Dynamic Clock Elimination in Parametric Timed Automata

Étienne André

Université Paris 13, Sorbonne Paris Cité, LIPN, CNRS, UMR 7030
93430 Villetaneuse, France
Etienne.Andre@univ-paris13.fr

Abstract

The formalism of parametric timed automata provides designers with a formal way to specify and verify real-time concurrent systems where timing requirements are unknown (or parameters). Such models are usually subject to the state space explosion. A popular way to partially reduce the size of the state space is to reduce the number of clock variables. In this work, we present a technique for dynamically eliminating clocks. Experiments using IMITATOR show a diminution of the number of states and of the computation time, and in some cases allow termination of the analysis of models that could not terminate otherwise. More surprisingly, even when the number of clocks remains constant, there is little noticeable overhead in applying the proposed clock elimination.

1998 ACM Subject Classification D.4.7 Real-time systems and embedded systems

Keywords and phrases Verification, Real-time systems, Parameter synthesis, State space reduction, Inverse Method

Digital Object Identifier 10.4230/OASIcs.FSFMA.2013.18

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. Timed automata (TA) are an extension of finite state automata with clocks, i.e., real-valued variables that are compared with constants in guards and invariants, and may be reset along transitions. TA 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 a system. It consists in considering the delays as unknown constants, or *parameters*, and synthesizing constraints on these parameters guaranteeing the system correctness. Parameterizing TA gives parametric timed automata (PTA) [4].

A fundamental problem in the exploration of the reachability space in PTA is to compact as much as possible the generated space of symbolic states. We propose here a state space reduction based on clock elimination.

Related Work

It is well known that the fewer clocks, the more efficient real-time model checking is [11]. Furthermore, a smaller number of clocks may imply a more compact state space: when constraints are represented using arrays and matrices, the fewer clocks, the smaller the constraints are, the more compact the state space is. Formalisms such as (parametric) timed Petri nets [24] or stateful timed CSP [22] have the advantage to dynamically create and discard clocks (or firing times in Petri nets). Hence, clocks only appear in symbolic states when they are actually useful. In contrast, in (parametric) timed automata, according to their standard semantics, clocks must be present in all states.

Still, several works have been proposed to reduce the state space based on the clocks. A well known approach in timed automata is to abstract the value of the clocks as soon as they become larger than the system's largest constant. This technique is implemented in most tools for TA such as UPPAAL [19]; unfortunately, this approach does not apply to PTA, where the constants are replaced with parameters. In [15], two methods are proposed to reduce the number of clocks: (1) the detection of active clocks (the other clocks can be safely eliminated) and (2) the detection of clocks equal to each others (in which case only one such clock can be kept). It is shown that the resulting automaton is bisimilar to the original one, and experiments show large state space reductions. Our work is close to the first method, but extended to the parametric case. Furthermore, the constraints are implemented in [15] in the form of difference bound matrices, where adding and removing clocks is straightforward. In contrast, we use polyhedra where such operations are much more costly; however, experiments show that the overhead in the worst case is still very limited in our setting. Finally, our original motivation was to ensure termination of some systems, which is not necessary in the non-parametric setting since most algorithms rely on symbolic state space partitions guaranteeing termination.

More recently, an approach has been proposed in [10] to avoid the use of global clocks in networks of timed automata, to be analyzed in a distributed setting. Although that approach does not reduce the number of clocks (in contrast to ours), it simplifies the model since less synchronization is needed between the different TA in parallel.

Finally, our work is partially inspired by the parametric extension of stateful timed CSP (PSTCSP) [7]. In PSTCSP, clocks are dynamically created, and discarded when no longer used. Whereas this clock elimination natively belongs to the semantics of PSTCSP, and hence does not require any additional computation, we have to propose algorithms to be able to dynamically eliminate clocks in PTA.

Contribution

We introduce here a technique to eliminate clocks on-the-fly, when it is guaranteed that they will not be read in guards and invariants until their next reset. Our approach is based on a static computation of the location where clocks can be safely eliminated, as well as on a dynamic elimination of these clocks during the analysis.

We implemented our approach in IMITATOR [5], a tool for the synthesis of timing parameters in which operations on constraints rely on the Parma Polyhedra Library [9]. Experiments show a diminution of the number of states and of the computation time, and in some cases allow termination of the analysis of models that could not terminate otherwise. Surprisingly, even when the number of clocks (and hence of states) remains constant, the computation time does not increase, i.e., there is little noticeable overhead in applying the proposed clock elimination.

