \Bigg(\Bigg(\Big(\big(C_1 \wedge g(X) \wedge X' = \rho(X)\big)\downarrow_{X' \cup P} \wedge I(l')(X')\Big)_{[X' \leftarrow X]}\Bigg)^{\uparrow} \cup \Bigg(\Big(\big(C_2 \wedge g(X) \wedge X' = \rho(X)\big)\downarrow_{X' \cup P} \wedge I(l')(X')\Big)_{[X' \leftarrow X]}\Bigg)^{\uparrow}\Bigg)$ (Lemma 1 + distributivity of $\wedge$ over $\cup$)

$\Longrightarrow \Bigg(\Big(\big(C \wedge g(X) \wedge X' = \rho(X)\big)\downarrow_{X' \cup P} \wedge I(l')(X')\Big)_{[X' \leftarrow X]}\Bigg)^{\uparrow} = \Bigg(\Bigg(\Big(\big(C_1 \wedge g(X) \wedge X' = \rho(X)\big)\downarrow_{X' \cup P} \wedge I(l')(X')\Big)_{[X' \leftarrow X]}\Bigg)^{\uparrow} \cup \Bigg(\Big(\big(C_2 \wedge g(X) \wedge X' = \rho(X)\big)\downarrow_{X' \cup P} \wedge I(l')(X')\Big)_{[X' \leftarrow X]}\Bigg)^{\uparrow}\Bigg)$ (Lemma 1 + distributivity of $\wedge$ over $\cup$)

$\Longrightarrow \Bigg(\Big(\big(C \wedge g(X) \wedge X' = \rho(X)\big)\downarrow_{X' \cup P} \wedge I(l')(X')\Big)_{[X' \leftarrow X]}\Bigg)^{\uparrow} \wedge I(l')(X) = \Bigg(\Bigg(\Big(\big(C_1 \wedge g(X) \wedge X' = \rho(X)\big)\downarrow_{X' \cup P} \wedge I(l')(X')\Big)_{[X' \leftarrow X]}\Bigg)^{\uparrow} \wedge I(l')(X) \cup \Bigg(\Big(\big(C_2 \wedge g(X) \wedge X' = \rho(X)\big)\downarrow_{X' \cup P} \wedge I(l')(X')\Big)_{[X' \leftarrow X]}\Bigg)^{\uparrow} \wedge I(l')(X)\Bigg)$ (distributivity of $\wedge$ over $\cup$)

$\Longrightarrow C' = C'_1 \cup C'_2$.

■

2.5 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 [AS13].

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

input : PTA $\mathcal{A}$ of initial state s_0, parameter valuation π
output: Constraint K over the parameters

1 $i \leftarrow 0$; $K_c \leftarrow \text{true}$; $S_{new} \leftarrow \{s_0\}$; $S \leftarrow \{\}$
2 **while** true **do**
3 **while** *there are* π*-incompatible states in* S_{new} **do**
4 Select a π-incompatible state (l, C) of S_{new} (i.e., s.t. $\pi \not\models C$) ;
5 Select a π-incompatible J in $C{\downarrow_P}$ (i.e., s.t. $\pi \not\models J$) ;
6 $K_c \leftarrow K_c \wedge \neg J$; $S \leftarrow \bigcup_{j=0}^{i-1} Post^j_{\mathcal{A}(K_c)}(\{s_0\})$; $S_{new} \leftarrow Post_{\mathcal{A}(K_c)}(S)$;
7 **if** $S_{new} \sqsubseteq S$ **then return** $K \leftarrow \bigcap_{(l,C)\in S} C{\downarrow_P}$
8 $i \leftarrow i + 1$; $S \leftarrow S \cup S_{new}$; $S_{new} \leftarrow Post_{\mathcal{A}(K_c)}(S)$

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.[1] 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 the parameters P of the constraints associated with all the reachable states. Otherwise, the exploration goes one step further (line 8).

Recall from [AS13] 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.

[1] Note that we use a slightly different (although equivalent) form compared to [AS13]. The reason for splitting the states between S and S_{new} will be useful when defining IM'_{Mrg} (see Section 4.3).

3 Merging States in Parametric Timed Automata

3.1 Principle

We extend here the notion of merging from [Dav05] to the parametric case. In short, two states are mergeable if they share the same discrete part, and the union of their corresponding constraints is convex.

Definition 9 (Merging). *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* $merge(s_1, s_2)$.

