BRICS RS-02-35 P. Bouyer: Timed Automata May Cause Some Troubles

BRICS

Basic Research in Computer Science

Timed Automata May Cause Some Troubles

Patricia Bouyer

BRICS Report Series **RS-02-35**

ISSN 0909-0878 **August 2002**

See back inner page for a list of recent BRICS Report Series publications.
Copies may be obtained by contacting:

BRICS
Department of Computer Science
University of Aarhus
Ny Munkegade, building 540
DK–8000 Aarhus C
Denmark

Telephone: +45 8942 3360
Telefax: +45 8942 3255
Internet: BRICS@brics.dk

BRICS publications are in general accessible through the World Wide Web and anonymous FTP through these URLs:

http://www.brics.dk
ftp://ftp.brics.dk
This document in subdirectory RS/02/35/

Timed Automata May Cause Some Troubles

Patricia Bouyer*

LSV – CNRS UMR 8643 & ENS de Cachan
61, Av. du Président Wilson
94235 Cachan Cedex – France

BRICS†– Aalborg University
Fredrik Bajers Vej 7E
9220 Aalborg Ø – Denmark

e-mail: `bouyer@cs.auc.dk`

Abstract

Timed automata are a widely studied model. Its decidability has been proved using the so-called region automaton construction. This construction provides a correct abstraction for the behaviours of timed automata, but it does not support a natural implementation and, in practice, algorithms based on the notion of zones are implemented using adapted data structures like DBMs. When we focus on forward analysis algorithms, the exact computation of all the successors of the initial configurations does not always terminate. Thus, some abstractions are often used to ensure termination, among which, a widening operator on zones.

In this paper, we study in details this widening operator and the forward analysis algorithm that uses it. This algorithm is most used and implemented in tools like KRONOS and UPPAAL. One of our main results is that it is hopeless to find a forward analysis algorithm, that uses such a widening operator, and which is correct. This goes really against what one could think. We then study in details this algorithm in the more general framework of updatable timed automata, a model which has been introduced as a natural syntactic extension of classical timed automata. We describe subclasses of this model for which a correct widening operator can be found.

1 Introduction

Real-Time Systems. Since their introduction by Alur and Dill in [AD90, AD94], timed automata are one of the most studied models for real-time systems. Numerous works have been devoted to the "theoretical" comprehension of timed automata: determinization [AFH94], minimization [ACD+92],

*This work has been partly supported by the french project RNRT "Calife".

†Basic Research in Computer Science (`www.brics.dk`), funded by the Danish National Research Foundation.

power of clocks [ACH94, HKWT95], power of ε-transitions [BDGP98], extensions of the model [DZ98, HRS98, CG00, BFH$^+$01], logical characterizations [Wil94, HRS98],... have in particular been investigated. Practical aspects of the model have also been studied and several model-checkers are now available (HyTech[1] [HHWT97], Kronos[2] [DOTY96], Uppaal[3] [LPY97]). Timed automata afford to modelize many real-time systems and the existing model-checkers have allowed to verify a lot of industrial case studies (see the web pages of the tools or, for example, [HSLL97, TY98]).

Implementation of Timed Automata. The decidability of the timed automata model has been proved by Alur and Dill in [AD90, AD94]. It is based on the construction of the so-called region automaton: it abstracts finitely and in a correct way the behaviours of timed automata. However, in practice, such a construction is not implemented, because it does not support a natural implementation, but algorithms glancing on-the-fly through the automaton are preferred. These algorithms are based on the notion of zones and can be efficiently implemented using data structures like DBMs [Dil89] and CDDs [BLP$^+$99]. Among these algorithms are the forward analysis algorithms that compute all the reachable configurations. However, the exact forward computation does not always terminate. Abstractions have been proposed to avoid this termination problems (see for example [DT98]) and they are implemented in tools like Kronos and Uppaal.

Our contributions. In this paper, we study in details one of the abstractions proposed in [DT98]. It consists in a widening operator that is applied to zones to enforce the termination of the forward searching in the timed automaton. The most important result is that, against what one can think, there is no way to find a correct forward analysis algorithm for timed automata that uses this widening operator.
We then consider the more general framework of updatable timed automata, as defined in [BDFP00a]. This model is a syntactic extension of the classical timed automata coming from a model built for a communication protocol in [BF99] and its decidability has been precisely settled in [BDFP00a]. We then describe several subclasses of this model for which we can find a correct widening operator and thus a correct forward analysis algorithm. These subclasses contain in particular all the timed automata that do not use comparisons between clocks. From this study, it appears that problems in the correctness of the widening operator come only from the nature of the con-

[1] `http://www-cad.eecs.berkeley.edu:80/~tah/HyTech/`
[2] `http://www-verimag.imag.fr/TEMPORISE/kronos/`
[3] `http://www.uppaal.com/`

straints which are used in the automata, but not from the syntactic macros that have been added in the updatable timed automata model.

Outline of the paper. The structure of the paper is the following: we first recall some background on the implementation of timed automata and we describe the forward analysis algorithm that uses the widening operator we will study (section 2) ; we then discuss the correctness of this algorithm and get that, even if surprising, the algorithm is not correct in general (section 3) ; we then present the updatable timed automata framework (section 4) and we study very precisely the correctness of the widening operator (section 5) ; we also propose an implementation of the algorithm for the subclasses of updatable timed automata for which a correct widening operator can be found (section 5) ; we discuss some other methods for all the automata for which we can not propose a correct widening operator ; we finally conclude with some remarks (section 6).

Some technical proofs of this paper are presented in Appendix (page 42).

2 Implementation of Timed Automata, State of the Art

In this section, we present the basic notions for timed automata [AD90, AD94], and we focus on implementation issues for this model. In particular, we present an algorithm which is implemented in some tools.

2.1 Preliminaries

If Z is any set, let Z^* be the set of *finite* sequences of elements in Z. We consider as time domain $\mathbb{T}$ the set $\mathbb{Q}^+$ of non-negative rationals or the set $\mathbb{R}^+$ of non-negative reals and Σ as a finite set of *actions*. A *time sequence* over $\mathbb{T}$ is a finite non decreasing sequence $\tau = (t_i)_{1\leq i\leq p} \in \mathbb{T}^*$. A *timed word* $\omega = (a_i, t_i)_{1\leq i\leq p}$ is an element of $(\Sigma \times \mathbb{T})^*$, also written as a pair $\omega = (\sigma, \tau)$, where $\sigma = (a_i)_{1\leq i\leq p}$ is a word in Σ^* and $\tau = (t_i)_{1\leq i\leq p}$ a time sequence in $\mathbb{T}^*$ of same length.

2.1.1 Clock Valuations.

We consider a finite set X of variables, called *clocks*. A *clock valuation* over X is a mapping $v : X \rightarrow \mathbb{T}$ that assigns to each clock a time value. The set of all clock valuations over X is denoted $\mathbb{T}^X$. Let $t \in \mathbb{T}$, the valuation $v + t$ is defined by $(v + t)(x) = v(x) + t$, $\forall x \in X$. We also use the notation $(\alpha_i)_{1\leq i\leq n}$ for the valuation v such that $v(x_i) = \alpha_i$. For a subset C of X, we denote by $[C \leftarrow 0]v$ the valuation such that for each $x \in C$, $([C \leftarrow 0]v)(x) = 0$ and for each $x \in X \setminus C$, $([C \leftarrow 0]v)(x) = v(x)$.

2.1.2 Clock Constraints.

Given a set of clocks X, we introduce two sets of clock constraints over X. The most general one, denoted by $\mathcal{C}(X)$, is defined by the following grammar:

$$\varphi \quad ::= \quad x \sim c \mid x - y \sim c \mid \varphi \wedge \varphi \mid true$$
$$\text{where } x,\ y \in X,\ c \in \mathbb{Q},\ \sim \in \{<, \leq, =, \geq, >\}$$

We also use the proper subset of *diagonal-free* constraints where the comparison between two clocks is not allowed. This set is denoted by $\mathcal{C}_{df}(X)$ and is defined by the grammar:

$$\varphi \quad ::= \quad x \sim c \mid \varphi \wedge \varphi \mid true,$$
$$\text{where } x \in X,\ c \in \mathbb{Q} \text{ and } \sim \in \{<, \leq, =, \geq, >\}$$

We write $v \models \varphi$ when the clock valuation v satisfies the clock constraint φ.

A *k-bounded clock constraint* is a clock constraint that involves only constants between $-k$ and $+k$.

2.1.3 Timed Automata.

A *timed automaton* over $\mathbb{T}$ is a tuple $\mathcal{A} = (\Sigma, Q, T, I, F, X)$, where Σ is a finite alphabet of actions, Q is a finite set of states, X is a finite set of clocks, $T \subseteq Q \times [\mathcal{C}(X) \times \Sigma \times 2^X] \times Q$ is a finite set of transitions, $I \subseteq Q$ is the subset of initial states and $F \subseteq Q$ is the subset of final states.
A *path* in $\mathcal{A}$ is a finite sequence of consecutive transitions:

$$P = q_0 \xrightarrow{\varphi_1, a_1, C_1} q_1 \ \ldots\ q_{p-1} \xrightarrow{\varphi_p, a_p, C_p} q_p$$

where $(q_{i-1}, \varphi_i, a_i, C_i, q_i) \in T$ for each $1 \leq i \leq p$.

The path is said to be *accepting* if it starts in an initial state ($q_0 \in I$) and ends in a final state ($q_p \in F$). A *run* of the automaton through the path P is a sequence of the form:

$$\langle q_0, v_0 \rangle \xrightarrow[t_1]{\varphi_1, a_1, C_1} \langle q_1, v_1 \rangle \ \ldots\ \xrightarrow[t_p]{\varphi_p, a_p, C_p} \langle q_p, v_p \rangle$$

where $\tau = (t_i)_{1 \leq i \leq p}$ is a time sequence and $(v_i)_{1 \leq i \leq p}$ are clock valuations defined by:

$$\begin{cases} v_0(x) = 0,\ \forall x \in X \\ v_{i-1} + (t_i - t_{i-1}) \models \varphi_i \\ v_i = [C_i \leftarrow 0]\,(v_{i-1} + (t_i - t_{i-1})) \end{cases}$$

The label of the run is the timed word $w = (a_1, t_1) \ldots (a_p, t_p)$. If the path P is accepting then the timed word w is said to be accepted by the timed automaton. The set of all timed words accepted by $\mathcal{A}$ is denoted by $L(\mathcal{A})$.

2.2 Practical Issues

For verification purposes, a fundamental question about timed automata is to decide whether the accepted language is empty. This problem is called the *emptiness problem*. A class of timed automata is said *decidable* if the emptiness problem is decidable for this class. Note that this problem is equivalent to the *reachability problem* which tests whether a state can be reached in a timed automaton.

Alur and Dill proved in [AD90, AD94] that the emptiness problem is decidable for timed automata. The proof of this result is based on a "region automaton construction". In practice, this construction is not implemented because it does not support a natural implementation and it suffers from an enormous combinatorics explosion. The idea of the region automaton is to construct a finite simulation graph for the automaton based on an equivalence relation (of finite index) defined on the clock valuations. Some works have been done to reduce the size of this simulation graph by enlarging the equivalence relation, see for example [ACD+92, YL97, TY01]. However, in practice, such graphs are not constructed and on-the-fly zone algorithms glancing symbolically through the graph are implemented. One of the advantages of these algorithms is that they can easily be implemented using the *Difference Bounded Matrices* data structure (DBM for short), initially proposed by [Dil89]. For example, forward analysis algorithms (that is algorithms computing step-by-step the successors of the initial configurations) [Alu99] are implemented in tools like UPPAAL [BL96, LPY97] or KRONOS [BTY97, Daw97, Yov98]. The forward analysis algorithm computing the exact set of successors does not always terminate; in [MP99], subclasses of classical timed automata, for which termination is guaranteed, are proposed, but these classes are a bit restrictive. Different kinds of abstractions [DT98] are thus proposed to enforce the termination, among which a widening operator on zones.

The aim of this paper is to study in details the widening operator proposed in [DT98] and implemented in tools like KRONOS and UPPAAL. We will thus now present the widening operator defined in [DT98] and describe the forward analysis algorithm which uses this operator. This algorithm is implemented in some tools for the verification of classical timed automata. We also explain how this algorithm is implemented using the DBM data structure. Note that this data structure, apart from being adapted to the implementation of algorithms for timed automata, is very interesting in that it will be useful to prove properties of timed automata.

2.3 A Forward Analysis Algorithm for Timed Automata

2.3.1 Zones.

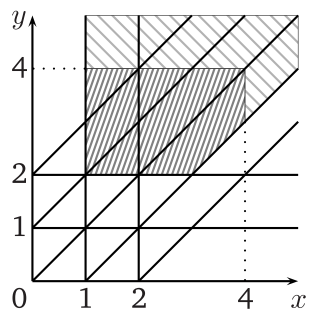
A *zone* is a subset of $\mathbb{T}^n$ defined by a general clock constraint. Let k be a constant. A *k-bounded zone* is a zone defined by a k-bounded clock constraint. Let Z be a zone. The set of k-bounded zones containing Z is finite and not empty ($\mathbb{T}^n$ is a k-bounded zone and contains Z), the intersection of these k-bounded zones is a k-bounded zone containing Z, and is thus the smallest one having this property. It is called the *k-approximation* of Z and is denoted $\text{Approx}_k(Z)$.

Example 1 *Consider the zone Z drawn with on the figure beside: Z is defined by the clock constraint*

$$1 < x < 4 \wedge 2 < y < 4 \wedge x - y < 1.$$

Taking $k = 2$, the k-approximation of Z is drawn adding the part ; it is defined by the clock constraint

$$1 < x \wedge 2 < y \wedge x - y < 1.$$



2.3.2 The Algorithm.

Let $\mathcal{A}$ be a classical timed automaton. If $e = (q \xrightarrow{g,a,C:=0} q')$ is a transition of $\mathcal{A}$ and if Z is a zone, then $\mathsf{Post}(Z, e)$ denotes the set $[C \leftarrow 0](g \cap \overrightarrow{Z})$ where $\overrightarrow{Z}$ represents the *future* of Z and is defined by

$$\overrightarrow{Z} = \{v + t \mid v \in Z \text{ and } t \geq 0\}$$

$\mathsf{Post}(Z, e)$ is the set of valuations which can be reached by waiting in the current state, q, and then taking the transition e. We associate with $\mathcal{A}$ the largest constant, k, appearing in $\mathcal{A}$ (*i.e.* the largest constant c such that there is a constraint $x \sim c$ for some clock x or $x - y \sim c$ for some clocks x and y). A maximal constant can be computed for each clock x (in a similar way), but for our purpose, it does not change anything, the presentation would just be a bit more complicated.

One of the forward analysis algorithms for classical timed automata uses the k-approximation (*cf* section 2.3.1) as a widening operator. It is presented as Algorithm 1 (q_0 is the initial location of the automaton whereas Z_0 represents the initial zone, it is often the valuation where all the clocks are set to zero).

