Untameable Timed Automata!
(Extended Abstract)

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@lsv.ens-cachan.fr

Abstract. Timed automata are a widely studied model for real-time systems. Since 8 years, several tools implement this model and are successfully used to verify real-life examples. In spite of this well-established framework, we prove that the forward analysis algorithm implemented in these tools is not correct! However, we also prove that it is correct for a restricted class of timed automata, which has been sufficient for modeling numerous real-life systems.

1 Introduction

Real-Time Systems - Since their introduction by Alur and Dill in [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 [ACH$^+$92], power of ε-transitions [BDGP98], power of clocks [ACH94,HKWT95], extensions of the model [DZ98,HRS98,CG00,BDFP00,BFH$^+$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 [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, such a construction does not support a natural implementation. Therefore, instead of this construction, algorithms glancing on-the-fly through the automaton are implemented. These last algorithms are based on the notion of

* Partly supported by the French RNRT Project Calife

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

1 http://www-cad.eecs.berkeley.edu:80/~tah/HyTech/

2 http://www-verimag.imag.fr/TEMPORISE/kronos/

3 http://www.uppaal.com/

zones and are efficiently implemented using data structures like DBMs [Dil89] and CDDs [BLP+99]. There are two classes of such algorithms, the class of forward analysis algorithms and the class of backward analysis algorithms. KRONOS implements the two kinds of algorithms, whereas UPPAAL implements only forward analysis algorithms, because it allows the feature of bounded integer variables, for which backward analysis is not well-appropriate.

Our contribution - In this paper, we are interested in the forward analysis algorithms. Our main result is that the forward analysis algorithm implemented in many tools is not correct for the whole class of timed automata. This might appear as very surprising because tools implementing this algorithm are successfully used since 8 years. The problem is due to the fact that we can compare the values of two clocks in the model. We then propose several subclasses of timed automata for which we can safely use a forward analysis algorithm. These subclasses contain in particular the large class of timed automata in which we can not compare two clocks, which might explain why this problem has not been detected before.

Outline of the paper - The structure of the paper is the following: after presenting basic definitions (Section 2), we recall some aspects of the implementation of timed automata and present in particular the algorithm we will study (Section 3). We then discuss the correctness of the algorithm and prove that it is surprisingly not correct (Section 4). However, it is correct for some subclasses of timed automata (Section 5). We conclude with a small discussion (Section 6). For lack of space, technical proofs are not presented in this paper but can be found in [Bou02b].

2 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 $\tau = (t_i)_{1 \le i \le p} \in \mathbb{T}^*$. A *timed word* $\omega = (a_i, t_i)_{1 \le i \le p}$ is an element of $(\Sigma \times \mathbb{T})^*$, also written as a pair $\omega = (\sigma, \tau)$, where $\sigma = (a_i)_{1 \le i \le p}$ is a word in Σ^* and $\tau = (t_i)_{1 \le i \le p}$ a time sequence in $\mathbb{T}^*$ of same length.

Clock Valuations - We consider a finite set X of variables, called *clocks*. A *clock valuation* over X is a mapping $v : X \to \mathbb{T}$ 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 $(\alpha_i)_{1 \le i \le n}$ 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 $x \in X \setminus C$, $([C \leftarrow 0]v)(x) = v(x)$.

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:

$$g \ ::= \ x \sim c \ \mid \ x - y \sim c \ \mid \ g \wedge g \ \mid \ \mathit{true}$$
$$\text{where } x,\ y \in X,\ c \in \mathbb{Z} \text{ and } \sim \in \{<, \le, =, \ge, >\}.$$

We also use the proper subset of *diagonal-free* constraints where the comparison between two clocks is not allowed. This set is defined by the grammar:

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

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

If v is a clock valuation we write $v \models g$ when v satisfies the clock constraint g and we say that v satisfies $x \sim c$ (resp. $x - y \sim c$) whenever $v(x) \sim c$ (resp. $v(x) - v(y) \sim c$).