Given a set S of states, $Merge(S)$ 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

$Q \leftarrow S$;
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**
  Choose $(l, C_1) \in Q$, $(l, C_2) \in Q$ such that $C_1 \neq C_2$ and $C_1 \cup C_2$ is convex ;
  $Q \leftarrow Q \setminus \{(l, C_1), (l, C_2)\} \cup \{merge((l, C_1), (l, C_2))\}$;
return Q

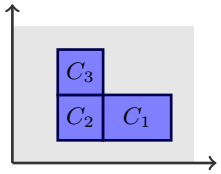


Fig. 1: Non-determinism of merging

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 $\{(l, C_1 \cup C_2), (l, C_3)\}$.

3.2 Merging and Reachability

We define below the semantics of PTAs with merging.

Definition 10 (Semantics of PTAs with merging). *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 10, *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 4. *Let* $\mathcal{A}$ *be a PTA. Then* $Reach^0(\mathcal{A}) = ReachM^0(\mathcal{A})$.

Proof. From Definitions 6 and 10. ■

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 on the two forthcoming lemmas 6 and 8.

First, the following lemma will be used to prove Lemma 6 by induction.

Lemma 5. *Let* $\mathcal{A}$ *be a PTA. Let* $(l_0, C_0) \overset{a_0}{\Rightarrow} \dots \overset{a_{n-1}}{\Rightarrow} (l_n, C_n) \in Paths(\mathcal{A})$, *and* $(l_0, C'_0) \overset{a_0}{\Rightarrow}_{Mrg} \dots \overset{a_{n-1}}{\Rightarrow}_{Mrg} (l_n, C'_n) \in PathsM(\mathcal{A})$ *such that* $C_i \subseteq C'_i$, *for all* $0 \leq i \leq n$.

Suppose $(l_n, C_n) \overset{a_n}{\Rightarrow} (l_{n+1}, C_{n+1}) \in \Rightarrow(\mathcal{A})$. *Then there exists* C'_{n+1} *such that*

1. $(l_n, C'_n) \stackrel{a_n}{\Rightarrow}_{Mrg} (l_{n+1}, C'_{n+1}) \in \Rightarrow_{Mrg}(\mathcal{A})$, *and*
2. $C_{n+1} \subseteq C'_{n+1}$.

Proof. Since $C_n \subseteq C'_n$ and $(l_n, C_n) \stackrel{a_n}{\Rightarrow} (l_{n+1}, C_{n+1}) \in \Rightarrow(\mathcal{A})$, then there exists C''_{n+1} such that $(l_n, C'_n) \stackrel{a_n}{\Rightarrow} (l_{n+1}, C''_{n+1})$ and $C_{n+1} \subseteq C''_{n+1}$ (from Lemma 2). We consider two cases, depending whether (l_{n+1}, C''_{n+1}) is merged or not in $PostM$.

- Suppose (l_{n+1}, C''_{n+1}) is not merged with another state. That is $(l_{n+1}, C''_{n+1}) \in Post_{\mathcal{A}}(ReachM^n(\mathcal{A}))$ and since it is not merged, then (l_{n+1}, C''_{n+1}) belongs to $PostM_{\mathcal{A}}(ReachM^n(\mathcal{A})) = ReachM^{n+1}(\mathcal{A})$. Hence the result holds, with $C'_{n+1} = C''_{n+1}$.
- Suppose (l_{n+1}, C''_{n+1}) is merged with other states, thus creating new state (l_{n+1}, C'_{n+1}), with $C'_{n+1} = C''_{n+1} \cup C'''_{n+1}$. Then $(l_n, C'_n) \stackrel{a_n}{\Rightarrow}_{Mrg} (l_{n+1}, C'_{n+1}) \in \Rightarrow_{Mrg}(\mathcal{A})$, and $C_{n+1} \subseteq C''_{n+1} \subseteq C'_{n+1}$.

■

Now, we can prove that a trace in a non-merged reachability graph has an equivalent trace in a merged reachability graph.

Lemma 6 (Merging and reachability ($\Longrightarrow$)). *Let $\mathcal{A}$ be a PTA. Let $(l_0, C_0) \stackrel{a_0}{\Rightarrow} \ldots \stackrel{a_{n-1}}{\Rightarrow} (l_n, C_n) \in Paths(\mathcal{A})$. Then there exist $C'_1, \ldots, C'_n$ such that:*

1. $(l_0, C_0) \stackrel{a_0}{\Rightarrow}_{Mrg} (l_0, C'_1) \stackrel{a_1}{\Rightarrow}_{Mrg} \ldots \stackrel{a_{n-1}}{\Rightarrow}_{Mrg} (l_n, C'_n) \in PathsM(\mathcal{A})$, *and*
2. $C_i \subseteq C'_i$, *for all* $1 \leq i \leq n$.