Outline

We recall preliminaries in Section 2. We define and characterize our dynamic clock elimination technique in Section 3. We present experiments using IMITATOR in Section 4 and conclude in Section 5.

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 $w : X \to \mathbb{R}_+^H$ assigning a non-negative real value to each clock variable. 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 $\pi : P \to \mathbb{R}_+^M$ 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 $(\pi(p_1), \dots, \pi(p_M))$.

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 in a similar manner inequalities over X (resp. P). A *constraint* is a conjunction of inequalities.

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, 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.

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 a constraint C and a clock x, we write $x \in C$ to denote that x is not a free variable in C.

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 denote by $C\backslash_X$ the constraint over the parameters obtained by eliminating its clock variables (e.g., using Fourier-Motzkin [21]). Similarly, 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 $C\backslash_X = C{\downarrow_P}$.

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:

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

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; then the original set of clocks X and d are eliminated; 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 represent the semantics of parametric timed automata.

▶ **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 a finite alternating sequence of states $s_i \in S$ and symbols $a_i \in \Sigma$ of the form $s_0 \overset{a_0}{\Rightarrow} s_1 \overset{a_1}{\Rightarrow} \cdots \overset{a_{m-1}}{\Rightarrow} s_m$, 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 [3] to the parametric case, where parameters can be used within guards and invariants in place of constants [4].

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'), 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 4). For example, in a PTA with two parameters *min* and *max*, one may want to constrain *min* to be always smaller or equal to *max*, in which case K is defined to be $min \leq max$.

Semantics

The (symbolic) semantics of PTA 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.

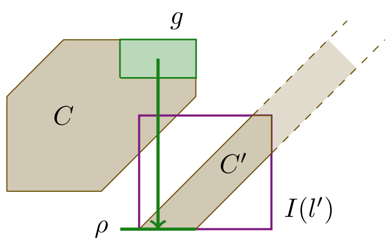


Figure 1 Forward reachability for timed automata.

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

The *initial state* of $\mathcal{A}$ 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 location, and the rest of the expression lets clocks evolve from the same initial value.

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

▶ **Definition 4** (Semantics of PTA). 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

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

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

$$C' = \Big(\big(C(X) \wedge g(X) \wedge X' = \rho(X)\big) \backslash_X \wedge I(l')(X') \Big)_{[X' \leftarrow X]} \quad \textit{and}$$

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

In Figure 1, we present in a graphical way the computation of the successor constraint of a state (l, C). First, C is intersected with the guard g of the transition. Then, the clocks that must be reset by the transition (as in ρ) are projected onto zero. Then, the constraint is intersected with the invariant of the destination location $I(l')$. Time elapsing is then applied. The resulting constraint C' is finally obtained by intersecting again with the invariant of the destination location $I(l')$.

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 $(s_1 \overset{a}{\Rightarrow} s_2) \in \Rightarrow(\mathcal{A})$; and we write s_0 for the (only) state in S_0.

A *path* of $\mathcal{A}$ is a finite alternating sequence of states and actions.

▶ **Definition 5** (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 $0 \leq i \leq n-1$.

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 6** (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 $l_0 \overset{a_0}{\Rightarrow} l_1 \overset{a_1}{\Rightarrow} \cdots \overset{a_{m-1}}{\Rightarrow} l_m$.

Finally, we recall the parallel composition of PTA: N PTA can be composed into a single parametric timed automaton, by performing a product of the N PTA.

▶ **Definition 7.** Let $N \in \mathbb{N}$. For all $1 \leq i \leq N$, let $\mathcal{A}_i = (\Sigma_i, L_i, (l_0)_i, X_i, P_i, K_i, I_i, \rightarrow_i)$ be a PTA. The sets L_i are mutually disjoint. A *network of PTA* is $\mathcal{A} = \mathcal{A}_1 \| \ldots \| \mathcal{A}_N$, where $\|$ is the operator for parallel composition defined in the following way. This network of PTA corresponds to the PTA $\mathcal{A} = (\Sigma, L, l_0, X, P, K, I, \rightarrow)$ where