Algorithm 1 Algorithm using k-Approximations for Classical Timed Automata

```
# TA-Algorithm (𝒜:TA) {
#     Define k as the maximal constant appearing in 𝒜;
#     Visited := ∅;                         (* Visited stores the visited states *)
#     Waiting := {(q0,Approx_k(Z0))};
#     Repeat
#        Get and Remove (q,Z) from Waiting;
#        If q is final
#           then {Return "Yes";}
#           else {If there is no (q,Z') ∈ Visited such that Z ⊆ Z'
#                    then {Visited := Visited ∪ {(q,Z)};
#                          Successor := {(q',Approx_k(Post(Z,e))) |
#                                                e transition from q to q'};
#                          Waiting := Waiting ∪ Successor;}}
#     Until (Waiting = ∅);
#     Return "No"; }
```

We first point out that this algorithm terminates because there are finitely many k-bounded zones and thus finitely many k-approximations of zones that can be computed for each control state of the automaton. This algorithm computes step-by-step an overapproximation of the set of reachable states and tests whether this approximation intersects the set of final states. Thus, if the answer of the algorithm is "No", it is sure that no final state can be reached. However, if the answer is "Yes", it can *a priori* be the case that the overapproximation intersects the set of final states whereas the exact set of reachable states does not intersect this set. We will discuss in details the correctness of Algorithm 1 in Section 3.

2.4 The Implementation: the DBM Data Structure

In order to implement Algorithm 1, an adapted data structure is needed to represent the zones and this data structure must allow to test for inclusion of zones and to compute easily the different operations used in the algorithm, that is the intersection of two zones, the future of a zone, the image of a zone by a reset and the k-approximation of a zone. Tools like UPPAAL or KRONOS use the data structure proposed by Dill in [Dil89], the DBM data structure. A detailed presentation of this data structure can be found in [CGP99] and in [Ben02]. However, we will recall some basic properties of it because it will be the core tool in the rest of the paper.

A *difference bounded matrice* (say *DBM* for short) for n clocks is an $(n+1)$-

square matrice of pairs

$$(m; \prec) \in \mathbb{V} = (\mathbb{Z} \times \{<, \leq\}) \cup \{(\infty; <)\}.$$

A DBM $M = (m_{i,j}, \prec_{i,j})_{i,j=1\dots n}$ defines the following subset of $\mathbb{T}^n$ (the clock x_0 is supposed to be always equal to zero, *i.e.* for each valuation v, $v(x_0) = 0$):

$$\{v : \{x_1, \dots, x_n\} \longrightarrow \mathbb{T} \mid \forall\, 0 \leq i, j \leq n,\ v(x_i) - v(x_j) \prec_{i,j} m_{i,j}\}$$

where $\gamma < \infty$ means that γ is some real (there is no bound on it).
This subset of $\mathbb{T}^n$ is a zone and will be denoted, in what follows, by $[\![M]\!]$. Each DBM on n clocks represents a zone of $\mathbb{T}^n$. Note that several DBMs can define the same zone.

Example 2 *The zone defined by the equations $x_1 > 3 \wedge x_2 \leq 5 \wedge x_1 - x_2 < 4$ can be represented by the two DBMs*

$$\begin{pmatrix} (0; \leq) & (-3; <) & (\infty; <) \\ (\infty; <) & (0; \leq) & (4; <) \\ (5; \leq) & (\infty; <) & (0; \leq) \end{pmatrix} \quad \textit{and} \quad \begin{pmatrix} (\infty; <) & (-3; <) & (\infty; <) \\ (\infty; \leq) & (\infty; <) & (4; <) \\ (5; \leq) & (\infty; <) & (0; \leq) \end{pmatrix}$$

Thus the DBMs are not a canonical representation of zones. Moreover, it isn't possible to test syntactically whether $[\![M_1]\!] = [\![M_2]\!]$. A normal form has thus been defined for representing zones. Its computation uses the Floyd algorithm and some syntactic rewritings (see [Dil89, CGP99] for a description of this procedure). In what follows, we denote by $\phi(M)$ the *normal form* of M. Before stating some very important properties of the normal form, we define a total order on $\mathbb{V}$ in the following way: if $(m; \prec), (m'; \prec') \in \mathbb{V}$, then

$$(m; \prec) \leq (m'; \prec') \iff \begin{cases} m < m' \\ \text{or} \\ m = m' \text{ and either } \prec = \prec' \text{ or } \prec' = \leq . \end{cases}$$

Of course, for each $m \in \mathbb{Z}$, it holds that $m < \infty$. We define $>$, $\geq$ and $<$ in a natural way. These orders are extended to the DBMs in the following way: let $M = (m_{i,j}; \prec_{i,j})_{i,j=0\dots n}$ and $M' = (m'_{i,j}; \prec'_{i,j})_{i,j=0\dots n}$ be two DBMs, then

$$M \leq M' \iff \text{for every } i, j = 0 \dots n,\ (m_{i,j}; \prec_{i,j}) \leq (m'_{i,j}; \prec'_{i,j}).$$

We can now state some (very useful) properties of normal forms. If M and M' are DBMs, then:

(i) $[\![M]\!] = [\![\phi(M)]\!]$ and $\phi(M) \leq M$,

(ii) $[\![M]\!] \subseteq [\![M']\!] \iff \phi(M) \leq M' \iff \phi(M) \leq \phi(M')$.

The last point expresses the fact that the test for inclusion of zones can be checked syntactically on the normal forms of the DBMs (representing the zones).
Normal forms of DBMs can be characterized in a natural way. If $M = (m_{i,j}; \prec_{i,j})_{i,j=0\ldots n}$ is a DBM such that $[\![M]\!] \neq \emptyset$, then the two following properties are equivalent:

(i) M is in normal form,

(ii) for every $i, j = 0 \ldots n$, for every real $-m_{j,i} \prec_{j,i} r \prec_{i,j} m_{i,j}$, there exists a valuation $v \in [\![M]\!]$ such that $v(x_j) - v(x_i) = r$ (still assuming that $v(x_0) = 0$).

This property expresses the fact that if a DBM is in normal form, then no constraint of this DBM can be tightened using the Floyd algorithm.

The emptiness can be checked on the DBMs, even if they are not in normal form. For this, we define an addition on the set $\mathbb{V}$:

$$(m; \prec) + (m'; \prec') = (m''; \prec'')$$

where $m'' = m+m'$ and $\prec''$ is $\leq$ if both $\prec$ and $\prec'$ are $\leq$ and $\prec''$ is $<$ otherwise. Now, if $M = ((m_{i,j}; \prec_{i,j})_{i,j})$ is a DBM, $[\![M]\!] = \emptyset$ if and only if there exists a negative cycle in M, which means that there exists a sequence of distinct indices $(i_1, i_2, \ldots, i_{l-1}, i_l = i_1)$ such that

$$(m_{i_1,i_2}; \prec_{i_1,i_2}) + (m_{i_2,i_3}; \prec_{i_2,i_3}) + \cdots + (m_{i_{l-1},i_l}; \prec_{i_{l-1},i_l}) < (0; \leq)$$

This condition will be used a lot in the remainder of the paper.

2.4.1 Computation of Some Operations on DBMs.

As we argued at the beginning of the section, the data structure used to represent zones must also be appropriate to compute the four operations on zones that are used by Algorithm 1, namely future, intersection, image by resets and k-approximation. These operations on DBMs are described nicely in [CGP99] and [Ben02], here we recall them quickly.

Intersection. Assume that $M = (m_{i,j}; \prec_{i,j})_{i,j=1\ldots n}$ and $M' = (m'_{i,j}; \prec'_{i,j})_{i,j=1\ldots n}$ are two DBMs. Then, defining $M'' = (m''_{i,j}; \prec''_{i,j})_{i,j=1\ldots n}$ by

$$(m''_{i,j}; \prec''_{i,j}) = \min((m_{i,j}; \prec_{i,j}), (m'_{i,j}; \prec'_{i,j})) \text{ for all indices } i, j = 1 \ldots n.$$

We get that $[\![M'']\!] = [\![M]\!] \cap [\![M']\!]$. Note that it can be the case that M'' is not in normal form, even if M and M' are in normal form.

Future. Assume that $M = (m_{i,j}; \prec_{i,j})_{i,j=1\ldots n}$ is a DBM in normal form. We define the DBM $\overrightarrow{M} = (m'_{i,j}; \prec'_{i,j})_{i,j=1\ldots n}$ by:

$$\begin{cases} (m'_{i,j}; \prec'_{i,j}) = (m_{i,j}; \prec_{i,j}) & \text{if } j \neq 0 \\ (m'_{i,0}; \prec'_{i,0}) = (\infty; <) \end{cases} .$$

We get that $[\![\overrightarrow{M}]\!] = \overrightarrow{[\![M]\!]}$ and that the DBM $\overrightarrow{M}$ is in normal form.

Image by resets. Assume that $M = (m_{i,j}; \prec_{i,j})_{i,j=1\ldots n}$ is a DBM in normal form. We define the DBM $M_{x_k:=0} = (m'_{i,j}; \prec'_{i,j})_{i,j=1\ldots n}$ by:

$$\begin{cases} (m'_{i,j}; \prec'_{i,j}) = (m_{i,j}; \prec_{i,j}) & \text{if } i, j \neq k \\ (m'_{k,k}; \prec'_{k,k}) = (m'_{k,0}; \prec'_{k,0}) = (m'_{0,k}; \prec'_{0,k}) = (0; \leq) & \\ (m'_{i,k}; \prec'_{i,k}) = (m_{i,0}; \prec_{i,0}) & \text{if } i \neq k \\ (m'_{k,i}; \prec'_{k,i}) = (m_{0,i}; \prec_{0,i}) & \text{if } i \neq k \end{cases}$$

We get that $[\![M_{x_k:=0}]\!] = [x_k \leftarrow 0][\![M]\!]$ and that the DBM $M_{x_k:=0}$ is in normal form.

k-approximation. Assume that $M = (m_{i,j}; \prec_{i,j})_{i,j=1\ldots n}$ is a DBM in normal form. We define the DBM $M_k = (m'_{i,j}; \prec'_{i,j})_{i,j=1\ldots n}$ by:

$$\begin{cases} (m'_{i,j}; \prec'_{i,j}) = (m_{i,j}; \prec_{i,j}) & \text{if } -k \leq m_{i,j} \leq k \\ (m'_{i,j}; \prec'_{i,j}) = (\infty; <) & \text{if } m_{i,j} > k \\ (m'_{i,j}; \prec'_{i,j}) = (-k; <) & \text{if } m_{i,j} < -k \end{cases}$$

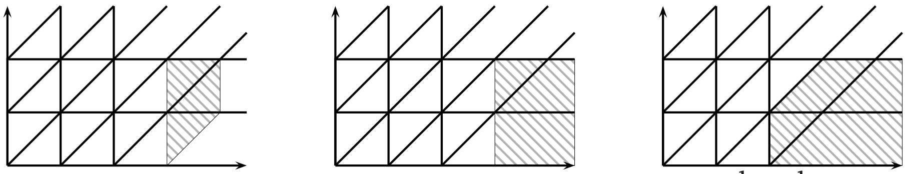
We get that $[\![M_k]\!] = \text{Approx}_k([\![M]\!])$, but M_k is not in normal form.

Combining all these constructions, if e is a transition of the form $q \xrightarrow{g,a,C:=0} q'$ and if M is a DBM representing a zone Z, using what precedes, it is easy to compute a DBM which represents the zone $\text{Approx}_k([C \leftarrow 0](g \cap \overrightarrow{Z})) = \text{Approx}_k(\mathsf{Post}(Z, e))$. This DBM may not be in normal form. The DBM data structure can thus be used to compute all the operations needed by Algorithm 1.

3 Correctness Problems

As argued in Section 2.3.2, Algorithm 1 raises a correctness problem. Indeed, it computes an overapproximation of the set of reachable states which can *a priori* be strictly larger than the set of reachable states. This algorithm is much implemented and used, nevertheless, we did not find any complete proof of its correctness in the literature. There exist of course several papers claiming that they present such proofs but these proofs are incomplete and even sometimes incorrect:

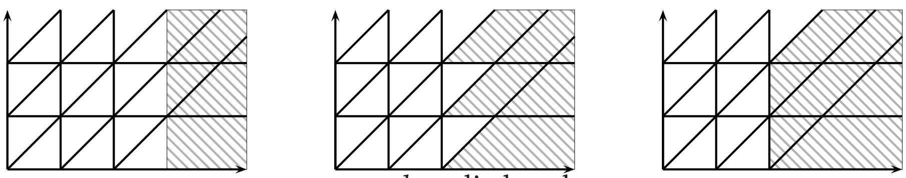
- In the PhD. Thesis [Tri98], the implementation of the c-*closure* presented page 127 (of [Tri98]) corresponds to our Approx_c approximation. Thus, their Yes/No Algorithm should correspond to Algorithm 1. However, the definition of the c-closure given page 21 does not correspond to the implementation. For example, consider the following graph where c is 2:

  zone Z | c-closure of Z | what the implementation really computes

  The proof of correctness of the Yes/No Reachability Algorithm (Lemma 5.9 page 58) is done using the c-closure which is defined and not the one which is implemented. Thus, the correctness of the implemented algorithm is not complete in this paper.

- In the PhD. Thesis [WT94], if Z is a zone then *round*(Z) corresponds to our Approx_k overapproximation. The theorem which aims at proving that Algorithm 1 is correct is Theorem 4.8 on page 90. The proof is done in the following way: considering a zone Z, it is decomposed into finitely many subzones that are assumed to be included in regions, then the computation of the *round* operator is done first for the subzones (this step is easy); *round*(Z) is obtained by computing the union of the *round* of these subzones. But *round* is not compatible with union, for example have a look at the following zone:

  zone Z | *round* applied to the subzones of Z | *round*(Z)

  Besides, this proof only aimed at proving the correctness of Algorithm 1 when we restrict to diagonal-free timed automata.

A surprising observation...

In trying to write a complete proof for the correctness of Algorithm 1, we have had some troubles, and studying precisely what were the problems we were confronted to, we have been forced to face the facts that **Algorithm 1 can not be correct!** Consider for example the automaton $\mathcal{C}ex$ depicted on figure 1.

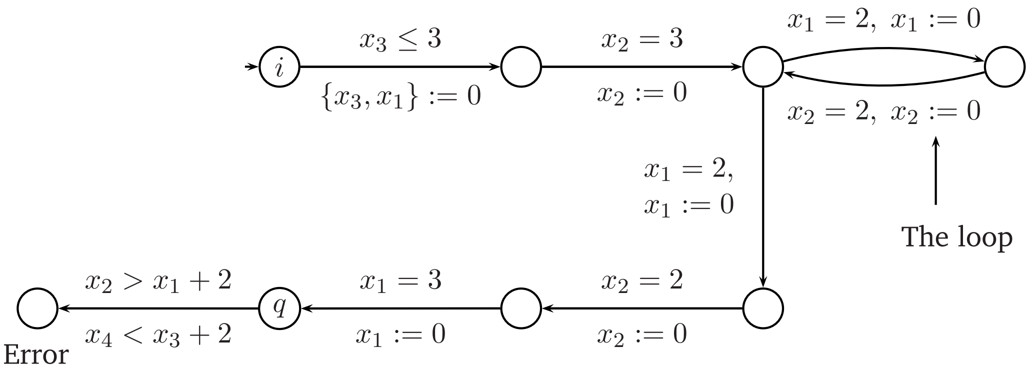


Figure 1: A surprising counter-example, $\mathcal{C}ex$

It is just a matter of computing successors of zones to get that the zones we can reach in state q, when we start with the valuation where all the clocks are set to zero in state i, are

$$\begin{cases} x_3 \geq 2\alpha + 5 \\ x_2 \geq 1 \\ x_4 \geq 2\alpha + 6 \\ 1 \leq x_2 - x_1 \leq 3 \\ x_3 - x_1 = 2\alpha + 5 \\ 2\alpha + 6 \leq x_4 - x_1 \leq 2\alpha + 8 \\ 2\alpha + 2 \leq x_3 - x_2 \leq 2\alpha + 4 \\ x_4 - x_2 = 2\alpha + 5 \\ 1 \leq x_4 - x_3 \leq 3 \end{cases}$$

where α is the number of loops which are taken (we call Z_α this zone). This kind of zones is depicted on figure 3.
We can then notice that, for every α, if we impose the condition $x_4 - x_3 < 2$ in Z_α, then $x_2 - x_1 < 2$ because

$$\begin{aligned} x_2 - x_1 &= (x_2 - x_4) + (x_4 - x_3) + (x_3 - x_1) \\ &< (-2\alpha - 5) + 2 + (2\alpha + 5) \\ &< 2 \end{aligned}$$

Thus, the state “Error” of $\mathcal{C}ex$ is not reachable. However, if we fix a maximal constant M and if we use the widening operator using this constant M (that

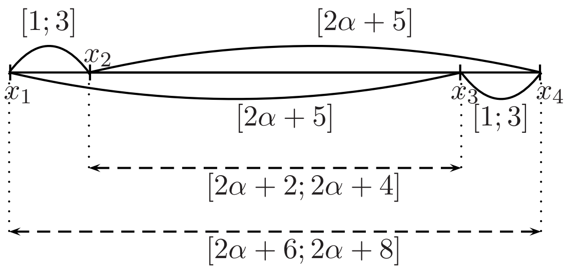


Figure 2: The zone Z_α

is if we use the operator Approx_M), then for a number of loops α sufficiently large (that is such that $2\alpha + 5 > M$), the constraint $(x_3 - x_1 \leq 2\alpha + 5)$ will be erased when we will apply the widening operator and thus Algorithm 1 will conclude that the state "Error" is reachable. Thus, we conclude that for the automaton $\mathcal{C}ex$, there exists no widening operator which is correct with respect to reachability.
From this counter-example, we get that the widening operator is not correct with respect to reachability and this was a big surprise! More precisely, we can state the following result:

Theorem 1 *There is no way to have a correct forward analysis algorithm for timed automata that uses a widening operator on zones.*

The aim of the rest of this paper is to study more precisely where appear the problems and to bring out subclasses for which we can find a correct widening operator. We decided to do this study in the more general framework of updatable timed automata, as defined in [BDFP00a].