Proof. By induction on n.

- **Base case.** For $n = 0$, we have a path reduced to a single state (l_0, C_0). By lemma 4, $(l_0, C_0) \in PathsM(\mathcal{A})$.
- **Induction step.** Suppose that $n \geq 0$ and the property holds for n. Let $(l_0, C_0) \stackrel{a_0}{\Rightarrow} \ldots \stackrel{a_n}{\Rightarrow} (l_{n+1}, C_{n+1}) \in Paths(\mathcal{A})$. By induction hypothesis, there exist $C'_1, \ldots, C'_n$ such that $(l_0, C_0) \stackrel{a_0}{\Rightarrow}_{Mrg} (l_0, C'_1) \stackrel{a_1}{\Rightarrow}_{Mrg} \ldots \stackrel{a_{n-1}}{\Rightarrow}_{Mrg} (l_n, C'_n) \in PathsM(\mathcal{A})$, and $C_i \subseteq C'_i$, for all $0 \leq i \leq n$.
  From Lemma 5, there exists C'_{n+1} such that
  1. $(l_n, C'_n) \stackrel{a_n}{\Rightarrow}_{Mrg} (l_{n+1}, C'_{n+1}) \in \Rightarrow_{Mrg}(\mathcal{A})$, and
  2. $C_{n+1} \subseteq C'_{n+1}$.

  Hence $(l_0, C_0) \stackrel{a_0}{\Rightarrow}_{Mrg} (l_0, C'_1) \stackrel{a_1}{\Rightarrow}_{Mrg} \ldots \stackrel{a_n}{\Rightarrow}_{Mrg} (l_{n+1}, C'_{n+1}) \in PathsM(\mathcal{A})$, and $C_i \subseteq C'_i$, for all $0 \leq i \leq n+1$.

■

We will show in Lemma 8 that the constraint associated to each state in the merged graph is the union of several constraints in the non-merged graph.

The following lemma will be used to prove Lemma 8 by induction. In the proof, we use the following notation: Given a state (l_i, C_i) of $\mathcal{LTS}(\mathcal{A})$, we write $(l_i, C_i) \stackrel{a}{\not\Rightarrow}$ if there exists no (l_{i+1}, C_{i+1}) in $\mathcal{LTS}(\mathcal{A})$ such that $(l_i, C_i) \stackrel{a}{\Rightarrow} (l_{i+1}, C_{i+1}) \in \Rightarrow(\mathcal{A})$.

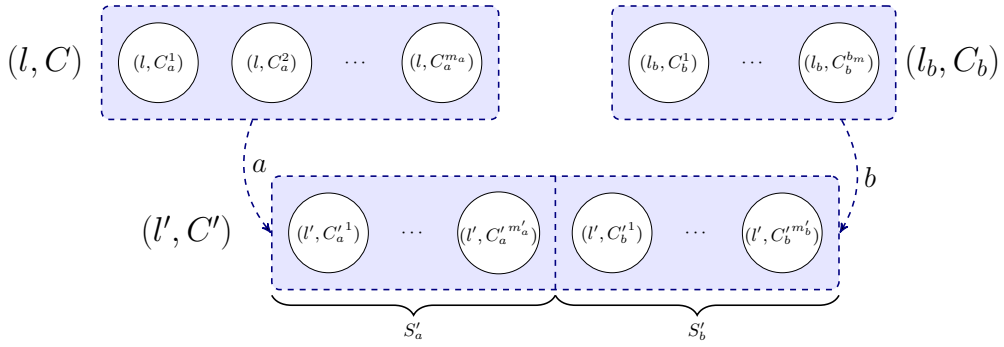


Fig. 2: Context of Lemma 7

Lemma 7. *Let $\mathcal{A}$ be a PTA, let $n \in \mathbb{N}$. Suppose that for all $(l, C) \in ReachM^n(\mathcal{A})$ there exist $m \in \mathbb{N}$ and $(l, C_1), \ldots, (l, C_m) \in Reach^*(\mathcal{A})$ such that*

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

Let $(l, C) \overset{a}{\Rightarrow} (l', C') \in \Rightarrow^n_{Mrg}(\mathcal{A})$. Then there exist $m' \in \mathbb{N}$ and $(l, C'_1), \ldots, (l, C'_l) \in Reach^n(\mathcal{A})$ such that

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

Proof. Suppose $(l, C) \overset{a}{\Rightarrow}_{Mrg} (l', C') \in \Rightarrow^n_{Mrg}(\mathcal{A})$. Suppose (l', C') is the result of merging two states (l', C'_a) and (l', C'_b); hence $C' = C'_a \cup C'_b$. (For sake of simplicity, we assume that only 2 states are merged; the reasoning can be generalized to n states in a straightforward manner.) By definition of the semantics of $\Rightarrow^n_{Mrg}$, there exist C_a and C_b such that $(l_a, C_a) \overset{a}{\Rightarrow} (l', C'_a)$ and $(l_b, C_b) \overset{b}{\Rightarrow} (l', C'_b)$. In our case, we have $(l_a, C_a) = (l, C)$. This is depicted in Fig. 2.