- $\Sigma = \bigcup_{i=1}^N \Sigma_i$, $L = \Pi_{i=1}^N L_i$, $l_0 = \langle (l_0)_1, \ldots, (l_0)_N \rangle$,
- $X = \bigcup_{i=1}^N X_i$, $P = \bigcup_{i=1}^N P_i$, $K = \bigwedge_{i=1}^N K_i$,
- $I(\langle l_1, \ldots, l_N \rangle) = \bigwedge_{i=1}^N I_i(l_i)$ for all $\langle l_1, \ldots, l_N \rangle \in L$,

and $\rightarrow$ is defined as follows. For all $a \in \Sigma$, let T_a be the subset of indices $i \in 1, \ldots, N$ such that $a \in \Sigma_i$. For all $a \in \Sigma$, for all $\langle l_1, \ldots, l_N \rangle \in L$, for all $\langle l'_1, \ldots, l'_N \rangle \in L$, we have that $(\langle l_1, \ldots, l_N \rangle, g, a, \rho, \langle l'_1, \ldots, l'_N \rangle) \in \rightarrow$ if:

- for all $i \in T_a$, there exist g_i, ρ_i such that $(l_i, g_i, a, \rho_i, l'_i) \in \rightarrow_i$, $g = \bigwedge_{i \in T_a} g_i$, $\rho = \bigcup_{i \in T_a} \rho_i$, and,
- for all $i \notin T_a$, $l'_i = l_i$.

3 On-the-fly Clock Elimination

3.1 Motivation

Consider the PTA depicted in Figure 2. This PTA contains 2 locations, 2 clocks x_1 and x_2, as well as 2 parameters p_1 and p_2. Although the clock x_2 is not used in l_2, its existence will generate an infinite set of states. More precisely, an infinite number of states with a constraint of the form $x_2 = x_1 + i \times p_1$ (with i infinitely growing) will be generated.

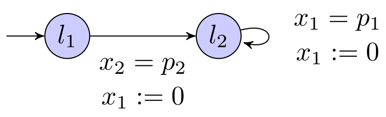


Figure 2 A looping automaton.

This situation is not met in the non-parametric setting. Indeed, it is well known that, once the value of a clock gets larger than the system's largest constant c, this clock value can be safely abstracted to an abstract value "greater than c". Unfortunately, this is not possible in the parametric setting, due to the fact that constants are unknown.

Here, we propose a simple technique based on dynamic clock elimination. We can note that x_2 is "useless" in l_2: indeed, it is not read in any guard, nor reset, and, since l_2 has no successor location except itself, x_2 will not be read in the future. As a consequence, x_2 can be safely discarded or *eliminated* in l_2, so as to ensure termination of the analysis.

Recall that this situation is not met in formalisms such as the parametric extension of stateful timed CSP [7]. Indeed, in this formalism, clocks are dynamically created, and discarded when no longer used.

3.2 General Approach

We propose here to eliminate useless clocks on-the-fly, i.e., during the analysis. By useless, we mean clocks that will not be useful in the future (i.e., not read in guards and invariants), until their next reset. Technically, detecting useless clocks would require to explore the system, and check whether a given clock will be used (i.e., read in a guard or in an invariant) in the future. Unfortunately, this would not be interesting to do in practice since this would require to analyze the whole system, which we want to avoid. Hence, one must accept to possibly exhibit an under-approximation of the set of useless clocks, in order to find a trade-off between efficiency and accuracy.

In this work, we propose the following technique. First, we detect the useless clocks in a static manner; hence, we construct prior to the analysis a table associating each location with the list of the clocks useless in this location. During the analysis, it is sufficient to check this table in order to know which clocks are useless.

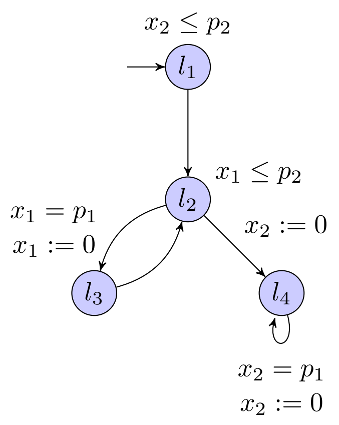
Second, we consider only *local* clocks, i.e., used in a single PTA. (Recall that the PTA analyzed can be made of a network of N PTA in parallel.) This requirement is motivated by obvious efficiency reasons: exploring each PTA in an independent manner is by far more efficient than exploring the composition of several PTA, required to detect the locations in which global clocks (used by several PTA) can be safely discarded. Note that, in all case studies we considered, all clocks were always local. Extending our work to the case of global clocks is discussed in Section 5.