4 The Updatable Timed Automata Framework

Updatable timed automata are a syntactic extension of timed automata that has been introduced in [BDFP00a]. We will now present the model, the basic results and propose a first algorithm which is correct for all the updatable timed automata that are decidable.

4.1 Definitions

4.1.1 Updates.

An *update* is a function from $\mathbb{T}^X$ to $\mathcal{P}(\mathbb{T}^X)$ which assigns to each valuation a set of valuations. In this work, updates are restricted according to the following definitions.

A *simple update* over a clock z has one of the two following forms:

$$\begin{array}{lll} up & ::= & z :\sim c \ \mid \ z :\sim y + c \\ & & \text{where } c \in \mathbb{Q},\ y \in X \text{ and } \sim \in \{<, \leq, =, \geq, >\} \end{array}$$

Let v be a valuation and up be a simple update over z. A valuation v' is in $up(v)$ if $v'(y) = v(y)$ for any clock $y \neq z$ and if $v'(z)$ verifies ($\sim \in \{<, \leq, =, \neq, \geq, >\}$):

$$\begin{cases} v'(z) \sim c \text{ and } v'(z) \geq 0 & \text{if } up = z :\sim c \\ v'(z) \sim v(y) + d \text{ and } v'(z) \geq 0 & \text{if } up = z :\sim y + d \end{cases}$$

In what follows, an *update* over a set of clocks X is a collection $up = (up_i)_{1 \leq i \leq k}$, where each up_i is a simple update over some clock $x_i \in X$ (note that it could happen that $x_i = x_j$ for some $i \neq j$). Let $v,\ v' \in \mathbb{T}^n$ be two clock valuations, $v' \in up(v)$ if and only if, for all i, the clock valuation v_i defined by

$$\begin{cases} v_i(x_i) = v'(x_i) \\ v_i(y) = v(y) & \text{for any } y \neq x_i \end{cases}$$

is in $up_i(v)$. The set of updates over the set of clocks X is denoted by $\mathcal{U}(X)$.

Example 3 *For the update $up = (x :> y, x :< 7)$, a valuation v' is in $up(v)$ if the value $v'(x)$ verifies: $v'(x) > v(y) \wedge v'(x) < 7$. Note that $up(v)$ may be empty. For instance, the update $(x :< 1, x :> 1)$ leads to the empty set.*

An update is a reset whenever it is a collection of simple updates of the form $x := 0$. The set of resets is denoted by $\mathcal{U}_0(X)$. For a subset C of X, if up is the reset $\bigwedge_{x \in C} x := 0$ (we write $C := 0$), for each valuation v, the valuation $up(v)$ is also denoted by $[C \leftarrow 0]v$.

4.1.2 Updatable Timed Automata.

An *updatable timed automaton* over $\mathbb{T}$ is a 6-tuple $\mathcal{A} = (\Sigma, Q, T, I, F, X)$, where Σ is a finite alphabet of actions, Q is a finite set of states, X is a finite set of clocks, $T \subseteq Q \times [\mathcal{C}(X) \times \Sigma \times \mathcal{U}(X)] \times Q$ is a finite set of transitions, $I \subseteq Q$ is the subset of initial states and $F \subseteq Q$ is the subset of final states.

A *path* in $\mathcal{A}$ is a finite sequence of consecutive transitions:

$$P = q_0 \xrightarrow{\varphi_1, a_1, up_1} q_1 \ \ldots \ q_{p-1} \xrightarrow{\varphi_p, a_p, up_p} q_p$$

where $(q_{i-1}, \varphi_i, a_i, up_i, q_i) \in T$ for each $1 \leq i \leq p$.

The path is said to be *accepting* if it starts in an initial state ($q_0 \in I$) and ends in a final state ($q_p \in F$). A *run* of the automaton through the path P is a sequence of the form:

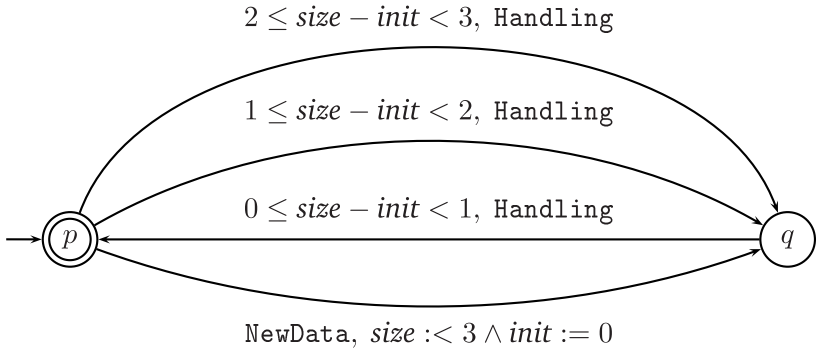
$$\langle q_0, v_0 \rangle \xrightarrow[t_1]{\varphi_1, a_1, up_1} \langle q_1, v_1 \rangle \ \ldots \ \xrightarrow[t_p]{\varphi_p, a_p, up_p} \langle q_p, v_p \rangle$$

where $\tau = (t_i)_{1 \leq i \leq p}$ is a time sequence and $(v_i)_{1 \leq i \leq p}$ are clock valuations defined by:

$$\begin{cases} v_0(x) = 0,\ \forall x \in X \\ v_{i-1} + (t_i - t_{i-1}) \models \varphi_i \\ v_i \in up_i\,(v_{i-1} + (t_i - t_{i-1})) \end{cases}$$

The label of the run is the timed word $w = (a_1, t_1) \ldots (a_p, t_p)$. If the path P is accepting then the timed word w is said to be accepted by the timed automaton. The set of all timed words accepted by $\mathcal{A}$ is denoted by $L(\mathcal{A})$.

Example 4 *Consider a system that handles data which come from an external environment. The time needed to deal the data depends on its (bounded) size. Updatable timed automata provide an elegant and concise way to model such a system. Indeed, it is sufficient to store the size in a variable taking non-deterministically a value less than 3 (assuming that the maximum size for the data is 3).*



Let $\mathcal{C} \subseteq \mathcal{C}(X)$ be a set of clock constraints and $\mathcal{U} \subseteq \mathcal{U}(X)$ be a set of updates, the class $Aut(\mathcal{C}, \mathcal{U})$ is the set of all updatable timed automata whose transitions only use clock constraints of $\mathcal{C}$ and updates of $\mathcal{U}$ only. The usual class of timed automata, defined in [AD90, AD94], is the family $Aut(\mathcal{C}_{df}(X), \mathcal{U}_0(X))$. However, what we now usually call the family of *classical timed automata*, as presented in section 2.1.3, is the family $Aut(\mathcal{C}(X), \mathcal{U}_0(X))$.

4.1.3 Known Results about the Emptiness Problem.

We characterized the decidable subclasses of updatable timed automata in [BDFP00a]. The following tabular summarizes roughly the results:

<table>
<tr><th>$\mathcal{U}_0(X) \cup \{ \ldots \}$</th><th>Diagonal-free clock constraints</th><th>General clock constraints</th></tr>
<tr><td>$x := c$, $x := y$</td><td rowspan="3">PSPACE-complete</td><td>PSPACE-complete</td></tr>
<tr><td>$x := x + 1$</td><td rowspan="4">Undecidable</td></tr>
<tr><td>$x := y + c$</td></tr>
<tr><td>$x := x - 1$</td><td rowspan="2">Undecidable</td></tr>
<tr><td>$x := y - c$</td></tr>
<tr><td>$x :< c$</td><td rowspan="4">PSPACE-complete</td><td>PSPACE-complete</td></tr>
<tr><td>$x :> c$</td><td rowspan="4">Undecidable</td></tr>
<tr><td>$x :\sim y + c$</td></tr>
<tr><td>$y + c <: x :< y + d$</td></tr>
<tr><td>$y + c <: x :< z + d$</td><td>Undecidable</td></tr>
</table>

with $\sim \in \{\leq, <, >, \geq\}$ and $c,\ d \in \mathbb{Q}^+$.

The table reads as follows: the class of updatable timed automata that only use diagonal-free clock constraints but that also use decrementation updates (*i.e.* updates of the form $x := x - 1$) is undecidable whereas the class of updatable timed automata that use general clock constraints and updates of the form $x :< c$ is decidable using a PSPACE algorithm.

The proof of these decidability results is based on the generalization of the construction of the region automaton for classical timed automata (as presented in [AD90, AD94]). For the following of the paper, we will need more precise decidability results, we present them now.

4.2 Regions and Precise Results about Decidability [BDFP00a]

4.2.1 Regions.

The definition of regions we use in this paper is the one presented in [BDFP00a]. It is a bit different from the initial definition of Alur and Dill [AD90, AD94]. We will see the difference in Example 5. Let X be a set of clocks, $n = |X|$ and

$$\alpha = ((\max_x)_{x \in X}; (\max_{x,y})_{x,y \in X}) \in \mathbb{N}^n \times (\mathbb{Z} \times \mathbb{Z})^{n^2}$$

be a tuple such that for every $x, y \in X$, $\begin{cases} -\max_{y,x} \leq \max_{x,y} \\ \max_{x,y} \leq \max_x \end{cases}$.

The tuple α defines a set of regions $\mathcal{R}_\alpha$: a *region* R of $\mathcal{R}_\alpha$ is defined by a tuple

$$((I_x)_{x \in X}; (J_{x,y})_{x \neq y \in X}; \prec)$$

where:

- I_x is one of the following intervals:

$$\begin{array}{c}]-\infty; -\max_x[\quad ; \quad]\max_x; +\infty[\quad ; \\ ]c; c+1[\quad \text{where} \quad -\max_x \leq c < \max_x \\ \text{or} \quad \{c\} \quad \text{where} \quad -\max_x \leq c \leq \max_x \end{array}$$

- $J_{x,y}$ is one of the following intervals:

$$\begin{array}{c}]-\infty; -\max_{y,x}[\quad ; \quad]\max_{x,y}; +\infty[\quad ; \\ ]c; c+1[\quad \text{where} \quad -\max_{y,x} \leq c < \max_{x,y} \\ \text{or} \quad \{c\} \quad \text{where} \quad -\max_{y,x} \leq c \leq \max_{x,y} \end{array}$$

- $\prec$ is a total preorder defined on the set

$$\{x \mid \exists\, c \in [-\max_x; \max_x[\text{ such that } I_x =]c; c+1[\}.$$

A valuation v is in the region R if:[4]

$$\left\{ \begin{array}{ll} v(x) \in I_x & \text{for each } x \in X \\ I_x =]\max_x; +\infty[\text{ or } I_y =]\max_y; +\infty[\implies v(x) - v(y) \in J_{x,y} & \text{for every } x \neq y \in X \\ x \prec y \implies \mathtt{frac}(v(x)) \leq \mathtt{frac}(v(y)) & \text{for every } x, y \in X. \end{array} \right.$$

Note that for a valuation v, there is a unique region in $\mathcal{R}_\alpha$ containing v; it is denoted by $[v]_\alpha$.

Example 5 *We define the tuple*

$$\alpha = ((\max_x, \max_y), (\max_{x,y}, \max_{y,x})) \quad \textit{as}$$

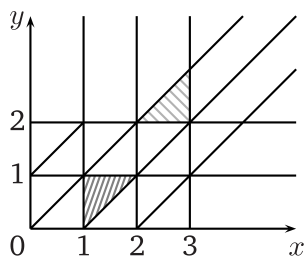
$$\max_x = 3,\ \max_y = 2,\ \max_{x,y} = 2,\ \max_{y,x} = 0$$

The set of regions $\mathcal{R}_\alpha$ is represented on the figure beside.
The region drawn with ▧ *is represented by the intervals*

$$\left\{ \begin{array}{l} I_x =]2, 3[\\ I_y =]2, +\infty[\end{array} \right. \quad \textit{and} \quad \left\{ \begin{array}{l} I_{x,y} =]0, 1[\\ I_{y,x} =]-1, 0[\end{array} \right.$$

The region drawn with ▨ *is represented by the following intervals and preorder:*

$$\left\{ \begin{array}{l} I_x =]1, 2[\\ I_y =]1, 0[\end{array} \right. \quad \textit{and } \prec \textit{ is defined by } y \prec x$$



[4] $\mathtt{frac}(\gamma)$ denotes the fractional part of γ

The difference with traditional sets of regions [AD90, AD94] is that we allow some diagonal constraints in the non-bounded regions.

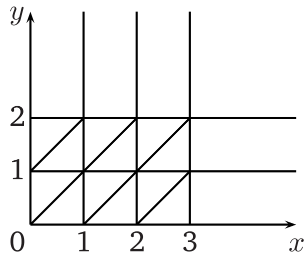
We can note that every convex union of regions of $\mathcal{R}_\alpha$ is a zone. The set of such convex unions of regions of $\mathcal{R}_\alpha$ is denoted by $\mathcal{Z}_\alpha$. If Z is a zone, we define $\mathrm{Approx}_\alpha(Z)$ the smallest zone of $\mathcal{Z}_\alpha$ that contains Z (this zone is well-defined because there are finitely many zones in $\mathcal{Z}_\alpha$, thus it can be argued as in section 2.3.1 for the k-approximation that this smallest zone exists). We say that $\mathrm{Approx}_\alpha(Z)$ is the *α-overapproximation* of Z. We can note that the k-overapproximation of a zone defined in Section 2.3.1 corresponds to the α-overapproximation when $\alpha = ((k, \ldots, k), (k, \ldots, k))$.

A set of regions $\alpha = ((\max_x))_{x \in X}; (\max_{x,y})_{x,y \in X})$ is called *diagonal-free* if for all clocks $x,\ y \in X$, $\max_{x,y} = -\infty$. In this case we simply write $\alpha = (\max_x)_{x \in X}$.

