Diagonal Constraints in Timed Automata

Forward Analysis of Timed Systems

Patricia Bouyer, François Laroussinie, Pierre-Alain Reynier

LSV – CNRS & ENS Cachan – France
{bouyer,fl,reynier}@lsv.ens-cachan.fr

Abstract. Timed automata (TA) are a widely used model for real-time systems. Several tools are dedicated to this model, and they mostly implement a forward analysis for checking reachability properties. Though diagonal constraints do not add expressive power to classical TA, the standard forward analysis algorithm is not correct for this model. In this paper we survey several approaches to handle diagonal constraints and propose a refinement-based method for patching the usual algorithm: erroneous traces found by the classical algorithm are analyzed, and used for refining the model.

1 Introduction

Model checking. The development of reactive, critical or embedded systems requires the use of formal verification methods. Model checking consists in verifying automatically that a model fulfills its specification and has been widely and successfully applied to industrial systems. It is often necessary to consider quantitative informations on time elapsing in both the model description and the property to be verified. Timed automata (TA) have been proposed by Alur and Dill [AD94] to model such real-time systems. Since then, many theoretical results have been obtained: decidability of reachability properties [AD94], model checking for timed temporal logics [ACD93,HNSY94], *etc*.

Reachability in timed automata. Decidability of reachability properties in timed automata is based on the construction of the so-called region automaton, which finitely abstracts behaviours of timed automata [AD94]. However in practice this construction is not implemented, and symbolic on-the-fly algorithms have been proposed to overcome the complexity blow-up induced by timing constraints. These procedures are often based on *zones* and *DBMs* to represent the sets of clock valuations. In particular, an on-the-fly forward reachability algorithm using zones has been developed and implemented in tools like UPPAAL [LPY97] or KRONOS [DOTY96]. Even if timed automata form a decidable class, the exact forward computation may not terminate. To overcome this problem, an *abstraction* operator over zones needs to be used [DT98].

Guards in timed automata. Classical timed automata [AD94] consider only simple constraints $x \sim c$ and *diagonal constraints* $x - y \sim c$. Surprisingly the standard forward reachability algorithm based on zones has been recently shown to be correct only for TA with simple constraints, but not always correct for TA using diagonal constraints [Bou03,BY03]: locations of TA with diagonal constraints may be found reachable by

the algorithm while they are not! This problem comes from the use of the abstraction operator over zones.
From [AD94,BDGP98] we know that diagonal constraints can be removed from TA. This gives a procedure for verifying TA with diagonal constraints: construct a TA without diagonal constraints and then apply the standard forward analysis algorithm. However, removing diagonal constraints induces a blowup in the size of the model (it is exponential in the number of diagonal constraints). This is clearly too expensive to be used on real-life systems. Moreover diagonal constraints do not always raise wrong diagnosis (only few examples can be found in the literature) and a systematic removal of all diagonal constraints may therefore not be pertinent.

Our contribution. In this paper we propose a refinement-based method which does not systematically remove all diagonal constraints. The core of our method is an algorithm which analyzes an erroneous trace provided by the classical algorithm and selects a set G of diagonal constraints which causes the error. We then remove diagonal constraints of G from the model and re-run the classical algorithm on the refined model.

Outline of the paper. In Section 2 we introduce basic notions of timed automata, forward reachability analysis, zones and abstractions. In Section 3 we survey several approaches to handle diagonal constraints and propose a refinement-based method for patching the usual algorithm. In section 4 we describe our algorithm for selecting pertinent diagonal constraints. For this, we introduce an extension of the DBM data structure which allows us to store information on the dependence of computed zones w.r.t. diagonal constraints. We prove correctness and progress of our refinement-based method.

2 Forward Analysis of Timed Automata

Basic definitions, timed automata. We consider as time domain $\mathbb{T}$ the set $\mathbb{Q}^+$ of non-negative rationals or the set $\mathbb{R}^+$ of non-negative reals. 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 $(v + t)(x) = v(x) + t, \forall x \in X$. For a subset r of X, we denote by $[r \leftarrow 0]v$ the valuation such that for each $x \in r$, $([r \leftarrow 0]v)(x) = 0$ and for each $x \in X \setminus r$, $([r \leftarrow 0]v)(x) = v(x)$.

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 grammar "$g ::= x \sim c \mid x - y \sim c \mid g \wedge g \mid \mathtt{tt}$" where $x,\ y \in X$, $c \in \mathbb{Z}$, $\sim \in \{\leq, =, \geq\}$ and $\mathtt{tt}$ stands for true. We also use the proper subset $\mathcal{C}_{df}(X)$ of *diagonal-free* constraints where the constraints of the form $x - y \sim c$ (called *diagonal constraints*) are not allowed. To simplify, we do not consider strict inequalities, but everything presented in this paper extends easily to strict inequalities. We write $v \models g$ when the clock valuation v satisfies the clock constraint g. A clock constraint is said k*-bounded* whenever it only uses constraints with constants between $-k$ and $+k$.

A *timed automaton* (TA for short) over $\mathbb{T}$ is a tuple $\mathcal{A} = (\Sigma, X, L, \ell_0, T)$, where Σ is a finite alphabet of actions, X is a finite set of clocks, L is a finite set of locations, $\ell_0 \in L$ is the initial location, and $T \subseteq L \times [\mathcal{C}(X) \times \Sigma \times 2^X] \times L$ is a finite set of edges (or transitions). If only diagonal-free constraints are used on transitions, the timed automaton is said to be *diagonal-free*. A *state* of $\mathcal{A}$ is a pair $\langle \ell, v\rangle$ where $\ell \in L$ is the current location and $v \in \mathbb{T}^X$ represents the current values of clocks. The initial state is $\langle \ell_0, v_0\rangle$ where v_0 is the valuation mapping all clocks in X to 0. The semantics of $\mathcal{A}$ can be described as an infinite transition system whose states are states of $\mathcal{A}$ and whose transitions correspond to time elapsing followed by an enabled edge in $\mathcal{A}$. More precisely, from a state $\langle \ell, v\rangle$, it is possible to reach a state $\langle \ell', v'\rangle$ if there exist $\delta \in \mathbb{T}$ and $(\ell, g, a, r, \ell') \in T$ such that $v+\delta \models g$ and $v' = [r \leftarrow 0](v+\delta)$. Now we can define a *run* of $\mathcal{A}$ as a finite sequence of such steps, it is denoted:

$$\langle \ell_0, v_0\rangle \xrightarrow[t_1]{g_1,a_1,r_1} \langle \ell_1, v_1\rangle \xrightarrow[t_2]{g_2,a_2,r_2} \ \ldots \ \xrightarrow[t_p]{g_p,a_p,r_p} \langle \ell_p, v_p\rangle$$

where t_i is the amount of time elapsed since state $\langle \ell_0, v_0\rangle$—the duration of time elapsing in the i-th location is then $\delta_i = t_{i+1} - t_i$. In the following we abstract away names of actions because we will only consider reachability properties.

Reachability in timed automata. Reachability is a fundamental problem in verification. For timed automata, it is stated as follows: given a timed automaton $\mathcal{A}$ and a set of locations L_f, does there exist a run leading to some state $\langle \ell, v\rangle$, with $l \in L_f$? This problem has been proved decidable (and PSPACE-complete) by Alur and Dill [AD94]. The proof is based on the well-known *region* construction: the (infinite) set of states of $\mathcal{A}$ is partitioned into a finite set of regions such that two states which belong to the same region satisfy the same reachability properties.

Algorithms for reachability. In practice the region construction is not used to check reachability properties because the number of regions is too high: it is not abstracted enough to be applied successfully over non-trivial systems. For this purpose, symbolic and on-the-fly algorithms have been proposed and implemented [LPY97]. They use the constraints of $\mathcal{C}(X)$ as symbolic representations for the sets of valuations. In this framework such a constraint is called a *zone* and is usually implemented with DBMs (Difference Bound Matrices [BM83,Dil90]). Backward analysis raises no real problem but forward analysis is more convenient for verifying timed automata with useful features like integer variables. Given a zone Z and an edge $e = (\ell, g, a, r, \ell')$, $\mathsf{Post}(Z, e)$ denotes the zone corresponding to the set $\{[r \leftarrow 0](v+t) \in \mathbb{T}^X \mid v \in Z,\ t \geq 0, \text{and } v+t \models g\}$. Symbolic transitions can then be defined over *symbolic states* (ℓ, Z) using the $\mathsf{Post}(\cdot)$ operator. The symbolic graph may however be infinite, because constants used in zones may grow for ever. The forward computation does not terminate in general. To avoid this phenomenon, an abstraction operator, called the *k-extrapolation* (k is a constant supposed to be greater than the maximal constant occurring in $\mathcal{A}$), is used at each iteration: $\mathsf{Extra}_k(Z)$ denotes the smallest zone containing Z and defined by a k-bounded clock constraint. Together with inclusion checking (line 9. of the algorithm), this clearly entails the termination of the classical procedure described as Algorithm FRA (see Algorithm 1). If a location of L_f is found as reachable, Algorithm FRA returns a witness trace (*i.e.* a sequence of consecutive edges).