3.3 Static Computation of the Useless Clocks per Location

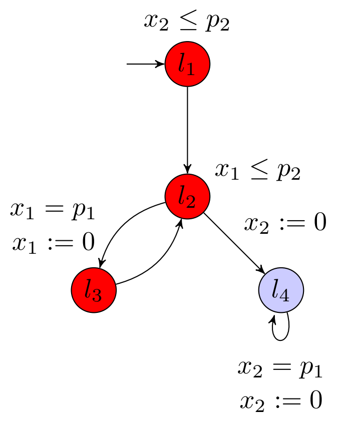
We introduce in Algorithm 1 an algorithm $useless(\mathcal{A}, x)$, that computes in a static manner the set of locations where a clock x is useless. This algorithm takes as input a PTA $\mathcal{A}$ and a clock x, and outputs the list of locations in $\mathcal{A}$ where x is useless.

```
Algorithm 1: $useless(\mathcal{A}, x)$
   input  : PTA $\mathcal{A}$, clock $x$
   output : List of locations where $x$ is unnecessary
1  $Marked \leftarrow \{l | \exists l', a, g, \rho : (l, a, g, \rho, l') \in \rightarrow \wedge x \in g\} \cup \{l | x \in I(l)\}$
2  $Waiting \leftarrow Marked$
3  while $Waiting \neq \emptyset$ do
4      pick $l'$ from $Waiting$
5      foreach $(l, a, g, \rho, l') \in \rightarrow$ do
6          if $x \notin \rho$ then
7              if $l \notin Marked$ then
8                  $Marked \leftarrow Marked \cup \{l\}$
9                  $Waiting \leftarrow Waiting \cup \{l\}$
10 return $L \setminus Marked$
```

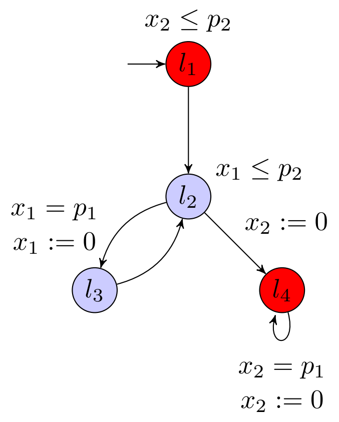
The algorithm makes use of a set of waiting locations ("*Waiting*") and a set of marked locations ("*Marked*"); this latter set corresponds to the locations where x is actually useful. Lines 1– 2 initialize the value of *Waiting* and *Marked* to the set of locations that are either predecessors of a guard involving x or have an invariant involving x. Then, it proceeds by coloring locations in a backward manner, starting from *Marked.* As long as the set of waiting locations is not empty, the algorithm picks a location l' from this set (line 4); then, for each transition whose destination location is l', the algorithm checks whether the clock x is reset along the transition (line 6). If not, and if the transition source l is not marked yet, then l is added both to the set of marked locations and to the waiting set (lines 8–9). The algorithm finally returns the set of locations in $\mathcal{A}$ that are not marked (line 10).



(a) A toy PTA $\mathcal{A}$



(b) Locations marked in $\mathit{useless}(\mathcal{A}, x_1)$



(c) Locations marked in $\mathit{useless}(\mathcal{A}, x_2)$

Figure 3 Static computation of the useless clocks: an example.

Let us apply Algorithm 1 to the simple PTA in Figure 3a and to clock x_1. Initially, $\mathit{Marked} = \mathit{Waiting} = \{l_1, l_2\}$. Let us pick l_1 from *Waiting*. Since l_1 has no predecessor, no action is performed. Let us pick l_2 from *Waiting*; l_2 has two predecessors l_1 and l_3. For l_1, $x_1 \notin \rho$, but $l_1 \in \mathit{Marked}$, hence again no action is performed. For l_3, $x_1 \notin \rho$, and $l_3 \notin \mathit{Marked}$, hence we add l_3 to both *Marked* and *Waiting*. We now pick l_3 from *Waiting*; l_3 has one predecessor l_1, already in *Marked*. The *Waiting* set is now empty, and the algorithm has marked l_1, l_2 and l_3, as showed in Figure 3b; the non-marked locations are returned, viz., $\{l_4\}$.

The result of the application of Algorithm 1 to $\mathcal{A}$ and x_2 is given in Figure 3c. The locations for which x_2 is useless are l_2 and l_3.