Example 6 *We define α as the (diagonal-free) tuple $(\max_x, \max_y)$ defined by:*

$$\max_x = 3,\ \max_y = 2$$

The set of regions $\mathcal{R}_\alpha$ is represented on the figure beside.



4.2.2 Precise Decidability Results.

In Section 4.1.3, we presented roughly the decidability results we obtained in [BDFP00a]. We will now make these results more precise. We distinguish two different cases, the updatable timed automata with diagonal-free clock constraints and the updatable timed automata with general clock constraints.

Diagonal-free updatable timed automata. Let $\mathcal{C} \subseteq \mathcal{C}_{df}(X)$ be a set of diagonal-free clock constraints and let $\mathcal{U} \subseteq \mathcal{U}(X)$ be a set of updates such that

$$up = \bigwedge_{x \in X} up_x \in \mathcal{U} \iff \text{for each } x,\ up_x \in \{\mathrm{det}_x, \mathrm{inf}_x, \mathrm{sup}_x, \mathrm{int}_x\} \text{ where:}$$

$$\begin{cases} \mathrm{det}_x ::= x := c \mid x := z + d \quad \text{with } c \in \mathbb{N},\ d \in \mathbb{Z} \text{ and } z \in X \\ \mathrm{inf}_x ::= x :\lhd c \mid x :< z + d \mid \mathrm{inf}_x \wedge \mathrm{inf}_x \\ \qquad\qquad\qquad\qquad \text{with } \lhd \in \{<, \leq\},\ c \in \mathbb{N},\ d \in \mathbb{Z} \text{ and } z \in X \\ \mathrm{sup}_x ::= x :\rhd c \mid x :> z + d \mid \mathrm{sup}_x \wedge \mathrm{sup}_x \\ \qquad\qquad\qquad\qquad \text{with } \rhd \in \{>, \geq\},\ c \in \mathbb{N},\ d \in \mathbb{Z} \text{ and } z \in X \\ \mathrm{int}_x ::= x :\in (c; d) \mid x :\in (c; z + d) \mid x :\in (z + c; d) \mid x :\in (z + c; z + d) \\ \qquad\qquad \text{where (and) are either [or], } z \text{ is a clock and } c,\ d \text{ are in } \mathbb{Z}. \end{cases}$$

If the system of linear Diophantine inequations over the variables $(\max_x)_{x \in X}$

$$\begin{aligned} &\{c \leq \max_x \mid x \sim c \in \mathcal{C} \text{ or } x :\sim c \in \mathcal{U}\} \\ \cup\, &\{\max_z \leq \max_y + c \mid z :\sim y + c \in \mathcal{U}\} \end{aligned} \tag{1}$$

has a solution, then the class $Aut(\mathcal{C},\mathcal{U})$ is decidable. Moreover, if $\alpha = (\max_x)_{x \in X}$ is a solution of this system, then we can test for emptiness of any updatable timed automaton from $Aut(\mathcal{C},\mathcal{U})$ using the region automaton based on the diagonal-free set of regions $\mathcal{R}_\alpha$. Of course, this condition is not *necessary*, it can be the case that this system does not have a solution, but that its emptyness can however be decided, for example when clocks are bounded, but updates which are used lead to systems which dont't have any solution.

Remark 1 *The system of equations (1) has been set in order for a solution α to define a set of regions $\mathcal{R}_\alpha$ such that the region automaton using $\mathcal{R}_\alpha$ as set of regions is bisimilar to the original updatable timed automaton. All the conditions described by (1) aim at ensuring this property. See [BDFP00a] for detailed explanations on how these conditions ensure the bisimulation property.*

General updatable timed automata. Let $\mathcal{C} \subseteq \mathcal{C}(X)$ be a finite set of clock constraints and $\mathcal{U} \subseteq \mathcal{U}(X)$ be a finite set of updates such that

$$up = \bigwedge_{x \in X} up_x \in \mathcal{U} \iff \forall x \in X, up_x \in \begin{array}{l} \{x := c, x :< c, x :\leq c \mid c \in \mathbb{N}\} \\ \cup\, \{x := y \mid y \in X\}. \end{array}$$

Then the class $Aut(\mathcal{C},\mathcal{U})$ is decidable. Moreover, if the tuple $\alpha = ((\max_x)_{x \in X}; (\max_{x,y})_{x,y \in X})$ is a solution of the following linear Diophantine inequation system:

$$\begin{array}{ll} & \{c \leq \max_x \mid x \sim c \in \mathcal{C}\} \\ \cup & \{c \leq \max_{x,y} \mid x - y \sim c \in \mathcal{C}\} \\ \cup & \{c \leq \max_x,\ \max_z \geq c + \max_{z,x} \mid x :< c \text{ or } x :\leq c \text{ or } x := c\ \in \mathcal{U},\ \text{and } z \in X\} \\ \cup & \{\max_x \leq \max_y,\ \max_{z,y} \geq \max_{z,x},\ \max_{x,z} \leq \max_{x,y} \mid x := y \in \mathcal{U} \text{ and } z \in X\} \end{array} \tag{2}$$

then we can test for emptiness of every updatable timed automaton from $Aut(\mathcal{C},\mathcal{U})$ using the region automaton based on the set of regions $\mathcal{R}_\alpha$.

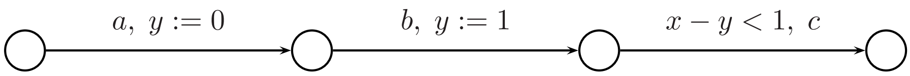
Remark 2 *Considering a classical timed automaton $\mathcal{A}$ and defining the constant k as the largest constant appearing in $\mathcal{A}$, the set of regions $\alpha = ((k,\ldots,k),(k,\ldots,k))$ is a solution to the system which is associated to $\mathcal{A}$ as above.*

Remark 3 *In both cases (diagonal-free and general frameworks), the set of solutions of the linear system corresponding to an updatable timed automaton might have several solutions. However, if there is a solution, then there is a unique minimal solution (for the natural order on the tuples): indeed, all the inequations are of one of the forms*

$$X \geq c,\ X - Y \geq c$$

where X and Y are variables and c is a constant. Thus, if we denote by $(X_i)_{i=1\ldots n}$ all the variables involved in the system, it can be represented by a DBM M and the unique minimal solution is the offset-point of the zone represented by M, that is the point with minimal coordinates belonging to the zone represented by M.

Example 7 *Consider the following updatable timed automaton:*



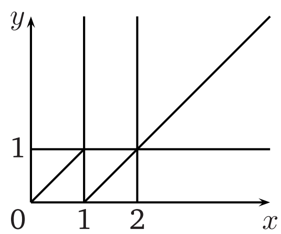
The system of inequations associated with this automaton is the following:

$$\begin{cases} \max_y \geq 0 \\ \max_x \geq 0 + \max_{x,y} \\ \max_y \geq 1 \\ \max_x \geq 1 + \max_{x,y} \\ \max_{x,y} \geq 1 \end{cases}$$

The minimal solution α of this system is:

$$\begin{cases} \max_x = 2 \\ \max_y = 1 \\ \max_{x,y} = 1 \end{cases}$$

The set of regions defined by these constants is thus presented with plain lines on the figure beside.



In what follows we will only consider decidable updatable timed automata, that is updatable timed automata that belong to some decidable class $Aut(\mathcal{C},\mathcal{U})$. We will associate with each decidable updatable timed automaton a set of regions, $\mathcal{R}_\alpha$ as described above, where α is the minimal solution of the previous system (the smallest is the solution, the smallest is the number of regions of $\mathcal{R}_\alpha$ and the more abstract is the region automaton).

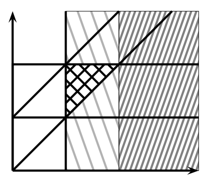
4.3 A First Correct Algorithm for Decidable Updatable Timed Automata

For each zone Z, for each tuple α, we define the closure by regions from $\mathcal{R}_\alpha$ of Z by

$$\text{Closure}_\alpha(Z) = \bigcup \{R \in \mathcal{R}_\alpha \mid R \cap Z \neq \emptyset\}.$$

We say that $\text{Closure}_\alpha(Z)$ is the *α-closure* of Z. First note that $\text{Closure}_\alpha(Z)$ can be different from $\text{Approx}_\alpha(Z)$: $\text{Approx}_\alpha(Z)$ is convex whereas $\text{Closure}_\alpha(Z)$ can be non-convex.

Example 8 *Let us consider the set of regions defined as on the picture beside. If we consider the zone Z drawn using then the closure by region* $\text{Closure}_\alpha(Z)$ *is drawn adding the part : it is thus non convex. The α-approximation* $\text{Approx}_\alpha(Z)$ *is obtained by adding the part in .*



The first algorithm is presented as Algorithm 2.

Algorithm 2 A First Algorithm for Decidable Updatable Timed Automata

```
# First UTA-Algorithm (A:UTA) {
#     Compute α using U and C;
#     Visited := ∅;                    (* Visited stores the visited states *)
#     Waiting := {(q_0,Closure_α(Z_0))};
#     Repeat
#        Get and Remove (q,Z) from Waiting;
#        If q is final
#           then {Return "Yes";}
#           else {If there is no (q,Z') ∈ Visited such that Z ⊆ Z'
#                    then {Visited := Visited ∪ {(q,Z)};
#                          Successor := {(q',Closure_α(Post(Z,e))) |
#                                                    e transition from q to q'};
#                          Waiting := Waiting ∪ Successor;}}
#     Until (Waiting = ∅);
#     Return "No"; }
```

This algorithm of course terminates (if α is fixed, when Z varies, there are finitely many different $\text{Closure}_\alpha(Z)$). We will prove the correctness of this algorithm.

Let $\mathcal{U}$ be a set of updates and $\mathcal{C}$ a set of clock constraints. Let α be the tuple defining the set of regions $\mathcal{R}_\alpha$ as in Section 4.2. It implies in particular (see [BDFP00a]) that for each clock constraint $g \in \mathcal{C}$, for each update $up \in \mathcal{U}$, for each region $R \in \mathcal{R}_\alpha$, for each valuation $v \in R$:

$$\begin{cases} R \subseteq g \quad \text{or} \quad R \subseteq \neg g \\ up(v) \cap R \neq \emptyset \implies \forall v' \in [v]_\alpha,\ up(v') \cap R \neq \emptyset \end{cases} \tag{3}$$

This ensures that the emptiness problem is decidable for the class $Aut(\mathcal{C},\mathcal{U})$ (see [BDFP00a]).

Proposition 1 *The "First UTA-Algorithm" tests for emptiness of every decidable updatable timed automaton.*

The proof of this proposition is proved using several intermediary results. We first claim that:

Lemma 1 *The two following conditions hold:*

$$\begin{cases} \mathsf{Closure}_\alpha(g) = g & \textit{for each clock constraint } g \textit{ in } \mathcal{C} \\ up(\mathsf{Closure}_\alpha(Z)) \subseteq \mathsf{Closure}_\alpha(up(Z)) & \textit{for each zone } Z \textit{ and for each update } up \textit{ in } \mathcal{U} \end{cases} \tag{4}$$

➜ *See the proof in Appendix.*

We also claim that:

Lemma 2 *For each zone Z, $\overrightarrow{\mathsf{Closure}_\alpha(Z)} \subseteq \mathsf{Closure}_\alpha(\overrightarrow{Z})$ and for each clock constraint g such that* $\mathsf{Closure}_\alpha(g) = g$, $g \cap \mathsf{Closure}_\alpha(Z) \subseteq \mathsf{Closure}_\alpha(g \cap Z)$.

➜ *See the proof in Appendix.*

Using these two lemmas, we are now ready to prove the proposition. Using the conditions (4), if $e = (q, g, a, up, q')$ is a transition, we compute:

$$\begin{aligned} \mathsf{Post}(\mathsf{Closure}_\alpha(Z), e) &= up(g \cap \overrightarrow{\mathsf{Closure}_\alpha(Z)}) \\ &\subseteq up(g \cap \mathsf{Closure}_\alpha(\overrightarrow{Z})) \text{ from Lemma 2} \\ &\subseteq up(\mathsf{Closure}_\alpha(g \cap \overrightarrow{Z})) \text{ from Lemma 2} \\ &\subseteq \mathsf{Closure}_\alpha(up(g \cap \overrightarrow{Z})) \text{ from condition (4)} \\ &\subseteq \mathsf{Closure}_\alpha(\mathsf{Post}(Z, e)) \end{aligned}$$

In particular, whenever $\mathsf{Post}(Z, e)$ is empty, $\mathsf{Post}(\mathsf{Closure}_\alpha(Z), e)$ is empty (because $\mathsf{Closure}_\alpha(Z)$ is empty if and only if Z is empty).

In fact, the "First UTA-Algorithm" computes for some consecutive transitions $(e_i)_{i=1\ldots p}$, the zone

$$\mathsf{Closure}_\alpha(\mathsf{Post}(\mathsf{Closure}_\alpha(\mathsf{Post}(\mathsf{Closure}_\alpha(\mathsf{Post}(\ldots, e_{p-2})), e_{p-1})), e_p)).$$

Using iteratively the above relation and the fact that Closure_α is involutive, we get that:

$$\text{Closure}_\alpha(\text{Post}(\text{Closure}_\alpha(\text{Post}(\dots, e_{p-1})), e_p)) \subseteq \text{Closure}_\alpha(\text{Post}(\text{Post}(\dots, e_{p-1}), e_p))$$

Thus, we get, if $R(\langle q_0, Z_0\rangle)$ and $\mathit{MUA}(\langle q_0, Z_0\rangle)$ denote respectively the exact set of reachable states and the set of states computed by the "First UTA-Algorithm",

$$R(\langle q_0, Z_0\rangle) \quad \subseteq \quad \mathit{MUA}(\langle q_0, Z_0\rangle) \quad \subseteq \quad \text{Closure}_\alpha(R(\langle q_0, Z_0\rangle))$$

Thus, $\mathit{MUA}(\langle q_0, Z_0\rangle) = \emptyset \iff R(\langle q_0, Z_0\rangle) = \emptyset$, and this concludes the proof of the theorem. □

Hence, for each decidable updatable timed automaton $\mathcal{A}$, we can test for emptiness of $L(\mathcal{A})$ using Algorithm 2. However, this algorithm can not be implemented easily, but its correctness is fundamental for proving the correctness of other algorithms.

5 Where We Focus on Correctness

We will now extend Algorithm 1 to (decidable) updatable timed automata and study precisely the correctness of this algorithm. We know that a correct widening operator can not be found for every automaton, but we will enlighten what are exactly the problems and bring out subclasses for which such an algorithm can be used.

5.1 The Natural Extension of Algorithm 1 for Decidable Updatable Timed Automata

In this section, we present a forward analysis algorithm for updatable timed automata. This algorithm is parameterized by a function f, which associates to each automaton a set of regions. From this set of regions will depend the widening operator which is used in the algorithm. Of course, we will focus on the class of decidable updatable timed automata, and we want to bring out subclasses of this model for which we can find a function f which leads to a correct algorithm. Our algorithm is then described by Algorithm 3.