Algorithm 1 Forward Reachability Analysis – FRA

Algorithm FRA ($\mathcal{A}$: timed automaton; L_f: set of final locations) $\{$
    Define k as the maximal constant appearing in $\mathcal{A}$;
    Visited := $\emptyset$; ($\star$ Visited stores the visited states $\star$)
    Waiting := $\{(\ell_0, \mathsf{Extra}_k(Z_0))\}$; ($\star$ $Z_0 = \{v_0\}$ $\star$)
    Repeat
        Get and Remove (ℓ, Z) from Waiting;
        If $\ell \in L_f$ ($\star$ ℓ is a final location $\star$)
            then $\{$Return "Yes" and a witness trace;$\}$
            else $\{$If there is no $(\ell, Z') \in$ Visited s.t. $Z \subseteq Z'$ ($\star$ inclusion checking $\star$)
                then $\{$Visited := Visited $\cup \{(\ell, Z)\}$;
                    Succ := $\{(\ell', \mathsf{Extra}_k(\mathsf{Post}(Z, e))) \mid e$ edge from ℓ to $\ell'\}$;
                    Waiting := Waiting $\cup$ Succ;$\}\}$
    Until (Waiting = $\emptyset$);
    Return "No"; $\}$

Completeness and correctness. Obviously, as the k-extrapolation of zones is an over-approximation, this algorithm is complete: any reachable location is found as reachable by the algorithm. The correctness ("only reachable locations are found as reachable by the algorithm") is more difficult to state. In [Bou03,BY03], Algorithm FRA has been proved correct for diagonal-free timed automata and it has been shown to be **not correct** for timed automata using also diagonal constraints. Figure 1 illustrates this correctness problem: Algorithm FRA sees location "Error" of $\mathcal{A}$ as reachable whereas it is not (see Appendix A for details). This problem is not due to the choice of constant k or to the definition of $\mathsf{Extra}_k(\cdot)$: if we replace the operator $\mathsf{Extra}_k(\cdot)$ by any operator $\mathsf{Extra}_K(\cdot)$ for some K or even by any abstraction operator Abs such that for every zone Z, $\mathsf{Abs}(Z)$ is a zone containing Z, and $\{\mathsf{Abs}(Z) \mid Z \text{ zone}\}$ is finite (to ensure termination of the forward analysis algorithm), then the algorithm will not be correct and will announce state "Error" as reachable in $\mathcal{A}$ (see [Bou04]).

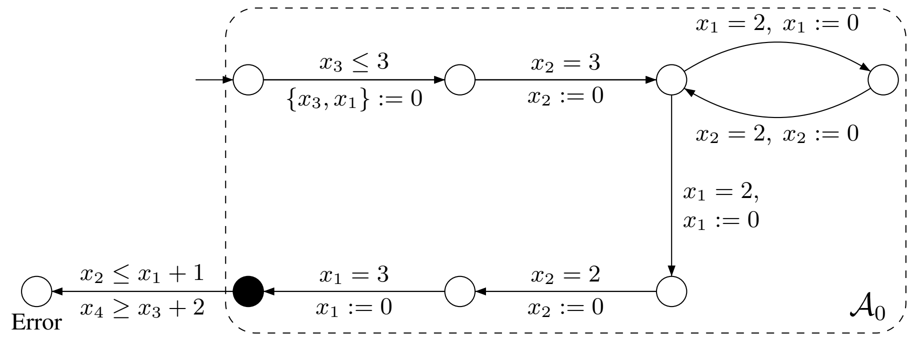


Fig. 1. Automaton $\mathcal{A}$

Extrapolation and zones. As explained above, zones are a suitable symbolic representation for clock valuations but contrary to what is sometimes assumed, zones are not a symbolic way of handling regions: there is indeed no simple correspondence between zones and (sets of) regions, which explains the correctness problem encountered in Algorithm FRA. For example, in the diagonal-free case, there exist k-bounded zones which are strictly included in a region. Concerning models with diagonal constraints, one can compute (using Algorithm FRA) a zone Z such that there is a region R with $Z \cap R = \emptyset$ while $\mathsf{Extra}_k(Z) \cap R \neq \emptyset$. Such a phenomenon appears for example in the automaton of Figure 1. This emphasizes the fact that we have always to be careful when handling regions and zones, and to separately consider methods based on regions and methods based on zones. This seems related to abstraction problems encountered in the verification of infinite-state systems, and we will use classical refinement techniques [CGJ$^+$00] to solve our problem.

3 Methods for Handling Diagonal Constraints

Our aim is to propose an efficient forward algorithm based on zones for checking reachability properties in timed automata with diagonal constraints. We can distinguish two main approaches. First there are the ones based on a systematic removal of diagonal constraints: the original TA is replaced by a diagonal-free TA and a standard algorithm is then applied. Secondly there are methods in which diagonal constraints are treated only when they induce spurious traces: if the constraints generate no problem, then these methods provide no extra-cost compared with the standard reachability algorithm. This last criterion is very important because there are only few problematic cases.

3.1 Systematic Removal of Diagonal Constraints

It is well known that given a TA $\mathcal{A}$ with diagonal constraints, it is possible to build a diagonal-free TA $\mathcal{A}'$ s.t. $\mathcal{A}$ and $\mathcal{A}'$ verify the same reachability properties [BDGP98]. This construction combined with the classical reachability algorithm for diagonal-free TA provides a correct forward algorithm for TA, and such a method avoids the expensive region automaton construction. Nevertheless removing diagonal constraints entails a complexity blow-up: if n is the number of diagonal constraints in $\mathcal{A}$, the size of $\mathcal{A}'$ is in $\mathcal{O}(|\mathcal{A}| \cdot 2^n)$. This approach is clearly too expensive, especially if we assume that diagonal constraints mostly raise no error.

Intuitively, if we want to avoid problems with diagonal constraints, it seems sufficient to ensure the following property: $\forall Z$ computed, $\forall g$ diagonal, $Z \subseteq g \vee Z \subseteq \neg g$ $(*)$. The method proposed by Bengtsson and Yi in [BY04] relies on this criterion: after each application of the extrapolation operator, the zone which is obtained is split so that Property $(*)$ holds. Like the construction of [BDGP98], this solution suffers from an exponential blow-up of the number of zones visited during the computation. Indeed, the complexity of Algorithm FRA crucially depends on the number of zones which need to be handled, and with both two previous methods, the number of zones is multiplied by 2^n, where n is the number of diagonal constraints of the initial automaton.

Finally we could also restrict the removal of the two previous methods to the *active* diagonal constraints in the current control location, following the idea proposed for clocks by Daws and Yovine in [DY96], and generalized in [BBFL03].
Note that all these methods induce a complexity blow-up even if there is no false-positive execution. Indeed, timed automata with diagonal constraints are exponentially more succinct than diagonal-free timed automata [BC05].

3.2 Target Methods for Spurious Traces

In this case, the aim is to develop special heuristics when a false-positive is found. This would permit to have algorithms as efficient as the standard one when there is no problem with diagonal constraints.
First note that given a symbolic execution, it is easy to check whether it is consistent (*i.e.* whether a corresponding run actually exists in $\mathcal{A}$) or if it is a *false positive*. This can be done by using a forward computation with no extrapolation (the finiteness of the execution ensures termination).
Therefore a natural (but wrong!) method could be: (1) use the standard reachability algorithm, (2) if a location is found reachable through a symbolic execution ρ, check whether ρ is a false positive, (3) if ρ is a false positive, run further the algorithm. But this procedure is not complete: some reachable locations may be missed by this algorithm. For example, assume that a false positive contains a symbolic state (ℓ, Z), and assume that later in the algorithm, a symbolic state (ℓ, Z') is computed with $Z' \subseteq Z$. Because of the inclusion checking, Algorithm FRA will stop the computation, whereas it is possible that a valid run goes through symbolic state (ℓ, Z'): inclusion between extrapolated zones does not imply inclusion between exact zones. Such a situation is illustrated in Appendix B. On the other hand, removing states of the false positive trace from the list of visited states could prevent termination of the computation.
We now consider two methods that extend this idea in order to have a complete and correct algorithm.