From the hypothesis, there exist $m_a \in \mathbb{N}$ and $(l, C_a^1), \ldots, (l, C_a^{m_a}) \in Reach^*(\mathcal{A})$ such that $C_a = \bigcup_{1 \leq i \leq m_a} C_a^i$, and there exist $m_b \in \mathbb{N}$ and $(l_b, C_b^1), \ldots, (l_b, C_b^{m_b}) \in Reach^*(\mathcal{A})$ such that $C_b = \bigcup_{1 \leq i \leq m_b} C_b^i$.

Let $S_a = \{(l, C_a^i) \in \bigcup_{1 \leq i \leq m_a} (l, C_a^i) | (l, C_a^i) \overset{a}{\Rightarrow} (l', C_a'^{\,i})$, such as the union of all $C_a'^{\,i}$ is convex$\}$. That is, S_a is the set of states composing the merged state (l, C) that have a successor state through action a, and such that the union of the constraint of their successor state is convex. There may be different such sets of states, with a disjoint convex constraint; for the sake of simplicity, we assume there is only one. Since there is only one, then for each state $s \notin S_a$, we have $s \overset{a}{\not\Rightarrow}$. Let S'_a be the set of target states of S_a. That is, $S'_a = \{(l', C') | \exists (l, C) \in S_a, (l, C) \overset{a}{\Rightarrow} (l', C'_i)\}$. Observe that, since all states in S_a belong to $Reach^*(\mathcal{A})$, then all states in S'_a belong to $Reach^*(\mathcal{A})$ too. Also note that, by construction, states in S'_a can be merged to give only one merged state.

By Lemma 3, we have:

$$(l, \bigcup_{(l,C)\in S_a} C) \stackrel{a}{\Rightarrow} (l', \bigcup_{(l,C)\in S_a} \{C' | (l,C) \stackrel{a}{\Rightarrow} (l', C')\})$$

By definition of S'_a, we have:

$$(l, \bigcup_{(l,C)\in S_a} C) \stackrel{a}{\Rightarrow} (l', \bigcup_{(l',C')\in S'_a} C')$$

Since for each state $s \notin S_a$, we have $s \stackrel{a}{\not\Rightarrow}$, then

$$(l, \bigcup_{1\leq i\leq m_a} C_i) \stackrel{a}{\Rightarrow} (l', \bigcup_{(l',C')\in S'_a} C')$$

Since $C_a = \bigcup_{1\leq i\leq m_a} C_i$, then

$$(l, C_a) \stackrel{a}{\Rightarrow} (l', \bigcup_{(l',C')\in S'_a} C')$$

By following a similar reasoning for C_b, we have

$$(l_b, C_b) \stackrel{b}{\Rightarrow} (l', \bigcup_{(l',C')\in S'_b} C')$$

with S'_b a set defined in a similar manner to S'_a. Then, we have

$$C'_a = \bigcup_{(l',C')\in S'_a} C' \text{ and } C'_b = \bigcup_{(l',C')\in S'_b} C'$$

Hence

$$C' = \left(\bigcup_{(l',C')\in S'_a} C' \right) \cup \left(\bigcup_{(l',C')\in S'_b} C' \right)$$

By construction, all states of S'_a and S'_b belong to $Reach^*(\mathcal{A})$. This gives the result. ■

Lemma 8 (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 $(l, C_1), \ldots, (l, C_m) \in Reach^*(\mathcal{A})$ such that*

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

Proof. By induction on n.

– **Base case.** For $n = 0$, we have only one state (l_0, C_0) in $ReachM^n(\mathcal{A})$. By lemma 4, $(l_0, C_0) \in Reach^*(\mathcal{A})$, hence the result holds.

– **Induction step.** Suppose that $n \geq 0$ and the property holds for n. Then, by Lemma 7, the results holds for $n+1$.