Algorithm 3 Algorithm for Updatable Timed Automata

```
# UTA-Algorithm [f] (A:UTA) {
#     Define β as f(A);
#     Visited := ∅;                    (* Visited stores the visited states *)
#     Waiting := {(q0,Approx_β(Z0))};
#     Repeat
#        Get and Remove (q,Z) from Waiting;
#        If q is final
#           then {Return "Yes";}
#           else {If there is no (q,Z') ∈ Visited such that Z ⊆ Z'
#                    then {Visited := Visited ∪ {(q,Z)};
#                          Successor := {(q',Approx_β(Post(Z,e))) |
#                                                   e transition from q to q'};
#                          Waiting := Waiting ∪ Successor;}}
#     Until (Waiting = ∅);
#     Return "No"; }
```

Considering classical timed automata and defining the function f as the one which associates to every classical timed automaton $\mathcal{A}$ the tuple $((k, \ldots, k), (k, \ldots, k))$ where k is the maximum constant appearing in $\mathcal{A}$, Algorithm 3 parametrized by f corresponds to Algorithm 1. As Algorithm 1, Algorithm 3 terminates because there are finitely many overapproximations of zones that can be associated with each control state. From the counter-example of section 3, we know that it is hopeless to find a parameter function f such that for every decidable updatable timed automaton, Algorithm 3 is correct, but our aim is to distinguish classes together with appropriate parameters such that it is the case.

We will first study in details the (partial) correctness of Algorithm 3, and we will then present how the DBM data structure can be used in order to implement Algorithm 3 when this last one is correct.

5.2 Partial Correctness

As we have seen in section 3, Algorithm 3 is not correct in general. However, we will propose some subclasses for which it is correct. We will write the proofs very carefully and with details because, as it will appear, some properties of zones and DBMs are not intuitive at all, and, in the past, mistakes have been done (see section 3), maybe because of this bad intuition we have about zones. We will thus explain where the problems exactly come from.

As we proved that Algorithm 2 is correct with respect to reachability, to prove that Algorithm 3 (parameterized by a function f) is correct for a subclass of timed automata, it is sufficient to prove the following: given a timed automaton $\mathcal{A}$ of this class, α associated to $\mathcal{A}$ as in section 4.2 and $\beta = f(\mathcal{A})$, then for every "computed zone"[5],

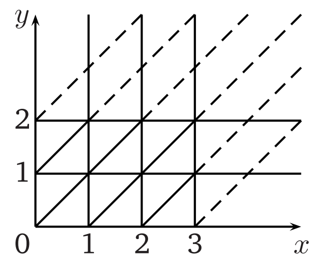
$$\begin{array}{ccccc} Z & \subseteq & \text{Approx}_\beta(Z) & \subseteq & \text{Closure}_\alpha(Z) \\ \uparrow & & \uparrow & & \uparrow \\ \text{Exact set} & & \text{Algorithm 3} & & \text{Algorithm 2} \end{array} \tag{5}$$

because, if it is the case, it means that the approximation Approx_β used in Algorithm 3 is coarser than the closure by region Closure_α used in Algorithm 2. Thus, as Algorithm 2 is correct with respect to reachability, Algorithm 3 will also be correct for the corresponding class of automata.

5.2.1 Diagonal-Free Updatable Timed Automata.

Let $\mathcal{A}$ be a diagonal-free decidable updatable timed automaton. As described in Section 4.2, we associate with $\mathcal{A}$ a diagonal-free set of regions $\mathcal{R}_\alpha$ where $\alpha = (\max_x)_{x\in X}$. We then define a second set of regions, $\mathcal{R}_\beta$ where β is the tuple $((\text{Max}_x)_{x\in X}; (\text{Max}_{x,y})_{x,y\in X})$ with $\text{Max}_{x,y} = \text{Max}_x = \max_x$ for every clocks $x, y \in X$.

Example 9 *The sets of regions $\mathcal{R}_\alpha$ and $\mathcal{R}_\beta$ can be both represented (in a two-dimensional time-space) as on the figure beside, where α is drawn with solid lines whereas the refinement β also uses dashed lines ($\mathcal{R}_\beta$ refines $\mathcal{R}_\alpha$).*



If α and β are defined as above, we have the following property:

Proposition 2 *For each zone Z, the following holds:*

$$Z \subseteq \text{Approx}_\beta(Z) \subseteq \text{Closure}_\alpha(Z).$$

Proof It is equivalent to prove that for every zone Z, for every region R of $\mathcal{R}_\alpha$,

$$R \cap Z = \emptyset \implies R \cap \text{Approx}_\beta(Z) = \emptyset$$

Let $R = ((I_{x_i})_{x_i\in X}; \prec)$ be a region of $\mathcal{R}_\alpha$ and let $M_R = ((r_{i,j}; \prec_{i,j})_{i,j=0\ldots n})$ be

[5] We will make the notion of "computed zone" more precise after.

the DBM representing R such that:

$$\begin{cases} (r_{i,0};\prec_{i,0}) = \begin{cases} (c+1;<) & \text{if } I_{x_i} =]c;c+1[\\ (c;\leq) & \text{if } I_{x_i} = \{c\} \\ (+\infty;<) & \text{if } I_{x_i} =]\max_{x_i};+\infty[\end{cases} \\ (r_{0,i};\prec_{0,i}) = \begin{cases} (-c;<) & \text{if } I_{x_i} =]c;c+1[\\ (-c;\leq) & \text{if } I_{x_i} = \{c\} \\ (-\max_{x_i};<) & \text{if } I_{x_i} =]\max_{x_i};+\infty[\end{cases} \\ (r_{i,j};\prec_{i,j}) = \begin{cases} (c-d+1;<) & \text{if } I_{x_i} =]c;c+1[,\ I_{x_j} =]d;d+1[\\ & \qquad \text{and } x_j \prec x_i,\ x_i \not\prec x_j \\ (c-d;<) & \text{if } I_{x_i} =]c;c+1[,\ I_{x_j} =]d;d+1[\\ & \qquad \text{and } x_i \prec x_j,\ x_j \not\prec x_i \\ (c-d;\leq) & \text{if } I_{x_i} =]c;c+1[,\ I_{x_j} =]d;d+1[\\ & \qquad \text{and } x_i \prec x_j,\ x_j \prec x_i \\ (+\infty;<) & \text{in all other cases} \end{cases} \end{cases}$$

Note that the diagonal-free hypothesis on $\mathcal{R}_\alpha$ has the important consequence that if $(r_{i,0};\prec_{i,0}) = (+\infty;<)$, then for every $j \neq 0$, $(r_{i,j};\prec_{i,j}) = (r_{j,i};\prec_{j,i}) = (+\infty;<)$. That's this property which will be the central point of the proof.

Let Z be a zone and $M = ((m_{i,j};<_{i,j})_{i,j})$ a DBM in normal form that represents Z. Assume that $Z \cap R = \emptyset$. It means that there exists a sequence of distinct indices $(i_1, i_2, \ldots, i_l = i_1)$ such that

$$\alpha_{i_1,i_2} + \alpha_{i_2,i_3} + \cdots + \alpha_{i_{l-1},i_l} < (0;\leq) \tag{6}$$

where $\alpha_{i_j,i_k} = \min((m_{i_j,i_k};<_{i_j,i_k});(r_{i_j,i_k};\prec_{i_j,i_k}))$. As M is in normal form, we can assume that two successive $\alpha_{h,\ell}$ does not come from M, otherwise, we can simplify the sum (6). Note that if $(m_{i_j,i_k};<_{i_j,i_k}) < (r_{i_j,i_k};\prec_{i_j,i_k})$ and if I_{x_i} and I_{x_j} are both bounded, then we can distinguish two cases:

- <u>First case:</u> $(r_{i_j,i_k};\prec_{i_j,i_k}) = (r_{i_j,i_k};\leq) = (-r_{i_k,i_j};\leq) = (-r_{i_k,i_j};\prec_{i_k,i_j})$.

  In this case, $(m_{i_j,i_k};<_{i_j,i_k}) < (-r_{i_k,i_j};\prec_{i_k,i_j})$, thus

  $$(m_{i_j,i_k};<_{i_j,i_k}) + (r_{i_k,i_j};\prec_{i_k,i_j}) < (0;\leq)$$

  Thus, $Z \subseteq (x_{i_j} - x_{i_k} < -r_{i_k,i_j})$ and $R \cap (x_{i_j} - x_{i_k} < -r_{i_k,i_j}) \neq \emptyset$. Thus, we get that $\text{Approx}_\beta(Z) \subseteq (x_{i_j} - x_{i_k} < -r_{i_k,i_j})$ and then that $\text{Approx}_\beta(Z) \cap R = \emptyset$. Note that, obviously, the zone defined by $(x_{i_j} - x_{i_k} < -r_{i_k,i_j})$ is β-bounded because $\mathcal{R}_\beta$ refines $\mathcal{R}_\alpha$.

- <u>Second case:</u> $(r_{i_j,i_k};\prec_{i_j,i_k}) = (c;<)$ and $(r_{i_k,i_j};\prec_{i_k,i_j}) = (-c+1;<)$.

  In this case, $(m_{i_j,i_k};<_{i_j,i_k}) \leq (c-1;<)$. Thus, as previously, we obtain that $\text{Approx}_\beta(Z) \cap R = \emptyset$.

Let us assume now that we are not in one of the two previous cases.
Thus, if $(m_{i_j,i_{j+1}};<_{i_j,i_{j+1}}) < (r_{i_j,i_{j+1}};\prec_{i_j,i_{j+1}})$, it means that either $I_{x_{i_j}} =]\max_{x_{i_j}};+\infty[$ or $I_{x_{i_{j+1}}} =]\max_{x_{i_{j+1}}};+\infty[$. If $I_{x_{i_j}} =]\max_{x_{i_j}};+\infty[$, from the construction of M_R, we get that $i_{j-1} = 0$, because if $i_{j-1} \neq 0$, then $(r_{i_{j-1},i_j};\prec_{i_{j-1},i_j}) = (+\infty;<)$. If $I_{x_{i_{j+1}}} =]\max_{x_{i_{j+1}}};+\infty[$, we get that it is not possible, because every coefficient $r_{i_{j+1},*}$ is infinite. After this analysis, we get that in the sum (6), we can not have two coefficients coming from M. Thus, as R is not empty, we have exactly one coefficient coming from M in the sum (6). Collecting all these informations, we get that there exists a sequence of indices $(k_1,\ldots,k_p)$ such that

$$(r_{0,k_1};\prec_{0,k_1}) + (m_{k_1,k_2};<_{k_1,k_2}) + (r_{k_2,k_3};\prec_{k_2,k_3}) + \cdots + (r_{k_p,0};\prec_{k_p,0}) < (0;\leq)$$

Now, by computing the normal form of M_R, the only coefficients of M_R that can be tightened are the $(r_{x,y};\prec_{x,y})$ where $r_{x,0}$ is bounded. Thus, we can simplify the previous sum and get that

$$(r_{0,k_1};\prec_{0,k_1}) + (m_{k_1,k_2};<_{k_1,k_2}) + (r_{k_2,0};\prec_{k_2,0}) < (0;\leq)$$

Setting $(c;\prec) = (r_{k_2,0};\prec_{k_2,0}) + (r_{0,k_1};\prec_{0,k_1})$, we get that

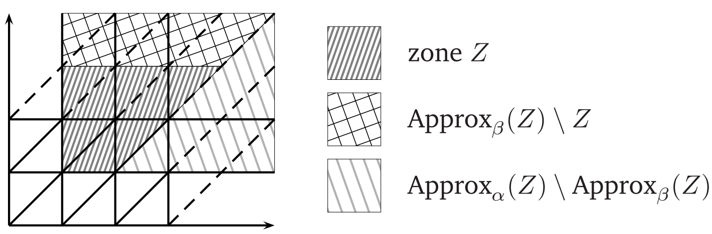
$$\begin{cases} Z \subseteq (x_{k_1} - x_{k_2} \prec -c) \\ R \cap (x_{k_1} - x_{k_2} \prec -c) = \emptyset \end{cases}$$

Thus, as

$$-\mathrm{Max}_{x_{k_1},x_{k_2}} = -\max_{x_{k_1}} \leq c \leq \max_{x_{k_2}} = \mathrm{Max}_{x_{k_2},x_{k_1}}$$

the zone $(x_{k_1} - x_{k_2} \prec -c)$ is β-bounded. As previously, we conclude that $\mathrm{Approx}_\beta(Z) \subseteq (x_{k_1} - x_{k_2} \prec -c)$ and thus that $R \cap \mathrm{Approx}_\beta(Z) = \emptyset$. □

Remark 4 *We can note that if we consider* Approx_α *instead of* Approx_β*, the previous property does not hold anymore. See for example the figure below.*



Using what we said before, we get the following theorem:

Theorem 2 *Algorithm 3 is correct (with respect to reachability) for decidable diagonal-free updatable timed automata (taking as parameter the function defined just before the previous proof).*

This theorem, even if it is not as general as we could expect, is already interesting because in many practical cases, the automata built for real systems are diagonal-free (see as examples [HSLL97] or [BBP02]).

5.2.2 General Updatable Timed Automata.

From the counter-example of section 3, it is hopeless to prove the correctness of Algorithm 3 in the general case. However, we will bring out a subclass of the general updatable timed automata for which we can find a parameter function f such that Algorithm 3 is correct. So, let us take a decidable updatable timed automaton $\mathcal{A}$ and let us consider the set of regions $\mathcal{R}_\alpha$ defined as in section 4.2 where $\alpha = ((\max_x)_{x \in X}; (\max_{x,y})_{x,y \in X})$. The parameter function f associates with $\mathcal{A}$ the tuple $\beta = ((\mathrm{Max}_x)_{x \in X}; (\mathrm{Max}_{x,y})_{x,y \in X})$ where $\mathrm{Max}_x = \mathrm{Max}_{x,y} = \sum_{z \in X} \max_z$ for every clocks $x, y \in X$. Roughly, the constant $\sum_{z \in X} \max_z$ is the maximum constant that can appear in a DBM in normal form that represents a region of $\mathcal{R}_\alpha$.

Theorem 3 *Algorithm 3 is correct (with respect to reachability) for decidable general updatable timed automata that use less than three clocks (the parameter is the one defined just above).*

This theorem is not in contradiction with what we announced in section 3. Indeed, the counter-example of figure 1 uses four clocks. The limit between correctness and non-correctness locates thus precisely between three and four clocks.
Proof The property (5) is equivalent to proving that for every region R of $\mathcal{R}_\alpha$, for every "computed" zone Z, if $R \cap Z = \emptyset$ then $R \cap \mathrm{Approx}_\beta(Z) = \emptyset$. Let $\mathcal{A}$ be a decidable general updatable timed automaton with less than three clocks, as defined in section 4.2. We will prove that, for each "computed" zone Z, the following holds: for each region R of $\mathcal{R}_\alpha$, whenever

$$R \cap Z = \emptyset$$

implies that there exists a β-bounded zone V such that

$$\begin{cases} Z \subseteq V \\ R \cap V = \emptyset \end{cases} \tag{7}$$

This property will imply what we want because $\mathrm{Approx}_\beta(Z)$ is the smallest β-bounded zone that contains the zone Z.
Assume that a zone Z verifies the condition (7) and suppose we are given a clock constraint g and an update up of the automaton $\mathcal{A}$. We will prove that $up(g \cap \overrightarrow{Z})$ also verifies the condition (7). Having proved that, we will deduce that if Z is one of the zones which are computed by the algorithm and if R is any region of $\mathcal{R}_\alpha$, if $Z \cap R = \emptyset$, then $\mathrm{Approx}_\beta(Z) \cap R = \emptyset$ because $\mathrm{Approx}_\beta(Z)$ is the smallest β-bounded zone containing Z.