Combining forward and backward computation. Since diagonal constraints are correctly handled with the backward computation, a possible approach consists in combining forward and backward computations. This algorithm works in two steps. First one performs a forward analysis: If a location is found reachable with a correct execution, the algorithm stops; If a false-positive execution is found, it is stored in the visited states list and the algorithm continues. At the end of the first step, either a correct execution has been found (and the answer is YES), or no spurious execution has been found (and the answer is NO), otherwise the second step begins. It consists in a backward computation from the target states of the spurious executions. This backward computation is restricted to the set of visited states computed in the first step. Such a method would work, but it has an important drawback: the backward computation does not handle additional data: in UPPAAL for ex., it is possible to add integer variables and operations over these data, they can be treated in forward computations but not in backward. This restricts a lot the applicability of the method.

A refinement-based and pure forward method. We now propose to use a refinement-based method (illustrated on Figure 2): (a) use the standard algorithm over a model M, (b) when a false positive ρ is found, refine the model in such a way that ρ will be correctly treated in the refined model M', and (c) restart the procedure over M'. Such a methodology has been proposed in [CGJ$^+$00] and has been applied to many kinds of infinite-state systems (constraint-based programs [HJMS02], hybrid systems [ADI03], *etc*). In our case, a refinement step will consist in removing some diagonal constraints of the initial automaton. Termination is clearly ensured if at least one diagonal constraint is selected at each iteration. As removing diagonal constraints is expensive, the key idea is to refine w.r.t. diagonal constraints only if it is necessary: when a false positive ρ is found, we want to find as few diagonal constraints as possible such that if we remove these diagonal constraints in the model, then the same false positive will not be found again by Algorithm FRA. Selecting pertinent diagonal constraints is the core of our algorithm and will be presented in the next section. We now briefly present the refinement step of our algorithm.

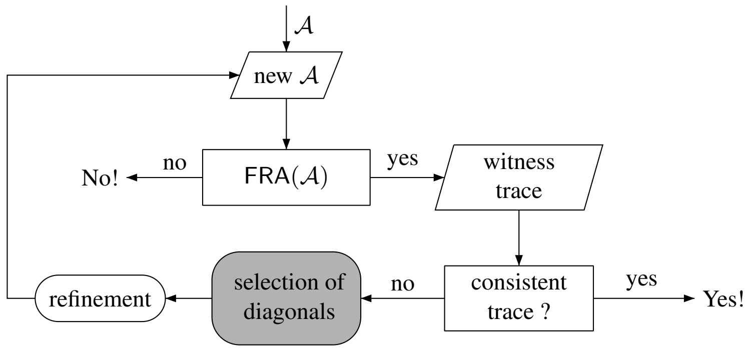


Fig. 2. Refinement-based method

Refinement w.r.t. a constraint g. Given a TA $\mathcal{A} = (\Sigma, X, L, \ell_0, T)$ and a diagonal constraint $g = (x - y \sim c)$, we consider the method proposed in [BDGP98] to remove g from $\mathcal{A}$: the truth value of g is encoded into locations of the refined automaton $\mathcal{A}_g$. This boolean value is not changed by time elapsing, it can only be modified by a reset of x or y. Locations of $\mathcal{A}_g$ are pairs (ℓ, ϵ) with $\ell \in L$ and $\epsilon \in \{\top, \bot\}$, and edges are directly derived from those of $\mathcal{A}$: they either relate locations with the same or opposite truth value depending on the reset of the corresponding edge in $\mathcal{A}$, and the occurrences of the guard $x - y \sim c$ are just replaced by their truth value in the current location. This construction can be directly extended to a set of diagonal constraints. If G is a finite set of diagonal constraints and $\mathcal{A}$ a TA, we note $\mathsf{Split}(\mathcal{A}, G)$ the TA which is obtained after refinement w.r.t. the constraints in G.

4 Diagonal Constraint Analysis

We assume a timed automaton $\mathcal{A} = (\Sigma, X, L, \ell_0, T)$ is given, and we set n the cardinal of set X. Let k be an integer greater than the maximal constant occurring in $\mathcal{A}$. Let $\mathsf{Diag}(\mathcal{A})$ be the set of diagonal constraints occurring in the guards of $\mathcal{A}$. In this section, we propose an algorithm which, given an erroneous trace ρ, selects a set of diagonal guards $G \subseteq \mathsf{Diag}(\mathcal{A})$ which satisfies the two following conditions: (i) $G \neq \emptyset$, and (ii) in $\mathsf{Split}(\mathcal{A}, G)$, the erroneous trace ρ doesn't exist anymore. Condition (i) ensures termination of the refinement method (as $\mathsf{Diag}(\mathcal{A})$ is a finite set), and condition (ii) is a *progress* condition: a given erroneous path will never be found twice.
Of course, the set of all diagonal guards appearing along the erroneous trace satisfies both conditions. But our aim is to select as few guards as possible. Our algorithm builds a possibly much smaller set of guards, and it does not increase the complexity: it is linear in the length of the run, like the consistency checking.

4.1 Erroneous Traces

Algorithm FRA returns a witness trace whenever a final state is found as reachable (a *trace* ρ is a sequence of consecutive edges of $\mathcal{A}$). Such a trace is *erroneous* whenever no real run follows the same edges as the trace. To formalize this notion, we associate with a trace ρ two zones: the zone Z^e_ρ which corresponds to all valuations which are reachable following trace ρ, and Z^a_ρ which corresponds to all valuations which are found as reachable when applying the abstract symbolic computation [1] along ρ. We can define these two families of zones inductively as follows. For the empty trace (denoted ϵ), $Z^e_\epsilon = Z^a_\epsilon = \{v_0\}$, and if ρ is a trace and α an edge such that $\rho.\alpha$ is also a trace, we have $Z^e_{\rho.\alpha} = \mathsf{Post}(Z^e_\rho, \alpha)$, and $Z^a_{\rho.\alpha} = \mathsf{Extra}_k(\mathsf{Post}(Z^a_\rho, \alpha))$. Note that the zones defined here only depend on ρ and not on automaton $\mathcal{A}$. A trace ρ is said *erroneous* (we also say that it is a *false positive*) for $\mathcal{A}$ when $Z^e_\rho = \emptyset$ whereas $Z^a_\rho \neq \emptyset$.

Algorithm FRA is correct for diagonal-free timed automata. Thus, if ρ is an erroneous trace, there must exist some diagonal guard along ρ which is the cause of the error. One hope could be that when the exact and the abstract computations disagree, the last guard encountered is diagonal (as it is the case for the automaton in Figure 1), and that it is sufficient to refine $\mathcal{A}$ w.r.t. this guard to get rid of the erroneous trace. This is however not the case: this guard can be a simple non diagonal constraint. Automaton in Figure 3 illustrates this point ($\mathcal{A}_0$ refers to the automaton depicted on Figure 1): the last transition, whose guard is non-diagonal, leads to an empty exact computation while the abstract one is not empty. Understanding the role of diagonal constraints along an erroneous trace thus requires a precise analysis of the whole execution.

4.2 Propagation of Constraints, the EDBM Data Structure

For analyzing an erroneous trace ρ which is output by Algorithm FRA, we will use a forward computation. We want to understand precisely the effect of each diagonal

[1] e stands for exact and a for abstract.

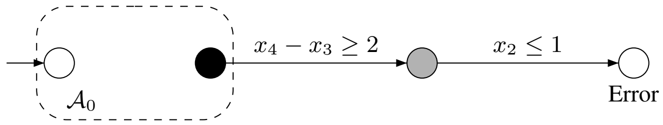


Fig. 3. The problem occurs on the simple constraint $x_2 \leq 1$.

constraint and therefore we will study the differences between the zones Z^e_{-} and Z^a_{-}. We won't directly reason on these zones but on their representations with DBMs. A DBM $M = (m_{i,j})_{0 \leq i,j \leq n}$ is a square matrix of size $n+1$ (where n is the number of clocks) whose entries belong to $\mathbb{Z} \cup \{\infty\}$. Its semantics is the zone $[\![M]\!] = \{v \in \mathbb{T}^X \mid \forall i,j \quad x_i - x_j \leq m_{i,j}\}$ (we use a clock x_0 assumed to be equal to 0 in order to represent constraints $x_i \sim c$ as a difference constraint $x_i - x_0 \sim c$). To have a non-ambiguous representation of zones by DBMs, we need to compute *normal forms*: it consists in applying Floyd's Algorithm for shortest paths over the implicit weighted oriented graph described by such a matrix (see [Dil90,Bou04] for classical results over DBMs).