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[4], $I \subseteq Q$ 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:

$$P = q_0 \xrightarrow{g_1, a_1, C_1} q_1 \ \ldots \ q_{p-1} \xrightarrow{g_p, a_p, C_p} q_p$$

where $(q_{i-1}, g_i, a_i, C_i, q_i) \in T$ 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:

$$(q_0, v_0) \xrightarrow[t_1]{g_1, a_1, C_1} (q_1, v_1) \ \ldots \ \xrightarrow[t_p]{g_p, a_p, C_p} (q_p, v_p)$$

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:

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

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 $L(\mathcal{A})$.

3 Implementation of Timed Automata

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 models is said *decidable* if the emptiness problem is decidable for these models. Note that this problem is equivalent to the *reachability problem* which tests whether a state can be reached in a model.

[4] For more readability, a transition will often be written as $q \xrightarrow{g,a,C} q'$ or even as $q \xrightarrow{g,a,C:=0} q'$ instead of simply the tuple (q, g, a, C, q').

3.1 From Decidability to Implementation

Alur and Dill proved in [AD94] that the emptiness problem is decidable for timed automata. The proof of this result is based on a "region automaton construction". This construction 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 set of clock valuations and which abstracts finitely and correctly (with respect to reachability) the behaviours of timed automata. 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]. There are two families of algorithms, the one performing a forward analysis and the one performing a backward analysis. The tool KRONOS [BTY97,Daw97,Yov98] implements the two kinds of algorithms whereas the tool UPPAAL [BL96,LPY97] implements only forward analysis procedures, because it is more appropriate for dealing also with (bounded) integer variables, a feature proposed by UPPAAL. In this work, we will prove that the forward analysis algorithms implemented in the two previous tools are not correct!

We will now present the basic notions of zones and DBMs, a data structure adapted for representing zones. We will then be able to present the forward analysis algorithm.

3.2 Zones

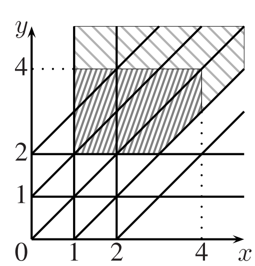
A *zone* is a subset of $\mathbb{T}^n$ defined by a general clock constraint. Let k be a constant. A k*-bounded zone* is a zone defined by a k-bounded clock constraint. Let Z be a zone. The set of k-bounded zones containing Z is finite and not empty, the intersection of these k-bounded zones is a k-bounded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

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

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



3.3 The DBM Data Structure

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 $M = (m_{i,j}, \prec_{i,j})_{i,j=1\ldots n}$ 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, $v(x_0) = 0$):

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

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

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

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 (see [Dil89,CGP99] for a description of this procedure). We will not detail the nice and numerous properties of the DBMs, but we can notice that this data structure allows to compute many operations on zones and also to test for inclusion of zones. More details about this can be found in [Ben02,Bou02b].

3.4 Forward Analysis 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. Forward analysis consists in computing the successors of the initial configuration(s) by iterating the previous Post function. The exact computation does not always terminate, an extrapolation operator on zones is thus used to enforce the termination. In tools, some other abstractions are used [DT98], but they are "orthogonal" to this extrapolation operator in the sense that they are used together with this operator to reduce time and space consumption of the verification process. The algorithm using the extrapolation operator is thus the basis of all the implemented forward analysis algorithms.

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 in $\mathcal{A}$). 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. The basis forward analysis algorithm which is implemented in

Algorithm 1 Forward Analysis Algorithm for Timed Automata

```
# Forward Analysis Algorithm (𝒜: TA ; k: integer) {
#     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, a final state is reachable";}
#             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, no final state is reachable"; }
```

tools is presented as Algorithm 1 (q_0 is the initial location of the automaton whereas Z_0 represents the initial zone, it is most of the time the valuation where all the clocks are set to zero) and is applied taking the maximal constant k as second parameter.