Computing the future. Assume that $R \cap \overrightarrow{Z} = \emptyset$. It means that $\overleftarrow{R} \cap Z = \emptyset$ where $\overleftarrow{R}$ is the past of R, that is $\overleftarrow{R} = \{v \mid \exists t.\ v + t \in R\}$. Moreover, using the bisimulation property of the set of regions $\mathcal{R}_\alpha$ (see remark 1, page 19), $\overleftarrow{R}$ is a union of regions $\bigcup_i R_i$. For each i, we can find a β-bounded zone V_i such that

$$\begin{cases} Z \subseteq V_i \\ R_i \cap V_i = \emptyset \end{cases}$$

We then define $V = \bigcap_i V_i$. It verifies that

$$\begin{cases} Z \subseteq V \\ \overleftarrow{R} \cap V = \emptyset \end{cases}$$

Let us prove that we can eliminate all the constraints of V which are of the form "x bounded by c". Let $M_R = ((r_{i,j}); \prec_{i,j})_{i,j})$ be a DBM for $\overleftarrow{R}$ such that for each i, $(m_{0,i}; \prec_{0,i})$ is $(0; \leq)$ (see [Ben02], part A). Moreover, assume that V can be rewritten as a DBM $((v_{i,j}; <_{i,j})_{i,j})$. As $\overleftarrow{R} \cap V = \emptyset$, if we assume that a constraint "x bounded by c" is needed in V, it means that there exists $(i_1, i_2, \dots, i_p = 0, i_{p+1}, \dots i_l = i_1)$ such that

$$\alpha_{i_1,i_2} + \dots + \alpha_{i_{p-2},i_{p-1}} + (v_{i_{p-1},0}; <_{i_{p-1},0}) + \alpha_{0,i_{p+1}} + \dots + \alpha_{i_{l-1},i_l} < (0; \leq) \quad (8)$$

where $\alpha_{k,h} = \min((r_{k,h}; \prec_{k,h}); (v_{k,h}; <_{k,h}))$. In particular,

$$\alpha_{0,i_{p+1}} = \min((r_{0,i_{p+1}}; \prec_{0,i_{p+1}}); (v_{0,i_{p+1}}; <_{0,i_{p+1}})) = (v_{0,i_{p+1}}; <_{0,i_{p+1}})$$

because $(r_{0,i_{p+1}}; \prec_{0,i_{p+1}}) = (0; \leq)$. Thus, the sum $(v_{i_{p-1},0}; <_{i_{p-1},0}) + (v_{0,i_{p+1}}; <_{0,i_{p+1}})$ in (8) can be safely replaced by a constraint $(v'_{i_{p-1},i_{p+1}}; <'_{i_{p-1},i_{p+1}}) = (v_{i_{p-1},0}; <_{i_{p-1},0}) + (v_{0,i_{p+1}}; <_{0,i_{p+1}})$. We thus define the zone

$$V' = \bigwedge\{x_i - x_j <_{i,j} v_{i,j} \in V \mid j \neq 0\} \wedge (x_{i_{p-1}} - x_{i_{p+1}} <'_{i_{p-1},i_{p+1}} v'_{i_{p-1},i_{p+1}})$$

which is a β-bounded zone. We get thus that

$$\begin{cases} \overrightarrow{Z} \subseteq V' \\ R \cap V' = \emptyset \end{cases}$$

Intersection with a clock constraint. Assume that g is a clock constraint and that $R \cap (g \cap Z) = \emptyset$. We distinguish two cases:
First case: $R \cap g = \emptyset$, by definition of the regions, there exists a constraint $x - y \prec c$ (constraint "defining α") such that $g \subseteq (x - y \prec c)$ and $R \cap (x - y \prec c) = \emptyset$. Thus, we get that $g \cap Z \subseteq (x - y \prec c)$ and $R \cap (x - y \prec c) = \emptyset$. Note

that as $x - y = c$ is a constraint defining the set of region $\mathcal{R}_\alpha$, $(x - y \prec c)$ is a β-bounded zone.
Second case: $R \subseteq g$. It thus means that $R \cap Z = \emptyset$, and we can use the induction hypothesis to prove that there exists a β-bounded zone V such that $Z \subseteq V$ and $R \cap V = \emptyset$. In particular, $Z \cap g \subseteq V$ and $R \cap V = \emptyset$.

Computing the image by an update. Let up be an update such as the ones of Section 4.2 (general case). It can be written as

$$up = \bigwedge_{x \in Y} (x := y_x) \wedge \bigwedge_{x \in X \setminus Y} (x :\sim c_x).$$

Then the image of a zone by up can be computed by first applying the update $\bigwedge_{x \in Y}(x := y_x)$ (which is equivalent to the update $\bigwedge_{x \in Y}(x := y_x) \wedge \bigwedge_{x \in X \setminus Y}(x := x)$) and then applying successively each update $x :\sim c_x$ (for $x \in X \setminus Y$).

First assume that up is $\bigwedge_{x \in X}(x := y_x)$ and that $R \cap up(Z) = \emptyset$. First of all, if $y_{x_1} = y_{x_2}$, then $up(Z)$ is included in the zone $(x_1 = x_2)$. If for one of such pairs, $R \cap (x_1 = x_2) = \emptyset$, then $up(Z) \subseteq (x_1 = x_2)$ and $(x_1 = x_2)$ is β-bounded. We thus assume that it is not the case. Thus, the set $E = \{R_i \in \mathcal{R}_\alpha \mid up(R_i) \subseteq R\}$ is not empty and we have moreover that $\{v \mid up(v) \in R\} = \bigcup_{R_i \in E} R_i$ (because of the bisimulation property of the set of regions $\mathcal{R}_\alpha$, see remark 1 page 19). For each i, $R_i \cap Z = \emptyset$, thus there exists a β-bounded zone V_i such that $Z \subseteq V_i$ and $V_i \cap R_i = \emptyset$. We distinguish two cases (**we have only three clocks**, say $\{x_1, x_2, x_3\}$):

- First case: $X = \{y_{x_1}, y_{x_2}, y_{x_3}\}$. We define

  $$V = \bigwedge_i \{x_i - x_j \prec m \mid y_{x_i} - y_{x_j} \prec m \in V_i\}$$

  We have that $up(Z) \subseteq V$ and that $R \cap V = \emptyset$, because if $v \in R \cap V$, then we can define safely v' such that $v'(y_{x_i}) = v(x_i)$ for each $i = 1, 2, 3$ and v' will be in $\bigcap_i V_i$, which is not possible because $v = up(v')$ and thus v' would be in one of the R_i's.

- Second case: $Y = \{y_{x_1}, y_{x_2}, y_{x_3}\} \neq X$. We define $R' = R \cap \bigcap_{y_{x_i} = y_{x_j}} (x_i = x_j)$. We have thus that $Z \cap R' = \emptyset$. The zones Z and R' can now be considered as one or two-dimensional zones (depending on the cardinality of Y). For concluding, we will apply the following (obvious) claim to the zones Z and R': Claim Let Z_1 and Z_2 be one or two-dimensional zones such that $Z_1 \cap Z_2 = \emptyset$. Assume that $M_1 = ((m_{i,j}; \prec_{i,j}))$ is a DBM in normal form defining Z_1, then there exists i and j such that

$Z_2 \subseteq \neg(x_i - x_j \prec_{i,j} m_{i,j})$. Note that one of the indices i or j may be 0. There exists thus a constraint $x_i - x_j \prec c$ that is a constraint defining the normal form of R' and such that $Z \subseteq \neg(x_i - x_j \prec c)$. Of course, from the definition of β, we get that the zone $(x_i - x_j \prec c)$ is β-bounded, and defining

$$V = \neg(x_i - x_j \prec c) \wedge \bigwedge_{y_{x_k} = y_{x_l}} (x_k = x_l)$$

we get that $up(Z) \subseteq V$ and $R \cap V = \emptyset$.

Assume that up is $x_k :\prec c$ and that

$$R \cap up(Z) = \emptyset$$

We have that $up(Z)$ is included in $x_k \prec c$, thus, if R is not included in $x_k \prec c$, we have finished because $(x_k \prec c)$ is a β-bounded zone. Assume now that $R \subseteq (x_k \prec c)$. We define $\text{Free}_{x_k}(R)$ as the set $\{v \mid \exists v' \in R \text{ s.t. } \forall x \in X \setminus \{x_k\},\ v(x) = v'(x)\}$. In the same way, we define $\text{Free}_{x_k}(Z)$. By construction of the set of regions $\mathcal{R}_\alpha$ (because of the bisimulation property...), $\text{Free}_{x_k}(R)$ is a union of regions and for each v in $\text{Free}_{x_k}(R)$, there exists $v' \in up(v) \cap R$. Thus, it is easy to see that $\text{Free}_{x_k}(R) \cap \text{Free}_{x_k}(Z) = \emptyset$. **As we only have 3 clocks**, $\text{Free}_{x_k}(Z)$ and $\text{Free}_{x_k}(R)$ can be considered as two-dimensional zones. Applying the previous claim, there exists a constraint $x_i - x_j \prec c$ that does not involve the clock x_k, which is a constraint of the normal form of $\text{Free}_{x_k}(R)$ and such that $\text{Free}_{x_k}(Z) \subseteq \neg(x_i - x_j \prec c)$. The constraint $\neg(x_i - x_j \prec c)$ is of course a β-bounded zone (because of the definition of β), $up(Z) \subseteq \neg(x_i - x_j \prec c)$ and $R \cap \neg(x_i - x_j \prec c) = \emptyset$.

The proof is now finished: we have proved that if Z verifies the property (7), then $up(\overrightarrow{Z} \cap g)$ verifies also (7). Applying the widening operator Approx_β at each step of the computation is thus correct: if R is a region and Z a computed zone, $R \cap Z = \emptyset$ implies that $R \cap \text{Approx}_\beta(Z) = \emptyset$. $\square$

Remark 5 *In the previous proof, the hypothesis on the small number of clocks is used. Indeed, updating a clock is "equivalent" to projecting on a smaller dimension, and when we only have three clocks, the projection is one or two-dimensional and has thus nice properties. An error which maybe explains the numerous mistakes done in the literature is that zones in big dimensions are not similar to what we can "see" in two dimensions, thus it is not correct to project and to visualize what happens in two dimensions. In the previous proof, for the reset, if we want to be general and to reason with n clocks, we get the following: applying the induction hypothesis to the projection of Z and any region "belonging" to $\text{Free}_{x_k}(R)$ will allow to get a β-bounded zone V which "separates" them* *(in the sense that $Z \subseteq V$ and Free$_{x_k}(R) \cap V = \emptyset$). However, we have no argument to say that this β-bounded zone does not use the clock x_k. Thus, from V, it is not possible to compute a β-bounded zone that "separates" $up(Z)$ and R. That's the reason why the theorem 3 is not true in general.*

5.3 Implementation of the Algorithm

From what precedes, we have that, even if it is not correct in general, Algorithm 3 can however be used safely for some subclasses of systems. We will thus show that the DBM data structure is appropriate to implement Algorithm 3: we will show how to compute the new operations on zones appearing in Algorithm 3 using the DBMs. New operations are of two types: image by updates and widening operator Approx$_\beta$.

Image by updates. Let up be an update and M be a DBM (in normal form). The DBM M represents a clock constraint φ over the set of clocks X. From up, we construct a clock constraint φ_{up} over the set of clocks $X \cup X'$ (where X' is a disjoint copy of X) by induction on up in the following manner:

$$\begin{cases} \varphi_{x:\sim c} = x' \sim c \\ \varphi_{x:\sim y+c} = x' \sim y + c \\ \varphi_{up_1 \wedge up_2} = \varphi_{up_1} \wedge \varphi_{up_2}. \end{cases}$$

We now define the formula $\varphi'_{up} = \varphi \wedge \varphi_{up}$. This formula represents the update up in the sense that if V is a valuation for the clocks $X \cup X'$, then

$$V \models \varphi'_{up} \iff V_{|X} \models \varphi \text{ and } V_{|X'} \in up(V_{|X}).$$

We assume that $up = \bigwedge_{i=1}^{n} up_i$ where up_i updates the clock x_i. We define $N = (\eta_{i,j})_{i,j=0\dots 2n}$ as the DBM of size $2n+1$ (over the set of clocks $X \cup X'$) as follows:

- for $i, j = 0 \dots n$, $\eta_{i,j} = (m_{i,j}; \prec_{i,j})$,
- for each update up_i ($1 \leq i \leq n$), we distinguish the different possible values of up_i:

if up_i is:	then we set:
$x_i := c$	$\eta_{i+n,0} = (c, \leq)$ and $\eta_{0,i+n} = (-c; \leq)$
$x_i := x_j + c$	$\eta_{i+n,j} = (c; \leq)$ and $\eta_{j,i+n} = (-c; \leq)$
$x_i :\prec c$ with $\prec \in \{<; \leq\}$	$\eta_{i+n,0} = (c; \prec)$ and $\eta_{0,i+n} = (0; \leq)$
$x_i :\succ c$ with $\succ \in \{>; \geq\}$	$\eta_{0,i+n} = (-c; \prec)$ (where $\prec$ is $<$ if $\succ$ is $>$ and $\prec$ is $\leq$ if $\succ$ is $\geq$)
$x_i :\prec x_j + c$ with $\prec \in \{<; \leq\}$	$\eta_{i+n,j} = (c; \prec)$ and $\eta_{j,i+n} = (\infty; <)$
$x_i :\succ x_j + c$ with $\succ \in \{>; \geq\}$	$\eta_{j,i+n} = (-c; \prec)$ (where $\prec$ is $<$ if $\succ$ is $>$ and $\prec$ is $\leq$ if $\succ$ is $\geq$)

If up_i is a conjunction of simple updates, we take the smallest constraints,

- for every $i, j = 0 \ldots 2n+1$, if $\eta_{i,j}$ has not been initialized yet , we set $\eta_{i,j} = (+\infty; <)$.

By construction, the DBM N represents the clock constraint φ'_{up} (indices $1 \leq i \leq n$ correspond to the set of clocks X whereas indices $n+1 \leq i \leq 2n+1$ correspond to the set of clocks X'):

$$[\![N]\!] = \{V \text{ valuation over } X \cup X' \mid V \models \varphi'_{up}\}.$$

We define $(\phi(N))_{|X'}$ as the square submatrice of size $n+1$ of N where we erase everything about the set of clocks X. It can be represented by the picture beside (the white part correspond to the clocks from X). Then we get that:

$$[\![(\phi(N))_{|X'}]\!] = up([\![M]\!]).$$

$$\left(\begin{array}{c|ccc|ccc} * & * & \ldots & * & * & \ldots & * \\ \hline * & * & \ldots & * & * & \ldots & * \\ \vdots & & & & & & \\ * & * & \ldots & * & * & \ldots & * \\ \hline * & * & \ldots & * & * & \ldots & * \\ \vdots & & & & & & \\ * & * & \ldots & * & * & \ldots & * \end{array}\right)$$

If we want to compute the image by the update up of a zone Z represented by a DBM M (in normal form), we first compute the DBM N, then its normal form and finally we erase some coefficients and the result of this algorithm gives a DBM (in normal form) that represents $up(Z)$.