As we use a shortest paths algorithm, the values of the entries depend not only on the last guard we intersect but also on other entries. Just as it is done in Floyd's Algorithm in order to compute all shortest paths, we will store the "dependence" of each entry.

For example, if the current zone is $x_2 \leq 5$, and if the next transition of the trace is $\xrightarrow{g,\{x_2\}}$ where g is $x_1 - x_2 \leq 3$, the next zone which is computed is $x_1 \leq 8 \wedge x_2 = 0 \wedge x_1 - x_2 \leq 8$, and we will store that the constraints $x_1 \leq 8$ and $x_1 - x_2 \leq 8$ depend on g (because they are inherited from the intersection of g with $x_2 \leq 5$). On the contrary, the constraint $x_2 = 0$ does not depend on g (it is solely due to the reset of clock x_2). For storing such a dependence information, we need to enrich the DBMs, we thus define the EDBM data structure:

Definition 1 (Extended DBM — EDBM). *An* extended DBM *is a pair $(M, \mathcal{S})$ of square matrices of size $n+1$ where M is a classical DBM and $\mathcal{S}$ is a matrix whose entries are non-empty sets of subsets of* $\mathsf{Diag}(\mathcal{A})$.

Let $(M, \mathcal{S})$ be an EDBM with $M = (m_{i,j})_{i,j=0\ldots n}$ and $\mathcal{S} = (\mathcal{S}_{i,j})_{i,j=0\ldots n}$. This EDBM represents the same zone as M, and the set $\mathcal{S}_{i,j}$ informally contains all diagonal guards on which entry $m_{i,j}$ may depend. In a weighted graph, it may exist several shortest paths between two vertices. The same holds for dependence sets: the set $\mathcal{S}_{i,j}$ may contain several subsets of $\mathsf{Diag}(\mathcal{A})$, each one contains sufficient information on the dependence of $m_{i,j}$ w.r.t. diagonal guards. Each set $G \in \mathcal{S}_{i,j}$ will be a candidate set for the refinement step whenever entry $m_{i,j}$ is detected as non correct. We store every possible set so as to choose the minimal (*i.e.* smallest) one at the end.

Operations on EDBMs. Given a constraint g, we use $\mathcal{S}et(g)$ to denote $\{\{g\}\}$ if g is diagonal and $\{\emptyset\}$ otherwise. We first define the two following basic operations on non-empty sets of sets:

(i) $\mathcal{S}_1 \curlyvee \mathcal{S}_2 = \{G \mid G \in \mathcal{S}_1 \cup \mathcal{S}_2\}$
(ii) $\mathcal{S}_1 \curlywedge \mathcal{S}_2 = \{G_1 \cup G_2 \mid G_1 \in \mathcal{S}_1 \text{ and } G_2 \in \mathcal{S}_2\}$

We can now extend classical operations on DBMs (needed by Algorithm FRA) to EDBMs. We consider two EDBMs $(M, \mathcal{S})$ and $(M', \mathcal{S}')$ with $M = (m_{i,j})_{i,j=0\dots n}$, $\mathcal{S} = (\mathcal{S}_{i,j})_{i,j=0\dots n}$, $M' = (m'_{i,j})_{i,j=0\dots n}$, and $\mathcal{S}' = (\mathcal{S}'_{i,j})_{i,j=0\dots n}$. We assume in addition that the DBM M is in normal form.

Future. $(M', \mathcal{S}') = \overrightarrow{(M, \mathcal{S})}$ whenever:

$$(m'_{i,j}, \mathcal{S}'_{i,j}) = \begin{cases} (\infty, \{\emptyset\}) & \text{if } j = 0 \\ (m_{i,j}, \mathcal{S}_{i,j}) & \text{otherwise} \end{cases}$$

Reset of clock x_k. $(M', \mathcal{S}') = [x_k \leftarrow 0](M, \mathcal{S})$ whenever

$$(m'_{i,j}, \mathcal{S}'_{i,j}) = \begin{cases} (0, \{\emptyset\}) & \text{if } i, j \in \{0, k\} \\ (m_{i,0}, \mathcal{S}_{i,0}) & \text{if } j = k, \\ (m_{0,j}, \mathcal{S}_{0,j}) & \text{if } i = k, \\ (m_{i,j}, \mathcal{S}_{i,j}) & \text{otherwise} \end{cases}$$

Intersection with $g = (x_k - x_l \le c)$. $(M', \mathcal{S}') = \mathsf{Inter}((M, \mathcal{S}), g)$ whenever:

$$(m'_{i,j}, \mathcal{S}'_{i,j}) = \begin{cases} (m_{i,j}, \mathcal{S}_{i,j}) & \text{if } m_{i,j} < m \\ (m_{i,j}, \mathcal{S}_{i,j} \curlyvee \overline{\mathcal{S}}) & \text{if } m_{i,j} = m \\ (m, \overline{\mathcal{S}}) & \text{if } m_{i,j} > m \end{cases}$$

where $m = m_{i,k} + c + m_{l,j}$ and $\overline{\mathcal{S}} = \mathcal{S}_{i,k} \curlywedge \mathcal{S}et(g) \curlywedge \mathcal{S}_{l,j}$

Note that the intersection operation contains a normalization step in order to tighten every entry w.r.t. the new constraint g. In fact, the resulting DBM M' is in normal form after each of these three operations. Following UPPAAL implementation of forward analysis [BBLP05], these operations are sufficient for computing the exact reachable zones along a trace. Computing successors w.r.t. an edge $\xrightarrow{g,r}$ is done by computing the future, then computing successively the intersection with all atomic guards forming g[2], and finally computing resets of all clocks in r:

$$\ell \xrightarrow{\wedge_{j=1}^{p} g_j, r} \ell' \quad \text{is transformed into} \quad \ell \overset{Fut.}{\longmapsto} \overset{\cap g_1}{\longmapsto} \cdots \overset{\cap g_p}{\longmapsto} \overset{r \leftarrow 0}{\longmapsto} \ell'$$

From now on, we decompose in this way every transition and then the whole trace ρ into elementary steps $(\alpha_i)_{i=1\dots p}$ (even if it is an abuse of notation, we write $\rho = \alpha_1 \dots \alpha_p$) such that each step is either an intersection with an atomic guard, a reset of clock, or a future operation. This decomposition allows us to consider each atomic guard successively and then to detect the atomic guard causing the error (emptiness is of course due to an intersection). Computing the symbolic execution along ρ then corresponds to applying successively each operation α_i (for $i = 1 \dots p$). We keep previous notations Z^e_ρ and Z^a_ρ, and we denote $(M_\rho, \mathcal{S}_\rho)$ the EDBM obtained after having applied successively operations α_i to the EDBM $(M_\epsilon, \mathcal{S}_\epsilon)$ where each entry of M_ϵ is 0, whereas each entry of $\mathcal{S}_\epsilon$ is $\{\emptyset\}$. Obviously, for every trace ρ, $[\![M_\rho]\!] = Z^e_\rho$. To ease the reading, we write $M_\rho(i,j)$ (resp. $\mathcal{S}_\rho(i,j)$) the entry (i,j) of the matrix M_ρ (resp. $\mathcal{S}_\rho$). We are now ready for presenting our algorithm which selects diagonal guards for the refinement step.

[2] An atomic guard is a guard of the form $x \sim c$ or $x - y \sim c$ with $\sim \in \{\le, \ge\}$.