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. Moreover, the DBM data structure can be used to compute all the operations that appear in Algorithm 1, see [Ben02,Bou02b] for details. Let us just point out how it is easy to compute the k-approximation of zones, which may explain why this operator has immediately been adopted in implementations, even before its formalization in [DT98]. Let k be an integer and Z a zone represented by a DBM in normal form $M = (m_{i,j}; \prec_{i,j})_{i,j=0\dots n}$. We define the DBM $M' = (m'_{i,j}; \prec'_{i,j})_{i,j=0\dots n}$ by

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

The DBM M' may not be in normal form, but $[\![M']\!] = \mathit{Approx}_k(Z)$.

Algorithm 1 computes step-by-step an overapproximation of the set of reachable states and tests whether this approximation intersects the set of final states, or not. 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 algorithm does a mistake. The algorithm will be said *correct* (with respect to reachability) whenever it never does such a mistake. We will now discuss in details the correctness of Algorithm 1.

4 A Correctness Problem

Algorithm 1 is implemented in tools like UPPAAL and KRONOS. The correctness of this algorithm is asserted in many papers in the literature and several attempts of proofs can be found [WT94,DT98,Daw98,Tri98,Pet99,Bou02a]. All these attempts of proofs are however incomplete or buggy (*cf* [Bou02b] for more details), and these proofs can not be corrected, because we will prove that Algorithm 1 is indeed not correct!

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 is not correct!** whatever is the choice of the parameter k. Consider the automaton $\mathcal{C}$ depicted on Fig. 1.

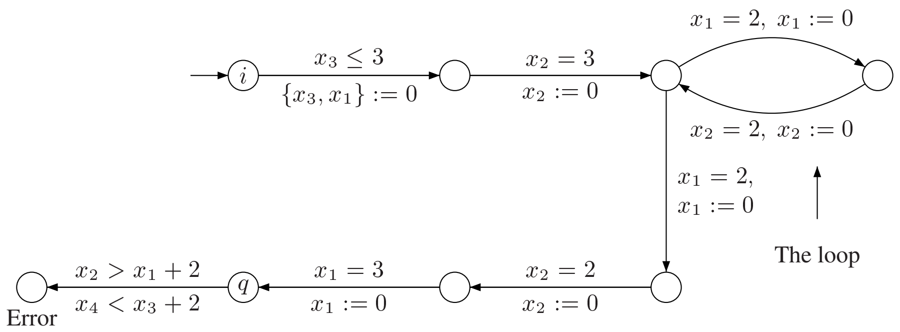


Fig. 1. A surprising timed automaton, $\mathcal{C}$

Consider a path from i to q in the automaton $\mathcal{C}$. If d is the date the first transition is taken and if α is the number of loops taken along the run, the valuation v of the clocks when arriving in q is defined by:

$$\begin{array}{ll} v(x_1) = 0 & v(x_3) = 2\alpha + 5 \\ v(x_2) = d & v(x_4) = d + 2\alpha + 5 \end{array}$$

Thus, applying an exact computation of the successors of the initial state i, with all the clocks set to 0, the set of valuations that can be reached in state q, when the loop is taken α times, is defined by the relations (1) in **Table 1**. This set of valuations is of course a zone and is denoted by Z_α. It can be depicted by the scheme on Fig. 2.

$$\begin{cases} x_2 \geq 1 \\ x_3 \geq 2\alpha + 5 \\ x_4 \geq 2\alpha + 6 \\ 1 \leq x_2 - x_1 \leq 3 \\ 1 \leq x_4 - x_3 \leq 3 \\ x_3 - x_1 = 2\alpha + 5 \\ x_4 - x_2 = 2\alpha + 5 \end{cases} \quad (1) \qquad \Bigg| \qquad \begin{cases} x_2 \geq 1 \\ x_3 > k \\ x_4 > k \\ 1 \leq x_2 - x_1 \leq 3 \\ 1 \leq x_4 - x_3 \leq 3 \\ x_3 - x_1 > k \\ x_4 - x_2 > k \end{cases} \quad (2)$$