In the case of a network of PTA (see Definition 7), the list of useless clocks in a global location is the union, for each of the PTA in parallel, of the clocks useless in the local location for this PTA.

▶ Remark. An alternative and equivalent way to present Algorithm 1 is to use the following recursively defined function (given in a functional programming-like syntax), that decides whether a clock is useless in a given location.

```
let uselessInLoc (x, l) =
    x notin I(l)
    and
    foreach (l, a, g, rho, l' ) in steps then
        x notin g
        and ( x in rho  or  uselessInLoc(x, l') )
```

3.4 Dynamic Elimination of the Clocks in Practice

Following the static computation of the locations in which each clock is useless, we can now eliminate the clocks on-the-fly during a reachability analysis. More precisely, this is performed *after* computing the constraint associated with a new state; once this constraint has been computed, useless clocks are eliminated. This elimination is a variable elimination à la Fourier-Motzkin [21], so as not to modify the relationship between the other clocks and parameters.

Algorithm 2: Computation of a new state in IMITATOR.

```
    input  : PTA $\mathcal{A}$, state $(l, C)$, transition $(l, a, g, \rho, l')$
    output : New state $(l', C')$
1 $C' \leftarrow C \wedge g$
2 $C' \leftarrow \rho(C')$
3 $C' \leftarrow C' \wedge I(l')$
4 $C' \leftarrow C'^{\uparrow}$
5 $C' \leftarrow \mathit{Eliminate}(C')$
6 return $(l', C')$
```

We give in Algorithm 2 a simplified[1] version of the computation of the successor state (l', C'), generated from a source state (l, C) via transition (l, a, g, ρ, l'), as implemented in IMITATOR [5]. The addition of the clock elimination is highlighted (line 5); in this expression, $\mathit{Eliminate}(C')$ denotes the elimination of the clocks useless in the destination location l', as computed by Algorithm 1 for each clock. In IMITATOR, the variable elimination is performed using the dedicated function of the Parma Polyhedra Library [9].

3.5 Characterization

In this section, we show that applying the dynamic clock elimination during a reachability analysis preserves parametric analyses, as well as the satisfiability of linear-time properties.

Let us denote by $U(l)$ the list of clocks useless in a given location l; the result of this function can be computed by applying Algorithm 1 for each clock.

We define below the semantics of PTA under dynamic clock elimination.

[1] IMITATOR also features discrete variables, as well as stopwatches; these features are beyond the scope of this paper, and are discarded here. Furthermore, after each modification of C', a satisfiability test is performed to check whether (l', C') is valid new state; if not, it is discarded (using an exception mechanism).

▶ **Definition 8.** Let $\mathcal{A} = (\Sigma, L, l_0, X, P, K, I, \rightarrow)$ be a PTA. The *semantics of $\mathcal{A}$ under dynamic clock elimination* is $\mathcal{LTS}_{dyn}(\mathcal{A}) = (\Sigma, S, S_0, \Rightarrow_{dyn})$ where

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

and a transition $(l, C) \Rightarrow^{a}_{dyn} (l', C')$ belongs to $\Rightarrow_{dyn}$ if $\exists C'' : (l, C) \overset{a}{\Rightarrow} (l', C'')$, and $C' = C''\backslash_{U(l')}$.

Hence, a transition in the semantics under dynamic clock elimination corresponds to a transition conform to the standard semantics of PTA (i.e., $(l, C) \overset{a}{\Rightarrow} (l', C'')$), followed by the elimination of the clocks useless in l' (i.e., $C' = C''\backslash_{U(l')}$).

We denote by $\mathit{Paths}_{dyn}(\mathcal{A})$ the set of paths of $\mathcal{A}$ computed using the semantics of $\mathcal{A}$ under dynamic clock elimination.

We characterize below the effect of dynamically eliminating clocks while performing a reachability analysis.