■

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 $(l_0, C_0) \overset{a_0}{\Rightarrow} \ldots \overset{a_{n-1}}{\Rightarrow} (l_n, C_n) \in Paths(\mathcal{A})$, there exist $C'_1, \ldots, C'_n$ such that:*
   (a) $(l_0, C_0) \overset{a_0}{\Rightarrow}_{Mrg} (l_0, C'_1) \overset{a_1}{\Rightarrow}_{Mrg} \ldots \overset{a_{n-1}}{\Rightarrow}_{Mrg} (l_n, C'_n) \in PathsM(\mathcal{A})$, 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 $(l, C_1), \ldots, (l, C_m) \in Reach^*(\mathcal{A})$ such that*
$$C = \bigcup_{1 \leq i \leq m} C_i.$$

Proof. From Lemmas 6 and 8. ■

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). *Let $\mathcal{A}$ be a PTA. Then:*

$$Traces(\mathcal{A}) \subseteq TracesM(\mathcal{A}).$$

Proof. From Theorem 1 item 1. ■

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. Then:*

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 $l \overset{a}{\Rightarrow} l' \in Traces(\mathcal{A})$.*

Proof. From Theorem 1. ■

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

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

$$Locations(\mathcal{A}) = LocationsM(\mathcal{A}) \text{ and } Actions(\mathcal{A}) = ActionsM(\mathcal{A}).$$

Proof. From Theorem 1. ■

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 Algorithm 3.

Algorithm 3: Inverse method with merging $IM_{Mrg}(\mathcal{A}, \pi)$

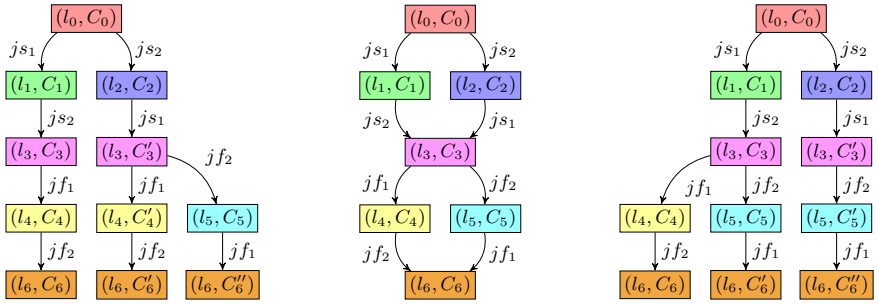
input : PTA $\mathcal{A}$ of initial state s_0, parameter valuation π
output: Constraint K_{Mrg} over the parameters

$i \leftarrow 0$; $K_c \leftarrow \text{true}$; $S_{new} \leftarrow \{s_0\}$; $S \leftarrow \{\}$
while true **do**
  while *there are* π*-incompatible states in* S_{new} **do**
    Select a π-incompatible state (l, C) of S_{new} (i.e., s.t. $\pi \not\models C$) ;
    Select a π-incompatible J in $C{\downarrow_P}$ (i.e., s.t. $\pi \not\models J$) ;
    $K_c \leftarrow K_c \wedge \neg J$; $S \leftarrow \bigcup_{j=0}^{i-1} PostM^{j}_{\mathcal{A}(K_c)}(\{s_0\})$; $S_{new} \leftarrow PostM_{\mathcal{A}(K_c)}(S)$;
  if $S_{new} \sqsubseteq S$ **then return** $K_{Mrg} \leftarrow \bigcap_{(l,C)\in S} C{\downarrow_P}$
  $i \leftarrow i + 1$; $S \leftarrow S \cup S_{new}$; $S_{new} \leftarrow PostM_{\mathcal{A}(K_c)}(S)$

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 typical jobshop example in the setting of parametric schedulability [FLMS12], 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. 3: 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 6) is given in Fig. 3a (in the form of a graph). Applying IM to $\mathcal{A}$ and π gives $K = d_2 > d_1$. From the correctness of IM [AS13], 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 $K_{Mrg} = \text{true}$; the merged trace set is given in Fig. 3b. 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. 3c. 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 9. *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 $\pi' \models K_{Mrg}$, $Locations(\mathcal{A}[\pi]) = Locations(\mathcal{A}[\pi'])$.*

Proof. From Lemma 9, we have that $\pi \models K_{Mrg}$. We will first show that $Locations(\mathcal{A}[\pi]) = Locations(\mathcal{A}(K_c))$, where K_c denotes the value of the current constraint at the end of the algorithm.

$\Longrightarrow$ The fact that $Locations(\mathcal{A}[\pi]) \subseteq Locations(\mathcal{A}(K_c))$ is straightforward. First note that $K_{Mrg} \subseteq K_c$. Hence $\pi \models K_{Mrg}$ implies that $\pi \models K_c$. Hence any location in $\mathcal{A}[\pi]$ is reachable in $\mathcal{A}(K_c)$.