β-overapproximation. Let $\beta = ((\max_x)_{x \in X}; (\max_{x,y})_{x,y \in X})$ defining a set of regions. All the β which are used in one of the correct algorithms described before verifies in addition that for every $x, y \in X$, $\max_{x,y} = \max_x$, We assume

$X = \{x_1, \ldots, x_n\}$. A DBM $M = (m_{i,j}; \prec_{i,j})_{i,j}$ is said *β-bounded* if:

$$\begin{cases} 0 \leq m_{i,0} \leq \max_{x_i} \text{ or } m_{i,0} = \infty & \text{for every } 1 \leq i \leq n \\ -\max_{x_i} \leq m_{0,i} \leq 0 & \text{for every } 1 \leq i \leq n \\ -\max_{x_j,x_i} \leq m_{i,j} \leq \max_{x_i,x_j} \text{ or } m_{i,j} = \infty & \text{for every } 1 \leq i,j \leq n \end{cases}$$

Lemma 3 *Let Z be a zone. Then*

$$Z \in \mathcal{Z}_\beta \iff \textit{there exists a } \beta\textit{-bounded DBM } M_\beta \textit{ such that } Z = [\![M_\beta]\!].$$

➜ *See the proof in Appendix.*

We consider the following algorithm (we assume $M = (m_{i,j}; \prec_{i,j})$ and $M' = (m'_{i,j}; \prec'_{i,j})$):

β-Overapproximation Algorithm (M:n-dimensional DBM)
M' = ϕ(M); (* Normal Form of M *)
for i = 1 to n do
for j = 1 to n do
if (m'$_{i,j}$;$\prec_{i,j}$)>(max$_{x_i,x_j}$;$\leq$) then (m'$_{i,j}$;$\prec_{i,j}$) becomes (+∞;<);
if (m'$_{i,j}$;$\prec_{i,j}$)<(-max$_{x_j,x_i}$;<) then
(m'$_{i,j}$;$\prec_{i,j}$) becomes (-max$_{x_j,x_i}$;<);
endfor;
if (m'$_{i,0}$;$\prec_{i,0}$)>(max$_{x_i}$;$\leq$) then (m'$_{i,0}$;$\prec_{i,0}$) becomes (+∞;<);
if (m'$_{0,i}$;$\prec_{0,i}$)<(-max$_{x_i}$;<) then (m'$_{0,i}$;$\prec_{0,i}$) becomes (-max$_{x_i}$,<);
endfor;
Return M';

If M is a DBM, applying this algorithm to M, we obtain a DBM that we denote by $\phi_\beta(M)$. We can note that $\phi_\beta(M)$ is a β-bounded DBM and thus that $[\![\phi_\beta(M)]\!]$ is in $\mathcal{Z}_\beta$.

Property 1 *If M is a DBM, then it holds that $[\![\phi_\beta(M)]\!] = \mathrm{Approx}_\beta([\![M]\!])$.*

➜ *See the proof in Appendix.*

The "β-Overapproximation Algorithm" applied to M computes thus a DBM that represents the β-overapproximation of $[\![M]\!]$. This DBM might not be in normal form. The complexity of the computation of such a β-overapproximation is not high (it is in $\mathcal{O}(n^3)$ if n is the number of clocks).

Combining the constructions of section 2.4.1 with what precedes, we get that if $e = q \xrightarrow{g,a,up} q'$ is a transition and if M is a DBM representing the zone Z, we can compute a DBM M' which represents the zone $\mathrm{Approx}_\beta(up(g \cap \overrightarrow{Z})) =$

$\text{Approx}_\beta(\mathsf{Post}(Z, e))$ but which might not be in normal form. Normal forms are computed before performing all the operations on DBMs, apart from the intersection for which it is not required, and apart from when we just performed either an image by update or a future operation (after such operations, DBMs are automatically in normal form). Thus, each operation in Algorithm 3 can be computed using the DBM data structure, which is thus appropriate to the implementation of this algorithm. Moreover, the complexity of each step of the algorithm is, as for classical timed automata, polynomial in the number of clocks.

5.4 Discussion

The automata for which we have troubles with the widening operator have diagonal clock constraints (and more than four clocks). We will propose two methods for alleviating this problem. Of course, these solutions will *a priori* not be as efficient as Algorithm 3.

5.4.1 First Proposition with a Preprocessing Step for Some Timed Automata.

This method does concern the decidable updatable timed automata with general clock constraints and deterministic updates: we allow updates of the form $x := y$ and $x := c$ where x and y are clocks and c is a constant. We know that every (classical) timed automaton with general clock constraints can be transformed into a (classical) diagonal-free timed automaton [BDGP98]. In the same way, every updatable timed automaton that uses only updates of the form $x := y$ and $x := c$ can be transformed into a diagonal-free updatable timed automaton that uses also updates of the form $x := y$ and $x := c$. This transformation can not be done if we use non-deterministic updates. As we know, this transformation suffers from a combinatorics explosion. However, if the number of diagonal constraints used in the automaton is small, the size of the automaton obtained after this preprocessing step will not be too big. Thus, Algorithm 3 can then be applied to the resulting diagonal-free automaton.

5.4.2 Second Proposition.

The last method we propose can be used for every decidable updatable timed automaton. In practice, it may be expensive, in memory as well as in time.

Let $\mathcal{A}$ be an updatable timed automaton. We associate with $\mathcal{A}$ the set of regions $\mathcal{R}_\alpha$ as in section 4.2. We assume that $\alpha = ((\max_x)_{x\in X}; (\max_{x,y})_{x,y\in X})$ and we define the tuple $\beta = ((\text{Max}_x)_{x\in X}; (\text{Max}_{x,y})_{x,y\in X})$ such that $\text{Max}_x = \text{Max}_{x,y} = \max_x$ for every $x, y \in X$.

Let R be a region of $\mathcal{R}_\alpha$ and Z be a zone. We assume that $M = ((m_{i,j}; \prec_{i,j})_{i,j})$ is a DBM in normal form for Z and $M_R = ((r_{i,j}; <_{i,j})_{i,j})$ is a DBM that represents R (computed directly from its definition using intervals, like in section 4.2.1, without tightening any constraint). Assume now that $Z \cap R = \emptyset$. It means that there exists a sequence of distincts indices $(i_1, \ldots, i_l = i_1)$ such that

$$\alpha_{i_1,i_2} + \cdots + \alpha_{i_{l-1},i_l} < (0; \leq) \tag{9}$$

where $\alpha_{h,\ell} = \min((r_{h,\ell}; \prec_{h,\ell}); (m_{h,\ell}; <_{h,\ell}))$ and two consecutives $\alpha_{h,\ell}$ do not belong to M (this is posssible as M is in normal form).

The idea will be then to ensure that there is just one of the coefficient $\alpha_{i_j,i_{j+1}}$ which comes from the DBM M_R. It will be the case whenever we have that, for every pair (i,j) where $i.j \neq 0$, there exists an interval $I_{i,j} \in \mathcal{I}_{i,j}$ where

$$\begin{aligned} \mathcal{I}_{i,j} \quad = \quad & \{]-\infty; -\max_{x_j}[,]\max_{x_i}; +\infty[\} \cup \{[c] \mid -\max_{x_j} \leq c \leq \max_{x_i}\} \\ & \cup \{]c; c+1[\mid -\max_{x_j} \leq c < \max_{x_i}\} \end{aligned}$$

such that $Z \subseteq I_{i,j}$. In that case, for every pair (i,j) such that $i.j \neq 0$, $(m_{i,j}; \prec_{i,j}) \leq (r_{i,j}; <_{i,j})$. Thus inequality (9) can be transformed into:

$$\alpha_{i,0} + \alpha_{0,j} + (m_{j,i}; \prec_{j,i}) < (0; \leq)$$

We can distinguish three cases:

1. $\alpha_{i,0} = (m_{i,0}; \prec_{i,0})$ and $\alpha_{0,j} = (r_{0,j}; <_{0,j})$.

   In this case, we get that $(r_{0,j}; <_{0,j}) + (m_{j,0}; \prec_{j,0}) < (0; \leq)$. Thus, $Z \subseteq \neg(-x_j <_{0,j} r_{0,j})$ and $R \subseteq (-x_j <_{0,j} r_{0,j})$

2. $\alpha_{i,0} = (r_{i,0}; <_{i,0})$ and $\alpha_{0,j} = (m_{0,j}; \prec_{0,j})$.

   In this case, we get that $(r_{i,0}; <_{i,0}) + (m_{0,i}; \prec_{0,i}) < (0; \leq)$. Thus, $Z \subseteq \neg(x_i <_{i,0} r_{i,0})$ and $R \subseteq (x_i <_{i,0} r_{i,0})$.

3. $\alpha_{i,0} = (r_{i,0}; <_{i,0})$ and $\alpha_{0,j} = (r_{0,j}; <_{0,j})$.

   In this case, we get that $(r_{i,0}; <_{i,0}) + (r_{0,j}; <_{0,j}) + (m_{j,i}; \prec_{j,i}) < (0; \leq)$. Thus, $Z \subseteq \neg(x_i - x_j <' c)$ and $R \subseteq (x_i - x_j <' c)$ where $(c; <') = (r_{i,0}; <_{i,0}) + (r_{0,j}; <_{0,j})$

In all the cases, there exists a β-bounded zone V such that

$$\begin{cases} Z \subseteq V \\ R \cap V = \emptyset \end{cases}$$

The solution we propose is thus to decompose a "computed zone" in the following way:

$$Z = \bigsqcup \{Z \cap \bigcap_{i,j} I_{i,j} \mid I_{i,j} \in \mathcal{I}_{i,j}\}$$

and to compute the image by Approx_β of each such sub-zone of Z. Note that an idea of splitting zones was already presented in [Bou00, Ben02], but it was a bit different, and, here, it appears as some kind of "necessary" condition for having a correct algorithm.

A work direction is now to look for more efficient algorithms that test for emptiness timed automata with diagonal clock constraints. Maybe backward analysis algorithms would be more appropriate to this model...

6 Conclusion

In this paper, we studied the widening operator proposed in [DT98] to overcome the non-termination problem of the forward analysis algorithm for timed automata. This operator is very nice in that it can be very easily computed using the standard DBM data structure. However, we have proved in this paper that it is hopeless to look for a correct forward analysis algorithm that uses such an operator: there exist timed automata for which no widening operator is correct. This result is very surprising because several (attempts of) correctness proofs can be found in the literature and the algorithm is implemented and used in model-checking tools like KRONOS and UPPAAL.

Having stated this result, we took the more general setting of updatable timed automata and brought out subclasses of this model for which we can propose a correct widening operator. Among these subclasses, we can find all the decidable *diagonal-free* updatable timed automata, and in particular all the classical timed automata which do not use diagonal clock constraints. This class of timed automata is sufficiently large, and that's maybe the reason why the bug has not been detected before. For the timed automata for which no correct widening operator can be used, other methods are proposed.

This work is also the link that was missing between the theoretical work we did on this updatable timed automata model in [BDFP00a, BDFP00b] (namely decidability and expressiveness) and a tool that would deal with updatable timed automata. If we restrict to diagonal-free systems, updates can be safely added without more difficulty than implementing the image by updates of zones. Considering general updatable timed automata, some problems appear when we want to implement them. However, the problems come from the use of diagonal clock constraints and not from the updates which are used.

Acknowledgments: I would like to thank François Laroussinie for interesting discussions on DBMs, Antoine Petit and Béatrice Bérard for their careful reading of some drafts of this paper and Kim G. Larsen, Emmanuel Fleury and Gerd Behrmann for discussions on the subject during our so-called "UPPAAL meetings". I also want to thank Sergio Yovine as well as anonymous referees for helpful comments about this work.

References

[ACD+92] Rajeev Alur, Costas Courcoubetis, David Dill, Nicolas Halbwachs, and Howard Wong-Toi. Minimization of timed transition systems. In *Proc. 3rd International Conference on Concurrency Theory (CONCUR'92)*, volume 630 of *Lecture Notes in Computer Science*, pages 340–354. Springer, 1992.

[ACH94] Rajeev Alur, Costas Courcoubetis, and Thomas A. Henzinger. The observational power of clocks. In *Proc. 5th International Conference on Concurrency Theory (CONCUR'94)*, volume 836 of *Lecture Notes in Computer Science*, pages 162–177. Springer, 1994.

[AD90] Rajeev Alur and David Dill. Automata for modeling real-time systems. In *Proc. 17th International Colloquium on Automata, Languages and Programming (ICALP'90)*, volume 443 of *Lecture Notes in Computer Science*, pages 322–335. Springer, 1990.

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

[AFH94] Rajeev Alur, Limor Fix, and Thomas A. Henzinger. A determinizable class of timed automata. In *Proc. 6th International Conference on Computer Aided Verification (CAV'94)*, volume 818 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 1994.

[Alu99] Rajeev Alur. Timed automata. In *Proc. 11th International Conference on Computer Aided Verification (CAV'99)*, volume 1633 of *Lecture Notes in Computer Science*, pages 8–22. Springer, 1999. Invited talk.

[BBP02] Béatrice Bérard, Patricia Bouyer, and Antoine Petit. Analysing the PGM protocol with UPPAAL. In *Proc. 2nd Workshop on Real-Time Tools (RT-TOOLS'02)*, 2002. Proc. published as Technical Report 2002-025, Uppsala University, Sweden.

[BDFP00a] Patricia Bouyer, Catherine Dufourd, Emmanuel Fleury, and Antoine Petit. Are timed automata updatable? In *Proc. 12th International Conference on Computer Aided Verification (CAV'2000)*, volume 1855 of *Lecture Notes in Computer Science*, pages 464–479. Springer, 2000.

[BDFP00b] Patricia Bouyer, Catherine Dufourd, Emmanuel Fleury, and Antoine Petit. Expressiveness of updatable timed automata. In *Proc. 25th International Symposium on Mathematical Foundations of Computer Science (MFCS'2000)*, volume 1893 of *Lecture Notes in Computer Science*, pages 232–242. Springer, 2000.

[BDGP98] Béatrice Bérard, Volker Diekert, Paul Gastin, and Antoine Petit. Characterization of the expressive power of silent transitions in timed automata. *Fundamenta Informaticae*, 36(2–3):145–182, 1998.

[Ben02] Johan Bengtsson. *Clocks, DBMs ans States in Timed Systems*. PhD thesis, Uppsala University, Uppsala, Sweden, 2002.

[BF99] Béatrice Bérard and Laurent Fribourg. Automated verification of a parametric real-time program: the ABR conformance protocol. In *Proc. 11th International Conference on Computer Aided Verification (CAV'99)*, volume 1633 of *Lecture Notes in Computer Science*, pages 96–107. Springer, 1999.

[BFH$^+$01] Gerd Behrmann, Ansgar Fehnker, Thomas Hune, Kim G. Larsen, Paul Pettersson, Judi Romijn, and Frits Vaandrager. Minimum-cost reachability for priced timed automata. In *Proc. 4th International Workshop on Hybrid Systems: Computation and Control (HSCC'01)*, volume 2034 of *Lecture Notes in Computer Science*, pages 147–161. Springer, 2001.

[BL96] Johan Bengtsson and Fredrik Larsson. UPPAAL, a tool for automatic verification of real-time systems. Master's thesis, Department of Computer Science, Uppsala University, Sweden, 1996.

[BLP$^+$99] Gerd Behrmann, Kim G. Larsen, Justin Pearson, Carsten Weise, and Wang Yi. Efficient timed reachability analysis using clock difference diagrams. In *Proc. 11th International Conference on Computer Aided Verification (CAV'99)*, volume 1633 of *Lecture Notes in Computer Science*, pages 341–353. Springer, 1999.

[Bou00] Patricia Bouyer. A new algorithm to decide emptiness of updatable timed automata. Research Report LSV–00–9, Laboratoire Spécification et Vérification, ENS de Cachan, France, 2000.

[BTY97] Ahmed Bouajjani, Stavros Tripakis, and Sergio Yovine. On-the-fly symbolic model-checking for real-time systems. In *Proc. 18th IEEE Real-Time Systems Symposium (RTSS'97)*, pages 25–35. IEEE Computer Society Press, 1997.

[CG00] Christian Choffrut and Massimiliano Goldwurm. Timed automata with periodic clock constraints. *Journal of Automata, Languages and Combinatorics (JALC)*, 5(4):371–404, 2000.