4.3 Correctness and Progress of the Algorithm

Algorithm 2 Selection of diagonal guards – Select_guards

Algorithm Select_guards ($\rho = \alpha_1 \dots \alpha_p$ an erroneous trace in $\mathcal{A}$) {
    Initialize EDBMs $(M, \mathcal{S})$ and $(M', \mathcal{S}')$ to $(M_\epsilon, \mathcal{S}_\epsilon)$;
    i := 0;
    Repeat
        $i := i + 1$;
        $(M, \mathcal{S}) := (M', \mathcal{S}')$;
        If α_i is the future operation, $(M', \mathcal{S}') := \overrightarrow{(M, \mathcal{S})}$;
        If α_i is the intersection with g_i , $(M', \mathcal{S}')$:= Inter($(M, \mathcal{S}), g_i$);
        If α_i is the reset of clock x, $(M', \mathcal{S}') := [x \leftarrow 0](M, \mathcal{S})$;
    Until $[\![M']\!] = \emptyset$
    Return $\mathcal{S}_{l,k} \curlywedge \mathcal{S}et(g_i)$; }   ($\star$ the last α_i is a guard g_i of the form $x_k - x_l \leq c$ $\star$)[3]

Presentation of the algorithm. Our algorithm for selecting diagonal guards along an erroneous trace is presented as Algorithm 2 and is called Select_guards. At the i-th step of the iteration, the EDBM stored in $(M', \mathcal{S}')$ is the EDBM $(M_{\rho_i}, \mathcal{S}_{\rho_i})$ with $\rho_i = \alpha_1 \dots \alpha_{i-1}$. As ρ is an erroneous trace, there exists some $1 \leq i \leq p$ such that $[\![M_{\rho_i}]\!] \neq \emptyset$ whereas $[\![M_{\rho_i.\alpha_i}]\!] = \emptyset$. The elementary step α_i is an intersection with some guard $g_i = (x_k - x_l \leq c)$ and we get that $M_{\rho_i}(l, k) + c < 0$ (otherwise $[\![M_{\rho_i.\alpha_i}]\!]$ would not be empty). Note that this inequality does not hold for the abstract computation because $Z^a_{\rho_i.\alpha_i} \neq \emptyset$. As a consequence, we get that the entry (l, k) is not correct in the abstract computation (*i.e.* in $Z^a_{\rho_i}$). That's why Algorithm Select_guards outputs the dependence set $\mathcal{S}_{\rho_i}(l, k)$, and adds the guard g_i whenever it is a diagonal guard. The refinement step will split the original automaton along all diagonals in some $G \in \mathcal{S}_{\rho'}(l, k) \curlywedge \mathcal{S}et(g_i)$. Note that adding g_i is necessary: consider the automaton in Figure 3 in which the two last transitions are switched; there is no diagonal guard before the last transition. We will now prove correctness of this selection algorithm.

Correctness of the algorithm. If ρ is a trace, and G a subset of $\mathsf{Diag}(\mathcal{A})$, we denote $\rho[G \leftarrow \mathtt{tt}]$ the trace where the transitions labelled by some $g \in G$ are replaced by a transition labelled by the constraint $\mathtt{tt}$. Roughly, the first lemma states that diagonal guards which are not selected by our algorithm are not involved in the computation of the corresponding entry.

Lemma 1. *Let $\mathcal{A}$ be a timed automaton, and ρ a trace in $\mathcal{A}$. Consider a pair (i, j) and a set of diagonal guards G.*

$$\text{If } \Big(\exists G_0 \in \mathcal{S}_\rho(i, j) \text{ s.t. } G_0 \cap G = \emptyset\Big) \text{ then } M_{\rho[G \leftarrow \mathtt{tt}]}(i, j) = M_\rho(i, j).$$

Proof (Sketch). The proof can be done by induction on the length of the trace ρ. It relies on the fact that all guards which may have been used for computing the current entry

[3] k or l is possibly 0.

have been selected. Therefore, other guards can be removed safely. (See Appendix C for a complete proof.) $\square$

Our algorithm outputs a set of sets G of diagonal guards, each set G contains candidates for splitting the original automaton during the refinement step. The following proposition states that if there is an erroneous trace in a timed automaton, then each set G which is part of the output of our algorithm is non-empty. This proves correctness of the refinement step: each time we need to refine, we will be able to do so.

Proposition 1. *Let $\mathcal{A}$ be a timed automaton, and ρ an erroneous trace in $\mathcal{A}$. Then* $\mathsf{Select\_guards}(\rho)$ *does not contain the empty set.*

Proof. The proof is done by contradiction. Suppose this set contains the empty set. Using the previous lemma, we can remove all diagonal constraints appearing along this path and obtain an erroneous path which does not contain any diagonal constraint. This contradicts the correctness of the algorithm in the non-diagonal case. $\square$

The following proposition states that any set G selected by our algorithm is pertinent w.r.t. ρ. In the refined automaton $\mathsf{Split}(\mathcal{A}, G)$, the guards of G have been removed, and every location of $\mathcal{A}$ is split into $2^{|G|}$ locations so as to encode the truth value of guards in G. The trace ρ in $\mathcal{A}$ corresponds to a set of traces – denoted $\pi_{\mathcal{A},G}(\rho)$ – in $\mathsf{Split}(\mathcal{A}, G)$. The next proposition states that Algorithm FRA will not output as a witness trace any $\rho' \in \pi_{\mathcal{A},G}(\rho)$. This is formally expressed as follows:

Proposition 2. *Let $\mathcal{A}$ be a TA, and ρ an erroneous trace in $\mathcal{A}$. Let $G \in \mathsf{Select\_guards}(\rho)$ and $\mathcal{A}' = \mathsf{Split}(\mathcal{A}, G)$. For all $\rho' \in \pi_{\mathcal{A},G}(\rho)$, the abstract computation in $\mathcal{A}'$ for ρ' leads to an empty state, i.e. $Z^a_{\rho'} = \emptyset$*

Proof. $\rho = \rho_1.\alpha$ is an erroneous trace such that $Z^e_{\rho_1} \neq \emptyset$, $Z^e_\rho = \emptyset$ while $Z^a_\rho \neq \emptyset$. Assume that the entry which is not correct in $Z^a_{\rho_1}$ is (l, k). Algorithm $\mathsf{Select\_guards}$ returns $\mathcal{S}_{\rho_1}(l, k) \wedge \mathcal{S}et(g)$ where g is the guard labelling α.
Now consider $\overline{G} = \mathsf{Diag}(\mathcal{A}) \backslash G$. Lemma 1 ensures that the trace $\rho_1[\overline{G} \leftarrow \mathtt{tt}]$ leads in $\mathcal{A}$ to a zone with the same entry (l, k) *via* an exact forward computation: $M_{\rho_1[\overline{G} \leftarrow \mathtt{tt}]}(l, k) = M_{\rho_1}(l, k)$. Emptiness of Z^e_ρ then implies emptiness of $Z^e_{\rho[\overline{G} \leftarrow \mathtt{tt}]}$.
Let $\mathcal{A}_1$ be the timed automaton obtained from $\mathcal{A}$ by replacing any guard in $\overline{G}$ by $\mathtt{tt}$. The trace $\rho[\overline{G} \leftarrow \mathtt{tt}]$ is a trace of $\mathcal{A}_1$ and it leads also to an empty zone, as for the computation in $\mathcal{A}$, because this depends only on the trace and not on the automaton.
Now let $\mathcal{A}_2$ be $\mathsf{Split}(\mathcal{A}_1, G)$. Any trace ρ_2 in $\pi_{\mathcal{A}_1,G}(\rho[\overline{G} \leftarrow \mathtt{tt}])$ leads, as $\mathcal{A}_1$ along $\rho[\overline{G} \leftarrow \mathtt{tt}]$, to an empty zone: $Z^e_{\rho_2} = Z^e_{\rho[\overline{G} \leftarrow \mathtt{tt}]} = \emptyset$. There is no more diagonal constraint in $\mathcal{A}_2$. Thus, the abstract forward computation along trace ρ_2 leads also to an empty zone: $Z^a_{\rho_2} = \emptyset$.
Clearly $\mathcal{A}_2$ accepts more runs than $\mathcal{A}' = \mathsf{Split}(\mathcal{A}, G)$ because some guards have been replaced by $\mathtt{tt}$. This holds for the exact and the abstract computation. Any trace ρ' in $\pi_{\mathcal{A},G}(\rho)$ thus leads to an empty zone in the abstract computation: $Z^a_{\rho'} = \emptyset$. $\square$

The two propositions show the correctness of our algorithm for selecting diagonal constraints.

4.4 Comments on the Method

When ρ is an erroneous trace, $\mathsf{Select\_guards}(\rho)$ may output several non-empty sets of diagonal guards. For the refinement step, we can choose any such set. In particular, we can choose the smallest one, so that the refinement is not too expensive.

The key point of the method we propose is the $\mathsf{Select\_guards}$ algorithm because it may avoid the removal of all diagonal constraints. Note that our method is not formally optimal in the sense that it is possible that refining w.r.t. a proper subset of the selected set is sufficient to treat correctly the current trace. However, we may find such a subset with an incremental test: add successively each selected guard until the erroneous trace is eliminated (this procedure is linear in $|\rho|$). It is clearly more efficient than any method consisting in splitting every constraint in $\mathsf{Diag}(\mathcal{A})$, especially because most of the diagonal constraints do not raise wrong diagnosis. For example, as explained in [BY04], they are very useful for modeling scheduling problems (see for example [FPY02]). Such systems contain a lot of diagonal constraints[4] but, as the clocks of these systems are bounded, our algorithm will never find erroneous traces, and no refinement is needed (for this class of models the classical forward algorithm FRA will be correct). Then a systematic splitting is expensive and useless. In Appendix D, we give several examples of computations of Algorithm $\mathsf{Select\_guards}$.