$\Longleftarrow$ We now show that $Locations(\mathcal{A}(K_c)) \subseteq Locations(\mathcal{A}[\pi])$. Let us pick up a state $(l, C) \in S$, where S is the set of states at the end of the algorithm. At the end of the algorithm, $S = ReachM^*(\mathcal{A}(K_c))$. Hence $l \in LocationsM(\mathcal{A}(K_c))$. From Corollary 3, we have that $Locations(\mathcal{A}(K_c)) = LocationsM(\mathcal{A}(K_c))$. Hence $l \in Locations(\mathcal{A}(K_c))$.
We will show that l belongs to $Locations(\mathcal{A}[\pi])$. Recall that $K_{Mrg} = \bigcap_{(l,C)\in S} C{\downarrow_P}$. Since $\pi \models K_{Mrg}$ then, for all $(l, C) \in S$, it holds that $\pi \models C{\downarrow_P}$. hence this is also the case for the (l, C) we picked up. From Theorem 1, there exist $m \in \mathbb{N}$ and $(l, C_1), \ldots, (l, C_m) \in Reach^*(\mathcal{A}(K_c))$ such that

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

Let us pick a constraint C_i such that $\pi \models C_i{\downarrow_P}$. Such a C_i necessarily exists since $\pi \models C{\downarrow_P}$.
Now, we can apply classical results on PTAs: consider a path of $\mathcal{A}(K_c)$ reaching (l, C_i); then, since $\pi \models C_i{\downarrow_P}$, there exists an equivalent path in $\mathcal{A}[\pi]$ (see, e.g., [AS13, Proposition 18]). Hence $l \in Locations(\mathcal{A}[\pi])$.

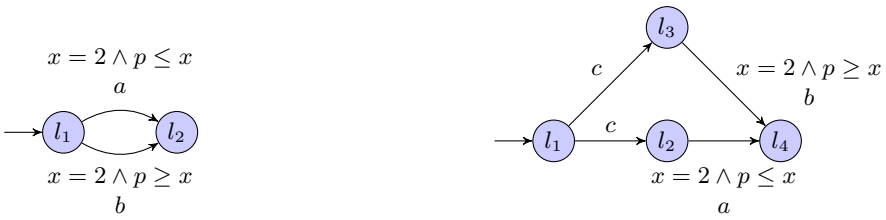
Now, let π' be a valuation such that $\pi' \models K_{Mrg}$. Using the same reasoning, it holds that $Locations(\mathcal{A}[\pi']) = Locations(\mathcal{A}(K_c))$.
Hence $Locations(\mathcal{A}[\pi]) = Locations(\mathcal{A}[\pi'])$. ■

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).

Consider the simple PTA in Fig. 4a. Observe that action a (respectively b) can be taken only if $p \leq 2$ (respectively $p \geq 2$). For a reference valuation π such that $p = 1$, only a can be executed. On the one hand, IM applied to this PTA and to π will output constraint $p < 2$, implying that only a can be executed. On the other hand, IM_{Mrg} will reach two symbolic states $(l_2, p \leq 2 \wedge x \geq 2)$



(a) PTA 1 (b) PTA 2

Fig. 4: Counterexample PTAs showing the non-preservation of actions by IM_{Mrg}

and $(l_2, p \geq 2 \wedge x \geq 2)$, and will merge them into $(l_2, p \geq 0 \wedge x \geq 2)$. Since this state is π-compatible, IM_{Mrg} will output true as a constraint. Then, choosing a valuation π' such as $p = 3$ (note that $\pi' \models IM_{Mrg}(\mathcal{A}, \pi)$), only b can be executed in $\mathcal{A}[\pi']$. Hence the set of actions is not the same as in $\mathcal{A}[\pi]$.

A restriction to the model so that IM_{Mrg} preserves actions could have been that, for each couple of locations l_1 and l_2, all transitions from l_1 to l_2 are always firing the same given action a. But this is not enough: the PTA in Fig. 4b conforms to this restriction, but the same situation as in Fig. 4a happens.

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 11 (Backward-determinism). *A PTA $\mathcal{A}$ is said to be* backward-deterministic *if for all $(l_1, g, a, \rho, l_2), (l_1', g', a', \rho', l_2') \in \rightarrow$, then $l_2 = l_2' \implies a = a'$.*

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 11. ■

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 by the new algorithm.