Table 1. Equations of the zones Z_α (on the left) and $\mathit{Approx}_k(Z_\alpha)$ (on the right)

Even if it is not explicit in the description of Z_α through the equations (1), we can easily deduce, and it appears clearly on the representation of the zone on Fig. 2, that if v is a valuation of Z_α, we have the very strong constraint that

$$v(x_4) - v(x_3) = v(x_2) - v(x_1) \tag{3}$$

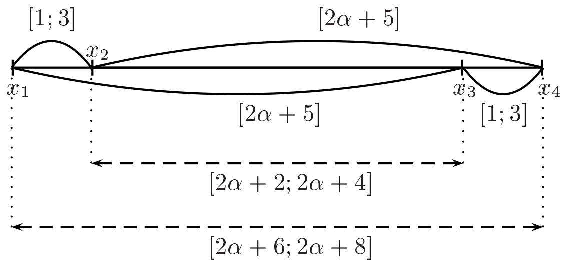


Fig. 2. The zone Z_α

In particular, we get that the state "Error" of $\mathcal{C}$ is not reachable.

However, the condition (3) is not explicit in the definition of Z_α, and this condition will be forgotten when we will apply an extrapolation operator to the zone. Let k be a constant and let α be such that $2\alpha + 2 > k$. Applying the extrapolation operator "Approx_k" to Z_α, the zone $\mathit{Approx}_k(Z_\alpha)$ we obtain is defined by the relations (2) in Table 1.

We thus get that Algorithm 1 will compute, whatever is the choice of the parameter k, that the state "Error" is reachable, which is, as said before, wrong. Thus, we conclude that for the automaton $\mathcal{C}$, no extrapolation operator is correct with respect to reachability: there is no way to choose correctly a constant k such that Algorithm 1 applied to $\mathcal{C}$ with the parameter k gives a correct answer.

Remark 1. Note that, for simplicity, we only applied the extrapolation operator to the zone computed for the state q. However, as it is an increasing operator, if we use it at each state along the computation, the zone that will be computed when arriving in state q will contain Z_α (if the loop is taken α times), and thus, the state "Error" will of course also be computed as reachable.

From this counter-example, we get that it is impossible to propose "good" constants which will allow to apply safely Algorithm 1. This is really a big surprise! To sum up, we get the following result:

Theorem 1. *Algorithm 1 is not correct with respect to reachability, whatever is the choice of the parameter* k.

Moreover, assume we want to modify slightly Algorithm 1 and we want to find a finite abstraction operator *abs*[5] such that, if Z is a zone, $\mathit{abs}(Z)$ is a zone containing Z. Then, a simple corollary of Theorem 1 is that Algorithm 1 modified in such a way that the extrapolation operator Approx_k is replaced by the abstraction operator *abs*, is also not correct with respect to reachability (taking k as the biggest constant involved in the definition of a zone from the set $\{\mathit{abs}(Z) \mid Z \text{ is a zone}\}$, we get that for every zone Z, $\mathit{Approx}_k(Z) \subseteq \mathit{abs}(Z)$).

[5] An abstraction operator is said *finite* whenever $\{\mathit{abs}(Z) \mid Z \text{ is a zone}\}$ is a finite set.

The aim of the remainder of this paper is to study more precisely where are the problems and to bring out subclasses for which we can find a correct extrapolation operator. This will partly explain why the bug has not been found before.