The splitting step could be implemented in several ways and we could avoid building explicitly $\mathsf{Split}(\mathcal{A}, G)$. First we could consider on-the-fly techniques where the truth of guards in G are stored using a vector of booleans. Another possibility could be to modify the computation of Post operator in order to integrate the splitting of diagonal constraints.

4.5 Related Work

Several refinement-based methods have already been proposed in the past for timed systems [AIKY95,MRS02,Sor04] and more generally for hybrid systems [ADI03]. In these works the verification process starts assuming there is no timing information in the system, and then, when a spurious trace is found, the system is refined using relevant constraints (or predicates) of this spurious trace. For timed systems, predicates which are used for refining the model are predicates which separate regions, which ensures that the refinement process stops. In those works, abstraction and refinement are used either to avoid computing the regions or to compute the coarsest time-abstract bisimulation [TY01] before verifying the system.

In our work, the aim and the techniques are different. We do not want to propose a verification algorithm fully based on abstraction and refinement, but we want to use the refinement paradigm for patching the classical forward analysis algorithm with no over-cost when no spurious trace is detected. Indeed, this algorithm, which is for example implemented in UPPAAL, has already proven its efficiency in many case studies. Moreover, as this algorithm is correct for TA without diagonal constraints, we select predicates only among the diagonal constraints of the TA we verify.

[4] There are n^2 diagonal constraints where n is the number of tasks to be scheduled.

5 Conclusion

In this paper we have studied the role of diagonal constraints in a forward analysis computation. We have described several approaches to handle diagonal constraints and proposed a refinement-based purely forward method for verifying reachability properties of timed automata with diagonal constraints. As diagonal constraints do not always raise wrong diagnosis, a systematic removal of all diagonal constraints appears as too expensive, and a refinement-based learning from erroneous traces of the classical algorithm seems more appropriate to patch the usual algorithm, as there will be no over-cost if no spurious trace is found. We think that, in practice, the cost of this approach is lower than the cost of the systematic removal of all diagonal constraints. As further developments, we would like to combine techniques we have proposed in this paper with more efficient refinement-based methods, as the lazy abstraction approach of [HJMS02]. We also plan to implement this method and to compare it with other approaches.

References

[ACD93] Rajeev Alur, Costas Courcoubetis, and David Dill. Model-checking in dense real-time. *Information and Computation*, 104(1):2–34, 1993.

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

[ADI03] Rajeev Alur, Thao Dang, and Franjo Ivančić. Counter-example guided predicate abstraction of hybrid systems. In *9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 2619 of *Lecture Notes in Computer Science*, pages 208–223. Springer, 2003.

[AIKY95] Rajeev Alur, Alon Itai, Robert P. Kurshan, and Mihalis Yannakakis. Timing verification by successive approximation. *Information and Computation*, 118(1):142–157, 1995.

[BBFL03] Gerd Behrmann, Patricia Bouyer, Emmanuel Fleury, and Kim G. Larsen. Static guard analysis in timed automata verification. In *Proc. 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'03)*, volume 2619 of *Lecture Notes in Computer Science*, pages 254–277. Springer, 2003.

[BBLP05] Gerd Behrmann, Patricia Bouyer, Kim G. Larsen, and Radek Pelànek. Zone based abstractions for timed automata exploiting lower and upper bounds. *Software Tools for Technology Transfer*, 2005. To appear.

[BC05] Patricia Bouyer and Fabrice Chevalier. On conciseness of extensions of timed automata. *Journal of Automata, Languages and Combinatorics*, 2005. To appear.

[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.

[BM83] Bernard Berthomieu and Miguel Menasche. An enumerative approach for analyzing time Petri nets. In *Proc. IFIP 9th World Computer Congress*, volume 83 of *Information Processing*, pages 41–46. North-Holland/ IFIP, 1983.

[Bou03] Patricia Bouyer. Untameable timed automata! In *Proc. 20th Annual Symposium on Theoretical Aspects of Computer Science (STACS'03)*, volume 2607 of *Lecture Notes in Computer Science*, pages 620–631. Springer, 2003.

[Bou04] Patricia Bouyer. Forward analysis of updatable timed automata. *Formal Methods in System Design*, 24(3):281–320, 2004.

[BY03] Johan Bengtsson and Wang Yi. On clock difference constraints and termination in reachability analysis of timed automata. In *Proc. 5th International Conference on Formal Engineering Methods (ICFEM 2003)*, volume 2885 of *Lecture Notes in Computer Science*, pages 491–503. Springer, 2003.

[BY04] Johan Bengtsson and Wang Yi. Timed automata: Semantics, algorithms and tools. In *Proc. 4th Advanced Course on Petri Nets (ACPN'03)*, volume 3098 of *Lecture Notes in Computer Science*, pages 87–124. Springer, 2004.

[CGJ+00] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement. In *Proc. 12th International Conference on Computer Aided Verification (CAV'00)*, volume 1855 of *Lecture Notes in Computer Science*, pages 154–169. Springer, 2000.

[Dil90] David Dill. Timing assumptions and verification of finite-state concurrent systems. In *Proc. of the Workshop on Automatic Verification Methods for Finite State Systems (1989)*, volume 407 of *Lecture Notes in Computer Science*, pages 197–212. Springer, 1990.

[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.

[DY96] Conrado Daws and Sergio Yovine. Reducing the number of clock variables of timed automata. In *Proc. 17th IEEE Real-Time Systems Symposium (RTSS'96)*, pages 73–81. IEEE Computer Society Press, 1996.

[FPY02] Elena Fersman, Paul Petterson, and Wang Yi. Timed automata with asynchrounous processes: Schedulability and decidability. In *Proc. 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'02)*, volume 2280 of *Lecture Notes in Computer Science*, pages 67–82. Springer, 2002.

[HJMS02] Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, and Grégoire Sutre. Lazy abstraction. In *Proc. 29th ACM Symposium on Principles of Programming Languages (POPL'02)*, pages 58–70. ACM Press, 2002.

[HNSY94] Thomas A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic model-checking for real-time systems. *Information and Computation*, 111(2):193–244, 1994.

[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.

[MRS02] M. Oliver Möller, Harald Rueß, and Maria Sorea. Predicate abstraction for dense real-time systems. In *Proc. Theory and Practice of Timed Systems (TPTS'02)*, volume 65(6) of *Electronic Notes in Theoretical Computer Science*, pages 1–20. Elsevier, 2002.

[Sor04] Maria Sorea. Lazy approximation for dense real-time systems. In *Proc. Joint Conference on Formal Modelling and Analysis of Timed Systems and Formal Techniques in Real-Time and Fault Tolerant System (FORMATS+FTRTFT'04)*, volume 3253 of *Lecture Notes in Computer Science*, pages 363–378. Springer, 2004.

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

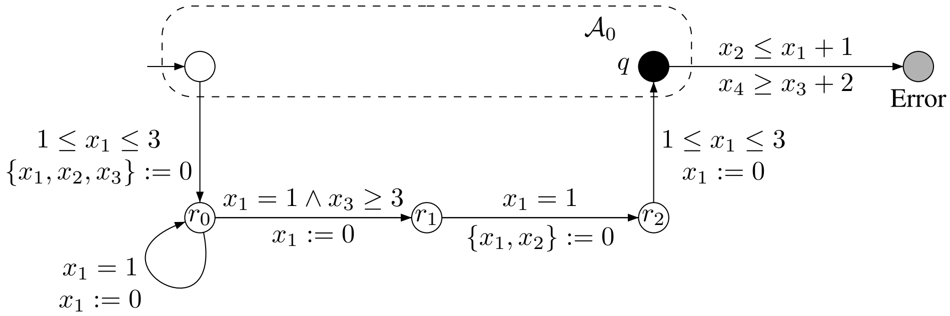
A Note on Automaton $\mathcal{A}$ of Figure 1

Consider the automaton $\mathcal{A}$ depicted in Figure 1. Consider a run along which the first transition is performed at date d (with $1 \leq d \leq 3$), and where the loop is taken α times, then the clock valuation v when arriving in the black state verifies: $v(x_1) = 0$, $v(x_2) = d$, $v(x_3) = 2\alpha + 5$ and $v(x_4) = 2\alpha + 5 + d$. In particular, the constraint $x_2 - x_1 = x_4 - x_3$ is always satisfied, which implies that the last transition is never enabled, and thus that state "Error" is not reachable. If α is large enough, then the exact and abstracted symbolic computations along the corresponding trace ρ are respectively the following zones [5]:

$$Z^e_\rho \begin{cases} x_1 = 0 \\ 1 \leq x_2 - x_1 \leq 3 \\ 1 \leq x_4 - x_3 \leq 3 \\ x_3 - x_1 = 2\alpha + 5 \\ x_4 - x_2 = 2\alpha + 5 \end{cases} \qquad\qquad Z^a_\rho \begin{cases} x_1 = 0 \\ 1 \leq x_2 - x_1 \leq 3 \\ 1 \leq x_4 - x_3 \leq 3 \\ x_3 - x_1 > k \\ x_4 - x_2 > k \end{cases}$$

Let g be the guard $x_2 - x_1 \leq 1 \wedge x_4 - x_3 \geq 2$. We clearly have $Z^e_\rho \cap g = \emptyset$ but $Z^a_\rho \cap g \neq \emptyset$, and then the last transition is found as reachable by Algorithm FRA. Note that strict inequalities are due to the extrapolation operator.