We introduce an improved version IM'_{Mrg} of IM_{Mrg} in Algorithm 4, 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 4: Inverse method with merging (variant) $IM'_{Mrg}(\mathcal{A}, \pi)$

input : PTA $\mathcal{A}$ of initial state s_0, parameter valuation π
output: Constraint K'_{Mrg} over the parameters

1 $i \leftarrow 0$; $K_c \leftarrow \text{true}$; $S_{new} \leftarrow \{s_0\}$; $S \leftarrow \{\}$
2 **while** true **do**
3 **while** *there are π-incompatible states in S_{new}* **do**
4 Select a π-incompatible state (l, C) of S_{new} (i.e., s.t. $\pi \not\models C$) ;
5 Select a π-incompatible J in $C{\downarrow_P}$ (i.e., s.t. $\pi \not\models J$) ;
6 $K_c \leftarrow K_c \wedge \neg J$; $S \leftarrow \bigcup_{j=0}^{i-1} PostM^j_{\mathcal{A}(K_c)}(\{s_0\})$; $S_{new} \leftarrow \mathit{Post}_{\mathcal{A}(K_c)}(S)$;
7 $S_{new} \leftarrow \mathit{Merge}(S_{new})$
8 **if** $S_{new} \sqsubseteq S$ **then return** $K'_{Mrg} \leftarrow \bigcap_{(l,C)\in S} C{\downarrow_P}$
9 $i \leftarrow i + 1$; $S \leftarrow S \cup S_{new}$; $S_{new} \leftarrow \mathit{Post}_{\mathcal{A}(K_c)}(S)$

We classify below the constraints output by the 3 versions of IM.

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

Proof. The first part of the relationship ($K \subseteq K'_{Mrg}$) comes from two reasons.

1. First, less inequalities are negated. Consider the case of two states s_1 and s_2 (both π-compatible) at a level n, such that each of them has a successor at

level $n+1$, one π-compatible (say s'_1), the other one π-incompatible (say s'_2). In IM, the second state s'_2 would be removed by negating a π-incompatible inequality J. Now, if these two states s_1 and s_2 are merged at level n in IM'_{Mrg} (giving state, say s), their merging could give only one successor s' (equivalent to the union of s'_1 and s'_2); if this successor is π-compatible, the inequality J will not be found, hence not be negated. As a consequence, IM'_{Mrg} will meet less π-incompatible states than IM.

2. The second reason is that the final intersection of the constraints over the parameters associated with all reachable states will be weaker in the case of IM'_{Mrg}. Indeed, since states have been merged, we will consider their union instead of their intersection. Considering the situation in Fig. 2, IM will return the intersection of all local constraints, whereas IM'_{Mrg} will only output $C{\downarrow_P} \cap C_b{\downarrow_P} \cap C'{\downarrow_P}$.

The second part of the relationship ($K'_{Mrg} \subseteq K_{Mrg}$) comes from a similar reasoning. In short, less π-incompatible inequalities will be negated, since the merging is performed before the π-compatibility test. This situation is depicted in Fig. 4a. And similarly, since states are more often merged, their final intersection will yield a weaker constraint too. ■

Note that IM'_{Mrg} still does not preserve traces; the situation in Fig. 3 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 [AFKS12], in addition to the classical IM. In [Dav05], 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 [Dav05,Dav06], 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) [BHZ08].

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., M, t and Cpl

			IM					IM'_{Mrg}					Comparison		
Example	$\|X\|$	$\|P\|$	States	Trans.	t	M	Cpl	States	Trans.	t	M	Cpl	States	t	K
AndOr	4	12	11	11	0.052	1.98	√	9	9	0.056	1.97	√	82	108	=
Flip-Flop	5	12	11	10	0.060	2.75	√	9	9	0.057	2.74	√	82	108	=
Latch	8	13	18	17	0.083	2.65	?	12	12	0.069	2.51	?	67	83	=
SPSMALL	10	26	31	30	0.618	7.746	?	31	30	0.613	8.254	?	100	99	=
SIMOP	8	7	-	-	-	OoM	-	172	262	2.52	35.3	?	0	0	-
BRP	7	6	429	474	3.50	40.0	√	426	473	4.30	41.1	√	99	123	=
CSMA/CD	3	3	301	462	0.514	12.5	√	300	461	0.574	13.2	√	100	112	=
CSMA/CD'	3	3	13,365	14,271	18.3	695	√	13,365	14,271	25.4	739	√	100	139	=
RCP	5	6	327	518	0.748	17.5	√	115	186	0.684	22.3	√	35	91	=
WLAN	2	8	-	-	-	OoM	-	8,430	15,152	2,137	100,502	√	0	0	-
ABT	7	7	63	62	0.344	7.55	?	63	62	0.335	7.44	?	100	97	=
AM02	3	4	182	215	0.369	6.18	√	53	70	0.112	3.49	√	29	30	$\subsetneq$
BB04	6	7	806	827	28.0	66.4	?	141	145	3.15	14.8	?	17	11	=
CTC	15	21	1,364	1,363	88.9	150	√	215	264	17.6	27.6	√	16	20	=
LA02	3	5	6,290	8,023	751	425	?	383	533	17.7	44.6	√	6.0	2.4	$\subsetneq$
LPPRC10	4	7	78	102	0.39	5.83	?	31	40	0.251	3.56	?	40	64	=
M2.4	3	8	1,497	1,844	8.89	95.7	√	119	181	0.374	9.12	√	7.9	4.2	$\subsetneq$