▶ **Theorem 9.** *Let $\mathcal{A}$ be a PTA. Then:*

$\Rightarrow$ *Let $(l_0, C_0) \overset{a_0}{\Rightarrow} \cdots \overset{a_{m-1}}{\Rightarrow} (l_m, C_m)$ be a path in $\mathit{Paths}(\mathcal{A})$. Then there exist C'_i, $0 \leq i \leq m$ such that $(l_0, C'_0) \overset{a_0}{\Rightarrow} \cdots \overset{a_{m-1}}{\Rightarrow} (l_m, C'_m)$ is a path in $\mathit{Paths}_{dyn}(\mathcal{A})$, with $C'_i = C_i\backslash_{U(l_i)}$ for $0 \leq i \leq m$.*

$\Leftarrow$ *Conversely, let $(l_0, C'_0) \overset{a_0}{\Rightarrow} \cdots \overset{a_{m-1}}{\Rightarrow} (l_m, C'_m)$ be a path in $\mathit{Paths}_{dyn}(\mathcal{A})$. Then there exist C_i, $0 \leq i \leq m$ such that $(l_0, C_0) \overset{a_0}{\Rightarrow} \cdots \overset{a_{m-1}}{\Rightarrow} (l_m, C_m)$ is a path in $\mathit{Paths}(\mathcal{A})$, with $C'_i = C_i\backslash_{U(l_i)}$ for $0 \leq i \leq m$.*

Proof (sketch). The first part ($\Rightarrow$) is obtained by induction on the length of the paths. Suppose the result holds for i, and let us prove it for $i + 1$. Consider $(l_i, C_i) \overset{a_i}{\Rightarrow} (l_{i+1}, C_{i+1})$. From the induction hypothesis, there exists (l_i, C'_i) with $C'_i = C_i\backslash_{U(l_i)}$. Since $C_i \subseteq C'_i$, then there exists C''_{i+1} such that $(l_i, C'_i) \overset{a_i}{\Rightarrow} (l_{i+1}, C''_{i+1})$. The fact that $C'_{i+1} = C_{i+1}\backslash_{U(l_{i+1})}$ can be proved by showing that the operations in the two items of Definition 4 preserve this equality. Note that this holds only because the clocks in $U(l_i)$ and $U(l_{i+1})$ are not used in the invariants, guards and resets in the definition.

The second part ($\Leftarrow$) is obtained using a similar reasoning.

◀

Basically, Theorem 9 states that each path in $\mathit{Paths}(\mathcal{A})$ has an equivalent in $\mathit{Paths}_{dyn}(\mathcal{A})$, and conversely. Furthermore, in each state, the relationship between all parameters and all clocks (except the clocks useless in this state) is the same in both semantics; this comes from the fact that $C'_i = C_i\backslash_{U(l_i)}$.

We exhibit below two corollaries of Theorem 9. The first corollary states that the projection of the constraints associated to the states of a path in both the standard semantics and the semantics under dynamic clock elimination are the same. Hence, the clock elimination is suitable to perform parametric model checking based on paths.

▶ **Corollary 10.** *Let $\mathcal{A}$ be a PTA. Let $(l_0, C'_0) \overset{a_0}{\Rightarrow} \cdots \overset{a_{m-1}}{\Rightarrow} (l_m, C'_m)$ be a path in $\mathit{Paths}_{dyn}(\mathcal{A})$, and let $(l_0, C_0) \overset{a_0}{\Rightarrow} \cdots \overset{a_{m-1}}{\Rightarrow} (l_m, C_m)$ be its equivalent path in $\mathit{Paths}(\mathcal{A})$.*

Then $C_i{\downarrow_P} = C'_i{\downarrow_P}$, for all $0 \leq i \leq m$.

Proof. Since $C'_i = C_i\backslash_{U(l_i)}$ then $C'_i\backslash_X = C_i\backslash_X$, hence $C'_i{\downarrow_P} = C_i{\downarrow_P}$. ◀

The second corollary states that the dynamic clock elimination preserves linear time properties. Given a linear-time property, we denote by $\varphi \models \mathit{Paths}(\mathcal{A})$ the fact that all paths of $\mathcal{A}$ satisfy φ (and similarly for Paths_{dyn}).

▶ **Corollary 11.** *Let $\mathcal{A}$ be a PTA. Let φ be a linear-time property.*
Then $\varphi \models \mathit{Paths}(\mathcal{A})$ if and only if $\varphi \models \mathit{Paths}_{dyn}(\mathcal{A})$.