B "Checking Witness Traces, and Running Further the Algorithm" is not Correct (Section 3)



Consider the automaton depicted above and assume that the extrapolation constant is 3. An execution going through the states r_0, r_1, r_2 until q allows to reach the Error location. Indeed with such an execution, the (exact) valuation when arriving in q depends on three parameters: the amount of time δ_0 spent in the initial state ($1 \leq \delta_0 \leq 3$), the number β of times the loop on r_0 has been taken ($\beta \geq 3$), and the amount of time δ_1 spent in r_2 ($1 \leq \delta_1 \leq 3$). The valuation v is defined by: $v(x_1) = 0$, $v(x_2) = \delta_1$, $v(x_3) = \beta + 2 + \delta_1$ and $v(x_4) = \delta_0 + \beta + 2 + \delta_1$. Then it is sufficient to choose $\delta_0 = 1$ and $\delta_1 = 2$ to satisfy the constraint $x_4 - x_3 \geq 2 \wedge x_2 - x_1 \leq 1$.

[5] Computations have been done automatically to present the zones in a concise way.

Now consider Algorithm FRA, we assume it uses a breadth first search. It finds the abstract symbolic execution leading to the black state q in 5 edges (without taking the loop) through $\mathcal{A}_0$: the abstract computed zone is $Z_1^a = 3 < x_3 \wedge 1 \leq x_2 \leq 3 \wedge x_1 = 0 \wedge 1 \leq x_4 - x_3 \leq 3 \wedge 3 < x_4 - x_2 \wedge 2 \leq x_3 - x_2$ and then the symbolic state (q, Z_1^a) is stored in $\mathsf{Visited}$. From this state, the location Error is found reachable, then the algorithms returns the corresponding witness. As it is a false positive, the algorithm continues its breadth first search. Then it will discover the symbolic state (r_2, Z_2^a) – reaching this state requires 6 edges – and the corresponding abstracted zone is $Z_2^a = 3 < x_3 \wedge x_1 = 0 \wedge 1 \leq x_4 - x_3 \leq 3 \wedge x_2 - x_1 \leq 0$. And we have: $\mathsf{Extra}_3(\mathsf{Post}(Z_2^a, \xrightarrow{1 \leq x_1 \leq 3, x_1 := 0})) = 1 \leq x_2 \leq 3 \wedge x_1 = 0 \wedge 1 \leq x_4 - x_3 \leq 3 \wedge 3 < x_3 - x_2$, it is included in Z_1^a, and its successor will not be considered because it already belongs to $\mathsf{Visited}$. Then the correct execution leading to Error won't be discovered by the algorithm.

C Proof of Lemma 1

First we have:

Lemma 2. *Let ρ be a path in a timed automaton $\mathcal{A}$ and G a set of diagonal guards. Then, for all pairs (i, j), $M_\rho(i, j) \leq M_{\rho[G \leftarrow \mathtt{tt}]}(i, j)$.*

Proof. By definition, the path $\rho[G \leftarrow \mathtt{tt}]$ is less restrictive than the path ρ. We have thus the following inclusions between zones: $Z_\rho^e \subseteq Z_{\rho[G \leftarrow \mathtt{tt}]}^e$. □

The proof of Lemma 1 is done by induction on the length of ρ: we consider a path ρ_1 and we extend it with an edge $\xrightarrow{\alpha}$ to get ρ_2. We assume the property holds for ρ_1, and we will prove it for ρ_2 by distinguishing possible cases for α. We fix two indices i and j and a set of diagonal guards G_0 such that $\exists G \in \mathcal{S}_{\rho_2}(i, j)$ s.t. $G_0 \cap G = \emptyset$. We fix such a G.

- **Case α is the constraint $(\gamma : x_k - x_l \leq c)$.**
  Let first recall the definition of the operator $\mathsf{Inter}(M, \gamma)$:

$$\mathcal{S}_{\rho_2}(i, j) = \begin{cases} \mathcal{S}_{\rho_1}(i, j) & \text{if } M_{\rho_1}(i, j) < m \\ \mathcal{S}_{\rho_1}(i, j) \curlyvee \mathcal{S}_0 & \text{if } M_{\rho_1}(i, j) = m \\ \mathcal{S}_0 & \text{if } M_{\rho_1}(i, j) > m \end{cases}$$

  where $m = M_{\rho_1}(i, k) + c + M_{\rho_1}(l, j)$ and $\mathcal{S}_0 = \mathcal{S}_{\rho_1}(i, k) \curlywedge \mathcal{S}et(\gamma) \curlywedge \mathcal{S}_{\rho_1}(l, j)$. Therefore, as $G \in \mathcal{S}_{\rho_2}(i, j)$, we have $G \in \mathcal{S}_{\rho_1}(i, j)$ or $G \in \mathcal{S}_0$ (or both) and so we always can apply the induction hypothesis to at least one of $\mathcal{S}_{\rho_1}(i, j)$ and $\mathcal{S}_0$.
  - **(a)** We apply induction hypothesis to $\mathcal{S}_{\rho_1}(i, j)$.
    By definition of Inter, this can occur if $M_{\rho_1}(i, j) \leq M_{\rho_1}(i, k) + c + M_{\rho_1}(l, j)$. It's what we assume now. Then we have $M_{\rho_2}(i, j) = M_{\rho_1}(i, j)$. By induction hypothesis, we get $M_{\rho_1}(i, j) = M_{\rho_1[G_0 \leftarrow \mathtt{tt}]}(i, j)$. Two cases may arise for $M_{\rho_2[G_0 \leftarrow \mathtt{tt}]}(i, j)$:
    - (i) either $M_{\rho_1[G_0 \leftarrow \mathtt{tt}]}(i, j) \leq M_{\rho_1[G_0 \leftarrow \mathtt{tt}]}(i, k) + c + M_{\rho_1[G_0 \leftarrow \mathtt{tt}]}(l, j)$
    - (ii) or $M_{\rho_1[G_0 \leftarrow \mathtt{tt}]}(i, j) > M_{\rho_1[G_0 \leftarrow \mathtt{tt}]}(i, k) + c + M_{\rho_1[G_0 \leftarrow \mathtt{tt}]}(l, j)$.

In the first case, we get that $M_{\rho_2[G_0 \leftarrow \mathtt{t}]}(i,j) = M_{\rho_1[G_0 \leftarrow \mathtt{t}]}(i,j) = M_{\rho_1}(i,j) = M_{\rho_2}(i,j)$ this gives the result.
We will prove that the second case cannot happen. By definition of Inter, we have in this case $M_{\rho_2[G_0 \leftarrow \mathtt{t}]}(i,j) = M_{\rho_1[G_0 \leftarrow \mathtt{t}]}(i,k) + c + M_{\rho_1[G_0 \leftarrow \mathtt{t}]}(l,j)$.

By induction hypothesis, we have $M_{\rho_1}(i,j) = M_{\rho_1[G_0 \leftarrow \mathtt{t}]}(i,j)$. By (ii), we have $M_{\rho_1[G_0 \leftarrow \mathtt{t}]}(i,j) > M_{\rho_1[G_0 \leftarrow \mathtt{t}]}(i,k) + c + M_{\rho_1[G_0 \leftarrow \mathtt{t}]}(l,j)$. By Lemma 2, we have $M_{\rho_1[G_0 \leftarrow \mathtt{t}]}(i,k) \geq M_{\rho_1}(i,k)$ and $M_{\rho_1[G_0 \leftarrow \mathtt{t}]}(l,j) \geq M_{\rho_1}(l,j)$, then $M_{\rho_1}(i,j) > M_{\rho_1}(i,k) + c + M_{\rho_1}(l,j)$ which contradicts the initial hypothesis.