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

denote the number of states, of transitions, a tentative size of the heap given by OCaml, 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 [CC07,AS13]. The SIMOP case study is an industrial networked automation system [AS13]. The next 5 models are common protocols [DKRT97,HRSV02,AS13]. The other models are scheduling problems (e.g., [AM02,BB04,LPP+10]). 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. 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

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

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

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 (namely, scheduling problems).

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 [AS11] and optimizations [And13] of IM. Regarding the implementation in IMITATOR, we aim at studying the replacement of polyhedra with parametric DBMs [HRSV02]; 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 [FK13].

References

AD94. Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994. 5

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 *LNCS*, pages 33–36. Springer, 2012. 2, 22

AFS12. Étienne André, Laurent Fribourg, and Romain Soulat. Enhancing the inverse method with state merging. In *NFM*, volume 7226 of *LNCS*, pages 100–105. Springer, 2012. 3

AFS13. Étienne André, Laurent Fribourg, and Romain Soulat. Merge and conquer: State merging in parametric timed automata. In *ATVA*, LNCS. Springer, October 2013. To appear. 1

AHV93. Rajeev Alur, Thomas A. Henzinger, and Moshe Y. Vardi. Parametric real-time reasoning. In *STOC*, pages 592–601. ACM, 1993. 2, 5

AM02. Yasmina Adbeddaïm and Oded Maler. Preemptive job-shop scheduling using stopwatch automata. In *TACAS*, volume 2280 of *LNCS*, pages 113–126. Springer, 2002. 23

And13. Étienne 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. 24

AS11. Étienne André and Romain Soulat. Synthesis of timing parameters satisfying safety properties. In *RP*, volume 6945 of *LNCS*, pages 31–44. Springer, 2011. 24

AS13. Étienne André and Romain Soulat. *The Inverse Method*. FOCUS Series in Computer Engineering and Information Technology. ISTE Ltd and John Wiley & Sons Inc., 2013. 2, 6, 7, 10, 18, 19, 23

BB04. Enrico Bini and Giorgio C. Buttazzo. Schedulability analysis of periodic fixed priority systems. *IEEE Transactions on Computers*, 53(11):1462–1473, 2004. 23

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. 3, 22

CC07. Robert Clarisó and Jordi Cortadella. The octahedron abstract domain. *Science of Computer Programming*, 64(1):115–139, 2007. 23

Dav05. Alexandre David. Merging DBMs efficiently. In *NWPT*, pages 54–56. DIKU, University of Copenhagen, 2005. 2, 11, 22

Dav06. Alexandre David. Uppaal DBM library programmer's reference. http://people.cs.aau.dk/~adavid/UDBM/manual-061023.pdf, 2006. 2, 22

DKRT97. Pedro R. D'Argenio, Joost-Pieter Katoen, Theo C. Ruys, and Jan Tretmans. The bounded retransmission protocol must be on time! In *TACAS*, volume 1217 of *LNCS*, pages 416–431. Springer, 1997. 23

FK13. Laurent Fribourg and Ulrich 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. 24

FLMS12. Laurent Fribourg, David Lesens, Pierre Moro, and Romain Soulat. Robustness analysis for scheduling problems using the inverse method. In *TIME*, pages 73–80. IEEE Computer Society Press, 2012. 18

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. 7, 23, 24

LPP+10. Thi Thieu Hoa Le, Luigi Palopoli, Roberto Passerone, Yusi Ramadian, and Alessandro Cimatti. Parametric analysis of distributed firm real-time systems: A case study. In *ETFA*, pages 1–8. IEEE, 2010. 23

SBM06. Ramzi Ben Salah, Marius Bozga, and Oded Maler. On interleaving in timed automata. In *CONCUR*, volume 4137 of *LNCS*, pages 465–476. Springer, 2006. 2, 3