[CGP99] Edmund Clarke, Orna Grumberg, and Doron Peled. *Model-Checking*. The MIT Press, Cambridge, Massachusetts, 1999.

[Daw97] Conrado Daws. Analyse par simulation symbolique des systèmes temporisés avec KRONOS. Research report, Verimag, 1997.

[Dil89] David Dill. Timing assumptions and verification of finite-state concurrent systems. In *Proc. of the Workshop on Automatic Verification Methods for Finite State Systems*, volume 407 of *Lecture Notes in Computer Science*, pages 197–212. Springer, 1989.

[DOTY96] Conrado Daws, Alfredo Olivero, Stavros Tripakis, and Sergio Yovine. The tool KRONOS. In *Proc. Hybrid Systems III: Verification and Control (1995)*, volume 1066 of *Lecture Notes in Computer Science*, pages 208–219. Springer, 1996.

[DT98] Conrado Daws and Stavros Tripakis. Model-checking of real-time reachability properties using abstractions. In *Proc. 4th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'98)*, volume 1384 of *Lecture Notes in Computer Science*, pages 313–329. Springer, 1998.

[DZ98] François Demichelis and Wieslaw Zielonka. Controlled timed automata. In *Proc. 9th International Conference on Concurrency Theory (CONCUR'98)*, volume 1466 of *Lecture Notes in Computer Science*, pages 455–469. Springer, 1998.

[HHWT97] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. HYTECH: A model-checker for hybrid systems. *Journal on Software Tools for Technology Transfer (STTT)*, 1(1–2):110–122, 1997.

[HKWT95] Thomas A. Henzinger, Peter W. Kopke, and Howard Wong-Toi. The expressive power of clocks. In *Proc. 22nd International Colloquium on Automata, Languages and Programming (ICALP'95)*, volume 944 of *Lecture Notes in Computer Science*, pages 417–428. Springer, 1995.

[HRS98] Thomas A. Henzinger, Jean-François Raskin, and Pierre-Yves Schobbens. The regular real-time languages. In *Proc. 25th International Colloquium on Automata, Languages and Programming (ICALP'98)*, volume 1443 of *Lecture Notes in Computer Science*, pages 580–591. Springer, 1998.

[HSLL97] Klaus Havelund, Arne Skou, Kim G. Larsen, and Kristian Lund. Formal modeling and analysis of an audio/video protocol: An industrial case study using UPPAAL. In *Proc. 18th IEEE Real-Time Systems Symposium (RTSS'97)*, pages 2–13. IEEE Computer Society Press, 1997.

[LPY97] Kim G. Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a nutshell. *Journal of Software Tools for Technology Transfer (STTT)*, 1(1–2):134–152, 1997.

[MP99] Supratik Mukhopadhyay and Andreas Podelski. Beyond region graphs: Symbolic forward analysis of timed automata. In *Proc. 19th Conference on Foundations of Software Technology and Theoretical Computer Science (FST&TCS'99)*, volume 1738 of *Lecture Notes in Computer Science*, pages 232–244. Springer, 1999.

[Tri98] Stavros Tripakis. *L'analyse formelle des systèmes temporisés en pratique*. PhD thesis, Université Joseph Fourier, Grenoble, France, 1998.

[TY98] Stavros Tripakis and Sergio Yovine. Verification of the fast reservation protocol with delayed transmission using the tool KRONOS. In *Proc. 4th IEEE Real-Time Technology and Applications Symposium (RTAS'98)*, pages 165–170. IEEE Computer Society Press, 1998.

[TY01] Stavros Tripakis and Sergio Yovine. Analysis of timed systems using time-abstracting bisimulations. *Formal Methods in System Design*, 18(1):25–68, 2001.

[Wil94] Thomas Wilke. Specifying timed state sequences in powerful decidable logics and timed automata. In *Proc. 3rd International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'94)*, volume 863 of *Lecture Notes in Computer Science*, pages 694–715. Springer, 1994.

[WT94] Howard Wong-Toi. *Symbolic Approximations for Verifying Real-Time Systems*. PhD thesis, Stanford University, USA, 1994.

[YL97] Mihalis Yannakakis and David Lee. An efficient algorithm for minimizing real-time transition systems. *Formal Methods in System Design*, 11(2):113–136, 1997.

[Yov98] Sergio Yovine. Model-checking timed automata. In *School on Embedded Systems*, volume 1494 of *Lecture Notes in Computer Science*, pages 114–152. Springer, 1998.

Appendix: Technical Proofs of this Paper

4.3 A First Correct Algorithm for Decidable Updatable Timed Automata

Lemma 1 *The two following conditions hold:*

$$\left\{\begin{array}{ll} \text{Closure}_\alpha(g) = g & \textit{for each clock constraint } g \textit{ in } \mathcal{C} \\ up(\text{Closure}_\alpha(Z)) \subseteq \text{Closure}_\alpha(up(Z)) & \textit{for each zone } Z \textit{ and for each update } up \textit{ in } \mathcal{U} \end{array}\right. \tag{4}$$

Proof Let g be in $\mathcal{C}$. Then $\text{Closure}_\alpha(g) = \bigcup\{R \in \mathcal{R}_\alpha \mid R \cap g \neq \emptyset\} = g$ because for each region $R \in \mathcal{R}_\alpha$, if $R \cap g \neq \emptyset$ then $R \subseteq g$.
We will now prove the second condition, namely that for each zone Z, for each update $up \in \mathcal{U}$,

$$up(\text{Closure}_\alpha(Z)) \subseteq \text{Closure}_\alpha(up(Z)) \tag{5}$$

Let us take a zone Z. We can write Z as $\bigcup_f Z_i$ where there exists regions R_i such that $Z_i = Z \cap R_i$ and Z_i is not empty. If we prove condition (5) for each zone Z_i, then condition (5) will also hold for Z:

$$\begin{array}{rcll} up(\text{Closure}_\alpha(Z)) & = & up(\text{Closure}_\alpha(\cup_f Z_i)) & \\ & = & up(\cup_f \text{Closure}_\alpha(Z_i)) & \\ & = & \cup_f up(\text{Closure}_\alpha(Z_i)) & \\ & \subseteq & \cup_f \text{Closure}_\alpha(up(Z_i)) & \text{if condition (5) holds for each } Z_i \\ & \subseteq & \text{Closure}_\alpha(\cup_f up(Z_i)) & \\ & \subseteq & \text{Closure}_\alpha(up(\cup_f Z_i)) & \\ & \subseteq & \text{Closure}_\alpha(up(Z)) & \end{array}$$

Thus, we just have to prove the condition (5) for zones included in a region (defined by α).
Let Z be a zone included in a region. We want to prove that

$$up(\text{Closure}_\alpha(Z)) \subseteq \text{Closure}_\alpha(up(Z))$$

Thus, let us assume that there exists a valuation $v' \in up(\text{Closure}_\alpha(Z)) \setminus \text{Closure}_\alpha(up(Z))$. There exists $v \in \text{Closure}_\alpha(Z)$ such that $v' \in up(v)$. Thus, we obtain that $up(v) \cap [v']_\alpha \neq \emptyset$. Let v_0 be in Z, then $[v]_\alpha = [v_0]_\alpha$ and thus, by hypothesis, $up(v_0) \cap [v']_\alpha \neq \emptyset$ and thus $[v']_\alpha \subseteq \text{Closure}_\alpha(up(Z))$. This is a contradiction to the fact that $v' \notin \text{Closure}_\alpha(up(Z))$. □

Lemma 2 *For each zone Z, $\overrightarrow{\text{Closure}_\alpha(Z)} \subseteq \text{Closure}_\alpha(\overrightarrow{Z})$ and for each clock constraint g such that $\text{Closure}_\alpha(g) = g$, $g \cap \text{Closure}_\alpha(Z) \subseteq \text{Closure}_\alpha(g \cap Z)$.*

Proof Let us take $v_0 \in \overrightarrow{\text{Closure}_\alpha(Z)}$. There exists some $v_1 \in \text{Closure}_\alpha(Z)$ and some $t \geq 0$ such that $v_0 = v_1 + t$. By definition of Closure_α, there exists some $v_2 \in Z$ such that $[v_1]_\alpha = [v_2]_\alpha$. Thus, $[v_0]_\alpha$ is a time successor of $[v_2]_\alpha$: there exists $v_3 \in [v_0]_\alpha$ and $t' \geq 0$ such that $v_3 = v_2 + t'$. Thus, $v_3 \in \overrightarrow{Z}$ and it follows that $v_0 \in \text{Closure}_\alpha(\overrightarrow{Z})$ because $[v_3]_\alpha = [v_0]_\alpha$. Thus, we get the property we wanted.

The second property holds because of the relation $\text{Closure}_\alpha(g) = g$: let us take some $v_0 \in g \cap \text{Closure}_\alpha(Z)$. We have that $[v_0]_\alpha \subseteq g$ and that there exists $v_1 \in [v_0]_\alpha \cap Z$. We obtain that $v_1 \in g \cap Z$ and thus that $v_0 \in \text{Closure}_\alpha(g \cap Z)$. □

5.3 Implementation of the Algorithm

Lemma 3 *Let Z be a zone. Then*

$$Z \in \mathcal{Z}_\beta \iff \textit{there exists a } \beta\textit{-bounded DBM } M_\beta \textit{ such that } Z = [\![M_\beta]\!].$$

Proof If M_β is a β-bounded DBM, then $[\![M_\beta]\!]$ is in $\mathcal{Z}_\beta$.
A region $R \in \mathcal{R}_\beta$ is β-bounded. Assume Z is the convex union of two zones Z_1 and Z_2 which belong to $\mathcal{Z}_\beta$ and such that there exists two β-bounded DBMs M_{Z_1} and M_{Z_2} such that $Z_1 = [\![M_{Z_1}]\!]$ and $Z_2 = [\![M_{Z_2}]\!]$. We assume that for $l = 1, 2$, $M_{Z_l} = (m^{(l)}_{i,j}; \prec^{(l)}_{i,j})_{i,j=1\ldots n}$ and we construct $M = (m_{i,j}; \prec_{i,j})_{i,j=1\ldots n}$ such that:

$$(m_{i,j}; \prec_{i,j}) = \max((m^{(1)}_{i,j}; \prec^{(1)}_{i,j}), (m^{(2)}_{i,j}; \prec^{(2)}_{i,j})).$$

We first note that M is a β-bounded DBM. We want to prove that $[\![M]\!] = Z$. We note first that for $l = 1, 2$, $M_{Z_l} \leq M$, thus $Z_l \subseteq [\![M]\!]$ and this implies that $Z \subseteq [\![M]\!]$.
Let r be a real such that $-m_{j,i} \prec_{j,i} r \prec_{i,j} m_{i,j}$. There exists k such that $(m_{i,j}; \prec_{i,j}) = (m^{(k)}_{i,j}; \prec^{(k)}_{i,j})$. We have that $(m^{(k)}_{j,i}; \prec^{(k)}_{j,i}) \leq (m_{j,i}; \prec_{j,i})$, thus $-m^{(k)}_{j,i} \prec^{(k)}_{j,i} r \prec^{(k)}_{i,j} m^{(k)}_{i,j}$. This implies that there exists a valuation $v \in Z_k$ such that $v(i) - v(j) = r$. As a consequence, we can not tighten any constant in M. Thus, $Z = [\![M]\!]$. This concludes the proof. □

Property 1 *If M is a DBM, then it holds that $[\![\phi_\beta(M)]\!] = \text{Approx}_\beta([\![M]\!])$.*

Proof Let M be a DBM in normal form. We get immediately that $\text{Approx}_\beta([\![M]\!]) \subseteq [\![\phi_\beta(M)]\!]$ because $\text{Approx}_\beta([\![M]\!])$ is the smallest zone from $\mathcal{Z}_\beta$ that contains $[\![M]\!]$ and $[\![\phi_\beta(M)]\!]$ is such a zone.
There exists a β-bounded DBM M_β such that $\text{Approx}_\beta([\![M]\!]) = [\![M_\beta]\!]$. We get that $M \leq M_\beta$. Assume that $M = (m_{i,j}; \prec_{i,j})_{i,j=1\dots n}$ and that $M_\beta = (m^{(\beta)}_{i,j}; \prec^{(\beta)}_{i,j})_{i,j=1\dots n}$. We will prove, coefficient by coefficient, that $\phi_\beta(M) \leq M_\beta$.
As a first example, assume that $(m_{i,0}; \prec_{i,0}) > (\max_i; \leq)$ and that $(m_{i,j}; \prec_{i,j}) > (\max_{i,j}; \leq)$, then $M \leq M_\beta$ implies that $(m^{(\beta)}_{i,j}; \prec^{(\beta)}_{i,j}) = (\infty; <)$.
As an other example, assume that $(m_{i,0}; \prec_{i,0}) > (\max_i; \leq)$ and that $(m_{i,j}; \prec_{i,j}) < (-\max_{j,i}; <)$, then $M \leq M_\beta$ implies that $(m^{(\beta)}_{i,j}; \prec^{(\beta)}_{i,j}) \geq (-\max_{j,i}; <)$.
We don't write the many other cases, but, in this way, we prove that $\phi_\beta(M) \leq M_\beta$, which implies that $[\![\phi_\beta(M)]\!] \subseteq \text{Approx}_\beta([\![M]\!])$. This concludes the proof. $\square$

Recent BRICS Report Series Publications

RS-02-35 **Patricia Bouyer.** ***Timed Automata May Cause Some Troubles.*** **August 2002. 44 pp.**

RS-02-34 **Morten Rhiger.** ***A Foundation for Embedded Languages.*** **August 2002. 29 pp.**

RS-02-33 **Vincent Balat and Olivier Danvy.** ***Memoization in Type-Directed Partial Evaluation.*** **July 2002. 18 pp. To appear in Batory and Consel, editors,** ***ACM SIGPLAN/SIGSOFT Conference on Generative Programming and Component Engineering,*** **GPCE '02 Proceedings, LNCS, 2002.**

RS-02-32 **Mads Sig Ager, Olivier Danvy, and Henning Korsholm Rohde.** ***On Obtaining Knuth, Morris, and Pratt's String Matcher by Partial Evaluation.*** **July 2002. 43 pp. To appear in Chin, editor,** ***ACM SIGPLAN ASIAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation,*** **ASIA-PEPM '02 Proceedings, 2002.**

RS-02-31 **Ulrich Kohlenbach and Paulo B. Oliva.** ***Proof Mining: A Systematic Way of Analysing Proofs in Mathematics.*** **June 2002. 47 pp.**

RS-02-30 **Olivier Danvy and Ulrik P. Schultz.** ***Lambda-Lifting in Quadratic Time.*** **June 2002.**

RS-02-29 **Christian N. S. Pedersen and Tejs Scharling.** ***Comparative Methods for Gene Structure Prediction in Homologous Sequences.*** **June 2002. 20 pp.**

RS-02-28 **Ulrich Kohlenbach and Laurenţiu Leuştean.** ***Mann Iterates of Directionally Nonexpansive Mappings in Hyperbolic Spaces.*** **June 2002. 33 pp.**

RS-02-27 **Anna Östlin and Rasmus Pagh.** ***Simulating Uniform Hashing in Constant Time and Optimal Space.*** **2002. 11 pp.**

RS-02-26 **Margarita Korovina.** ***Fixed Points on Abstract Structures without the Equality Test.*** **June 2002.**

RS-02-25 **Hans Hüttel.** ***Deciding Framed Bisimilarity.*** **May 2002. 20 pp.**

RS-02-24 **Aske Simon Christensen, Anders Møller, and Michael I. Schwartzbach.** ***Static Analysis for Dynamic XML.*** **May 2002. 13 pp.**