Proof. Since each path in $\mathit{Paths}(\mathcal{A})$ has an equivalent path in $\mathit{Paths}_{dyn}(\mathcal{A})$ and vice-versa, the sets of traces are equal. Hence the linear-time properties satisfied are equal. ◀

4 Experimental Validation

This clock elimination technique has been implemented in IMITATOR [5] (since version 2.6.1) as an optional feature (option `-dynamic-elimination`). We compare the efficiency of our dynamic clock elimination technique on the inverse method *IM* [8]. This algorithm 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., guaranteeing that the same linear-time properties are satisfied. The two algorithms compared are (1) *IM* and (2) IM_{dyn}, i.e., *IM* where useless clocks are eliminated on-the-fly using the algorithms of Section 3. Note that, since *IM* relies on the exploration of the parametric state space (after eliminating all clocks), from Corollary 10, the result of both algorithms will be the same.

Table 1 compares the performances and results of *IM* and IM_{dyn}. Columns $|X|$ and $|P|$ denote the number of clocks and parameters of the PTA, respectively. For each algorithm, columns $|S|$, $|T|$ and t denote the number of states, of transitions and the computation time in seconds, respectively. In the last 2 columns, we compare the results: first, we divide the number of states in *IM* by the number of states in IM_{dyn} and multiply by 100 (hence, a number smaller than 100 denotes an improvement of the clock elimination); second, we perform the same comparison for the computation time. Experiments were performed on a KUbuntu 13.04 64 bits system running on an Intel Core i7 CPU 2.67GHz with 4 GiB of RAM.

Table 1 Experiments.

			IM			IM_{dyn}			Comparison	
Example	$\|X\|$	$\|P\|$	$\|S\|$	$\|T\|$	t	$\|S\|$	$\|T\|$	t	$\|S\|$	t
Figure 2	2	2	-	-	loop	2	2	0.007	0	0
Figure 3	2	2	-	-	loop	6	8	0.006	0	0
AndOr	4	12	11	11	0.047	11	11	0.050	100	106
SPSMALL	10	26	31	30	0.580	31	30	0.584	100	101
Train	3	6	78	94	0.100	61	76	0.072	78	72
BRP	7	6	429	474	3.50	429	474	3.21	100	92
CSMA/CD$_6$	3	3	13,365	14,271	19.6	13,365	14,271	19.5	100	99
RCP	5	6	327	518	0.68	181	282	0.41	55	60
AAM06	3	8	1,497	1,844	8.28	768	997	2.92	51	35
AM02	3	4	182	215	0.392	182	215	0.386	100	98
BB04	6	7	806	827	25.4	806	827	27.2	100	107
CTC	15	21	1,364	1,363	83.4	201	291	2.52	15	3.0
LA02	3	5	6,290	8,023	710	4,932	7,154	473	78	67
LPPRC10	4	7	78	102	0.375	78	102	0.395	100	105

Description of the Models

The first 2 models are the looping PTA in Figure 2 and Figure 3a. The next 2 models are asynchronous circuits [13, 8]. The next case study is a classical train–gate–controller from [4]. The next 3 models are common protocols [14, 18, 17]. The other models are scheduling problems [1, 2, 12, 23, 20]. All models are described and available (with sources and binaries of IMITATOR) on IMITATOR's Web page[2].

Interpretation of the Experiments

Let us comment the experiments in Table 1. Although only the 2 toy models are such that only IM_{dyn} can analyze them whereas IM loops, the optimization of IM_{dyn} also leads to state space reductions in many other models. These state space reductions come from the fact that useless clocks may in general lead to the creation of many similar states, only different with respect to the (generally increasing) value of these clocks; when the useless clocks are eliminated, all these similar states are replaced with only one state.

The use of the optimized version IM_{dyn} has the following advantages. First, the state space is often reduced compared to the classical IM (without clock elimination). Although the dynamic elimination of clocks does not seem to bring anything in the case of hardware verification, it seems much more interesting for protocols and scheduling problems. This is particularly interesting for the scheduling problems, with a division of the number of states by a factor of up to 6 (CTC). Second, the computation time is always reduced when the dynamic clock elimination indeed reduces the state space, by a factor of up to 33 (CTC). Third, and more surprisingly, the overhead brought by the dynamic elimination does not yield a significant augmentation of the computation time, even when the clock elimination does not reduce the state space at all; the worst case is +7 % (BB04), which remains very reasonable. These experiments encourage us to consider to set this optimization as default in IMITATOR.

Finally, in some cases (BRP, CSMA/CD, AM02), the computation time is smaller in the case of dynamic clock elimination, despite the absence of state space reduction – which is surprising. This may be due to little variations of the processor. This might also be explained by the fact that, even when states are not merged, the computation of the successor states may be more efficient when the constraints are smaller (i.e., have fewer clocks).