5 Out of the Surface

What precedes might appear as a very bad news for the verification of timed systems, because one of the bases of the model-checking of timed automata collapses. However, it is now 8 years that this algorithm is implemented in tools like KRONOS and UPPAAL and used for checking the correctness of many timed systems and this "bug" has not been found yet. If we refine the previous result and if we focus on the counter-example we constructed, we get that Algorithm 1 is not correct for timed automata that use diagonal constraints (the last transition of $\mathcal{C}$ is labeled by the constraint $x_2 > x_1 + 2 \wedge x_4 < x_3 + 2$) and that use more than four clocks. For lack of space, the following results can not be detailed in this paper, see [Bou02b] for more details.

Diagonal-Free Timed Automata - We restrict to diagonal-free timed automata, *i.e.* timed automata which use only clock constraints of the form $x \sim c$. The very nice following theorem then holds.

Theorem 2. *Algorithm 1 is correct for diagonal-free timed automata (where the constant used for the extrapolation operator is the maximum constant used in one of the clock constraints of the automaton).*

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]). This might explain why the bug has not been found before. Moreover, from a theoretical point of view, every timed automaton can be transformed into a diagonal-free timed automaton (see [BDGP98] for a proof), but the transformation suffers from an exponential blow-up of the size of the automaton.

General Timed Automata - We have seen that it is hopeless to find a very wide class of general timed automata for which Algorithm 1 is correct. However, the following result completes the puzzle.

Proposition 3 *Algorithm 1 is correct for timed automata which use no more than three clocks (the constant used as a parameter for the extrapolation operator is for example the product $n.k$ where k is the maximal constant appearing in the automaton and n is the number of clocks which is used* [6]*).*

The reason why this theorem holds for three clocks, but not for four clocks, apart from the counter-example of Fig. 1, is that resetting clocks is similar to being in a two-dimensional time-space, and the two-dimensional time-space has very nice properties that one can **visualize** but that can not be extended to other dimensions.

[6] However, this constant can be tightened, see for example [Bou02b].

We have thus characterized in a very thin way the classes of automata for which we can safely use Algorithm 1: if the timed automaton is diagonal-free or if it has no more than three clocks, then we can use Algorithm 1, but if the timed automaton has both diagonal clock constraints and more than four clocks, then we can not use safely Algorithm 1, it may do a mistake.

6 Conclusion and Discussion

In this paper, we studied the forward analysis algorithm which is implemented in several tools. This algorithm belongs to the "basic knowledge" of all the people working on the verification of timed systems. However, in spite of the success stories of several implementations of this algorithm and their applications to real case studies, we did prove that it is not correct!

This algorithm is based on an abstraction operator, which is a very natural operator on DBMs, the data structure used to represent zones. However, when we apply this operator, we can lose some important relations in zones, like, for example, the equality of two differences of clocks. An alternative to this abstraction operator has to be proposed, in order to be able to verify safely timed automata that also use diagonal constraints on clocks. However, as said page 8, a corollary of our non-correctness result is that no finite abstraction on zones (which transforms a zone into a larger zone) is correct for a forward analysis. Some alternative propositions are done in [Bou02b], but it is not really satisfactory because the solutions suffer from a big combinatorics explosion. Zones are maybe not as appropriate as what one could think for analyzing timed automata...

Acknowledgments: I would like to thank Antoine Petit for his careful reading of some drafts of this paper and François Laroussinie for some interesting discussions on DBMs. I would also like to thank Kim G. Larsen, Emmanuel Fleury and Gerd Behrmann for our discussions on the implementation of timed automata during our so-called UPPAAL meetings.

References

[ACD+92] Rajeev Alur, Costas Courcoubetis, David Dill, Nicolas Halbwachs and Howard Wong-Toi. *An Implementation of Three Algorithms for Timing Verification Based on Automata Emptiness*. In *Proc. 13th IEEE Real-Time Systems Symposium (RTSS'92)*, pp. 157–166. IEEE Computer Society Press, 1992.

[ACH+92] Rajeev Alur, Costas Courcoubetis, Nicolas Halbwachs, David Dill and Howard Wong-Toi. *Minimization of Timed Transition Systems*. In *Proc. 3rd International Conference on Concurrency Theory (CONCUR'92)*, vol. 630 of *Lecture Notes in Computer Science*, pp. 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)*, vol. 836 of *Lecture Notes in Computer Science*, pp. 162–177. Springer, 1994.

[AD94] Rajeev Alur and David Dill. *A Theory of Timed Automata*. Theoretical Computer Science (TCS), vol. 126(2):pp. 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)*, vol. 818 of *Lecture Notes in Computer Science*, pp. 1–13. Springer, 1994.

[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.

[BDFP00] Patricia Bouyer, Catherine Dufourd, Emmanuel Fleury and Antoine Petit. *Are Timed Automata Updatable?*. In *Proc. 12th International Conference on Computer Aided Verification (CAV'2000)*, vol. 1855 of *Lecture Notes in Computer Science*, pp. 464–479. 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, vol. 36(2–3):pp. 145–182, 1998.

[Ben02] Johan Bengtsson. *Clocks, DBMs ans States in Timed Systems*. Ph.D. thesis, Department of Information Technology, Uppsala University, Uppsala, Sweden, 2002.

[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)*, vol. 2034 of *Lecture Notes in Computer Science*, pp. 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)*, vol. 1633 of *Lecture Notes in Computer Science*, pp. 341–353. Springer, 1999.

[Bou02a] Patricia Bouyer. *Modèles et algorithmes pour la vérification des systèmes temporisés*. Ph.D. thesis, École Normale Supérieure de Cachan, Cachan, France, 2002.

[Bou02b] Patricia Bouyer. *Timed Automata May Cause Some Troubles*. *Research Report LSV–02–9*, Laboratoire Spécification et Vérification, ENS de Cachan, France, 2002. Also Available as *BRICS Research Report RS-02-35*, Aalborg University, Denmark, 2002.

[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)*, pp. 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), vol. 5(4):pp. 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.

[Daw98] Conrado Daws. *Méthodes d'analyse de systèmes temporisés : de la théorie à la pratique*. Ph.D. thesis, Institut National Polytechnique de Grenoble, Grenoble, France, 1998.

[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*, vol. 407 of *Lecture Notes in Computer Science*, pp. 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)*, vol. 1066 of *Lecture Notes in Computer Science*, pp. 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)*, vol. 1384 of *Lecture Notes in Computer Science*, pp. 313–329. Springer, 1998.

[DZ98] François Demichelis and Wieslaw Zielonka. *Controlled Timed Automata*. In *Proc. 9th International Conference on Concurrency Theory (CONCUR'98)*, vol. 1466 of *Lecture Notes in Computer Science*, pp. 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), vol. 1(1–2):pp. 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)*, vol. 944 of *Lecture Notes in Computer Science*, pp. 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)*, vol. 1443 of *Lecture Notes in Computer Science*, pp. 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)*, pp. 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), vol. 1(1–2):pp. 134–152, 1997.

[Pet99] Paul Pettersson. *Modelling and Verification of Real-Time Systems Using Timed Automata: Theory and Practice*. Ph.D. thesis, Department of Computer Systems, Uppsala University, Uppsala, Sweden, 1999. Available as DoCS Technical Report 99/101.

[Tri98] Stavros Tripakis. *L'analyse formelle des systèmes temporisés en pratique*. Ph.D. 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)*, pp. 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, vol. 18(1):pp. 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)*, vol. 863 of *Lecture Notes in Computer Science*, pp. 694–715. Springer, 1994.

[WT94] Howard Wong-Toi. *Symbolic Approximations for Verifying Real-Time Systems*. Ph.D. 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, vol. 11(2):pp. 113–136, 1997.

[Yov98] Sergio Yovine. *Model-Checking Timed Automata*. In *School on Embedded Systems*, vol. 1494 of *Lecture Notes in Computer Science*, pp. 114–152. Springer, 1998.