5 Conclusion

We introduced here a state space reduction technique based on an on-the-fly elimination of unnecessary clocks in parametric timed automata. This technique has the following advantages: (1) some models that include loops preventing termination may terminate; (2) the relationship between the remaining clocks and parameters is preserved, which makes it suitable for many (parametric) model checking algorithms; (3) the application of this technique to the inverse method (implemented in IMITATOR) shows interesting state space reductions without adding any significant overhead in terms of computation time.

[2] `http://www.lsv.ens-cachan.fr/Software/imitator/dynamic/`

Future Work

So far, we considered only local clocks, i.e., clocks used in only one of the different PTA in parallel. Considering global clocks (i.e., used in most of the PTA describing the model) would be interesting. In order to avoid the static composition of all PTA prior to the analysis, this would require more complex algorithms than our current detection of the locations where a clock can be discarded. An alternative is to combine our technique with the technique of global clock elimination introduced in [10], if this latter technique can be extended to the parametric setting.

A future extension consists in extending the second algorithm of [15], i.e., to dynamically eliminate clocks that are *equal* to another clock. Although simple in theory, this optimization would require some operations on the constraints that may turn more complex and time-consuming in the parametric setting (using polyhedra) than in the non-parametric setting (using difference bound matrices).

We aim at extending this work to the case of hybrid systems, where clocks are generalized to variables with (in general) arbitrary rates. This could then be applied to the inverse method generalized to hybrid systems [16].

We are also interested in studying the optimization presented here with the (more restrictive) state space reduction based on convex merging recently proposed in [6].

Acknowledgement

I am grateful to an anonymous reviewer for his/her useful comments.

References

1 Yasmina Adbeddaïm, Eugene Asarin, and Oded Maler. Scheduling with timed automata. *Theoretical Computer Science*, 354(2):272–300, 2006.

2 Yasmina Adbeddaïm and Oded Maler. Preemptive job-shop scheduling using stopwatch automata. In *TACAS*, volume 2280 of *Lecture Notes in Computer Science*, pages 113–126. Springer, 2002.

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

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

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

6 Étienne André, Laurent Fribourg, and Romain Soulat. Merge and conquer: State merging in parametric timed automata. In *ATVA*, Lecture Notes in Computer Science. Springer, 2013. To appear.

7 Étienne André, Yang Liu, Jun Sun, and Jin Song Dong. Parameter synthesis for hierarchical concurrent real-time systems. In *ICECCS*, pages 253–262. IEEE Computer Society, 2012.

8 Étienne André and Romain Soulat. *The Inverse Method*. FOCUS Series in Computer Engineering and Information Technology. ISTE Ltd and John Wiley & Sons Inc., 2013.

9 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.

10 Sandie Balaguer and Thomas Chatain. Avoiding shared clocks in networks of timed automata. In *CONCUR*, volume 7454 of *Lecture Notes in Computer Science*, pages 100–114. Springer, 2012.

11 Johan Bengtsson and Wang Yi. Timed automata: Semantics, algorithms and tools. In *Lectures on Concurrency and Petri Nets*, volume 3098 of *Lecture Notes in Computer Science*, pages 87–124. Springer, 2003.

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

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

14 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 *Lecture Notes in Computer Science*, pages 416–431. Springer, 1997.

15 Conrado Daws and Sergio Yovine. Reducing the number of clock variables of timed automata. In *RTSS*, pages 73–81. IEEE Computer Society, 1996.

16 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.

17 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.

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

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

20 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.

21 Alexander Schrijver. *Theory of linear and integer programming.* John Wiley & Sons, Inc., 1986.

22 Jun Sun, Yang Liu, Jin Song Dong, Yan Liu, Ling Shi, and Étienne André. Modeling and verifying hierarchical real-time systems using Stateful Timed CSP. *ACM Transactions on Software Engineering and Methodology*, 22(1):3.1–3.29, 2013.

23 Naoyuki Tamura. CSP2SAT: JSS benchmark results. `http://bach.istc.kobe-u.ac.jp/csp2sat/jss/`, 2007.

24 Louis-Marie Traonouez, Didier Lime, and Olivier H. Roux. Parametric model-checking of stopwatch Petri nets. *Journal of Universal Computer Science*, 15(17):3273–3304, 2009.