- **(b)** We apply induction hypothesis to $\mathcal{S}_0 = \mathcal{S}_{\rho_1}(i,k) \Cap \mathcal{S}et(\gamma) \Cap \mathcal{S}_{\rho_1}(l,j)$. By definition of Inter, this can occur if $M_{\rho_1}(i,j) \geq M_{\rho_1}(i,k) + c + M_{\rho_1}(l,j)$. It's what we assume now. Then we have $M_{\rho_2}(i,j) = M_{\rho_1}(i,k) + c + M_{\rho_1}(l,j)$. As $G \in \mathcal{S}_0$, we have $G = G_1 \cup \{\gamma\} \cup G_2$ (or $G = G_1 \cup G_2$ whether γ is diagonal or not), where $G_1 \in \mathcal{S}_{\rho_1}(i,k)$ and $G_2 \in \mathcal{S}_{\rho_1}(l,j)$. Then we can apply induction hypothesis to both $\mathcal{S}_{\rho_1}(i,k)$ and $\mathcal{S}_{\rho_1}(l,j)$ and finally we get $M_{\rho_1}(i,k) = M_{\rho_1[G_0 \leftarrow \mathtt{t}]}(i,k)$ and $M_{\rho_1}(j,l) = M_{\rho_1[G_0 \leftarrow \mathtt{t}]}(j,l)$. Two cases may arise for $M_{\rho_2[G_0 \leftarrow \mathtt{t}]}(i,j)$:

  * (i) either $M_{\rho_1[G_0 \leftarrow \mathtt{t}]}(i,j) \geq M_{\rho_1[G_0 \leftarrow \mathtt{t}]}(i,k) + c + M_{\rho_1[G_0 \leftarrow \mathtt{t}]}(l,j)$
  * (ii) or $M_{\rho_1[G_0 \leftarrow \mathtt{t}]}(i,j) < M_{\rho_1[G_0 \leftarrow \mathtt{t}]}(i,k) + c + M_{\rho_1[G_0 \leftarrow \mathtt{t}]}(l,j)$.

  In the first case, we get that $M_{\rho_2[G_0 \leftarrow \mathtt{t}]}(i,j) = M_{\rho_1[G_0 \leftarrow \mathtt{t}]}(i,k) + c + M_{\rho_1[G_0 \leftarrow \mathtt{t}]}(l,j) = M_{\rho_1}(i,k) + c + M_{\rho_1}(l,j) = M_{\rho_2}(i,j)$ which provides the result.
  We will prove that the second case cannot happen. By definition of Inter , we have in this case $M_{\rho_2[G_0 \leftarrow \mathtt{t}]}(i,j) = M_{\rho_1[G_0 \leftarrow \mathtt{t}]}(i,j)$. By (ii), we have $M_{\rho_1[G_0 \leftarrow \mathtt{t}]}(i,j) < M_{\rho_1[G_0 \leftarrow \mathtt{t}]}(i,k) + c + M_{\rho_1[G_0 \leftarrow \mathtt{t}]}(l,j)$. The ind. hyp. gives $M_{\rho_1[G_0 \leftarrow \mathtt{t}]}(i,k) = M_{\rho_1}(i,k)$ and $M_{\rho_1[G_0 \leftarrow \mathtt{t}]}(l,j) = M_{\rho_1}(l,j)$, then we have: $M_{\rho_1[G_0 \leftarrow \mathtt{t}]}(i,j) < M_{\rho_1}(i,k) + c + M_{\rho_1}(l,j)$. But the initial hypothesis gives $M_{\rho_1}(i,k) + c + M_{\rho_1}(l,j) = M_{\rho_2}(i,j)$. The Lemma 2 provides $M_{\rho_2}(i,j) \leq M_{\rho_2[G_0 \leftarrow \mathtt{t}]}(i,j)$ what gives us a contradiction.

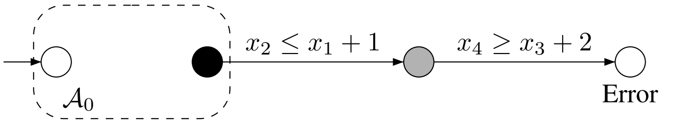
– **Case α is the future.**
If $j \neq 0$, then by the definition of $\overrightarrow{(M,\mathcal{S})}$, , we have $\mathcal{S}_{\rho_2}(i,j) = \mathcal{S}_{\rho_1}(i,j)$. So we can apply the induction hypothesis to $\mathcal{S}_{\rho_1}(i,j)$ what gives us $M_{\rho_1}(i,j) = M_{\rho_1[G_0 \leftarrow \mathtt{t}]}(i,j)$. Moreover, we also have $M_{\rho_2[G_0 \leftarrow \mathtt{t}]}(i,j) = M_{\rho_1[G_0 \leftarrow \mathtt{t}]}(i,j)$ and $M_{\rho_2}(i,j) = M_{\rho_1}(i,j)$ because $j \neq 0$. We thus get the result we want.
If $j = 0$, both $M_{\rho_2[G_0 \leftarrow \mathtt{t}]}(i,j)$ and $M_{\rho_2}(i,j)$ are $+\infty$.

– **Case α is the reset of clock x_k.**
In case $(i,j) \in \{(k,k),(0,k),(k,0)\}$, both $M_{\rho_2[G_0 \leftarrow \mathtt{t}]}(i,j)$ and $M_{\rho_2}(i,j)$ are 0.
If $j = k$ (resp. $i = k$), we can apply the induction hypothesis to the set $\mathcal{S}_{\rho_1}(i,0)$ (resp. $\mathcal{S}_{\rho_1}(0,j)$). As above, we obtain the result we want.
Otherwise we can apply induction hypothesis to $\mathcal{S}_{\rho_1}(i,j)$ and conclude in the same way.

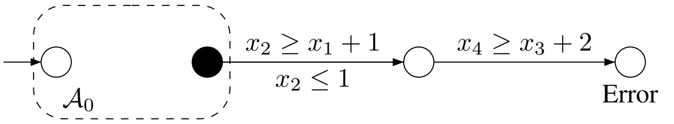
Finally, we obtain in every case that $M_{\rho_2}(i,j) = M_{\rho_2[G_0 \leftarrow \mathtt{t}]}(i,j)$, what ends the proof.

D Examples of Applications

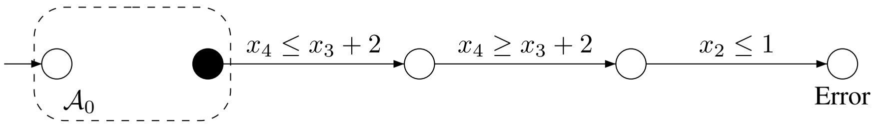
In this section we illustrate how Algorithm Select_guards behaves on several small examples. To help understanding these examples we recall that the zone obtained by an exact computation when reaching the black state is $x_1 = 0 \wedge 1 \leq x_2 - x_1 \leq 3 \wedge 1 \leq x_4 - x_3 \leq 3 \wedge x_3 - x_1 = 2\alpha + 5 \wedge x_4 - x_2 = 2\alpha + 5$ (see Appendix A).



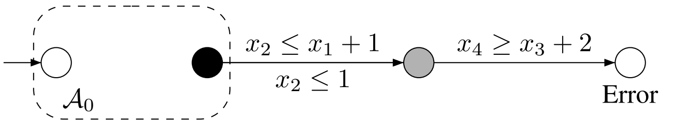
When reaching the grey state, the constraint on the difference $x_4 - x_3$ is $x_4 - x_3 \leq 1$, and this is due to diagonal guard $x_2 \leq x_1 + 1$. Algorithm Select_guards thus outputs $\{\{x_2 \leq x_1 + 1, x_4 \geq x_3 + 2\}\}$. In this case, too many diagonal guards are selected, because it would be sufficient to split along one of the two diagonal constraints.



For the above automaton, Algorithm Select_guards outputs $\{\{x_4 \geq x_3 + 2\}\}$. Only one of the two diagonal constraints is selected.



For the above automaton, Algorithm Select_guards outputs $\{\{x_4 \geq x_3 + 2\}\}$. Only one of the two diagonal constraints is selected.



For the above automaton, Algorithm Select_guards outputs $\{\{x_4 \geq x_3 + 2\}, \{x_4 \geq x_3 + 2, x_2 \leq x_1 + 1\}\}$: there are two possible sets of diagonal constraints which can be used for splitting. This is because the constraint $x_2 - x_1 \leq 1$ in the grey state is due to $x_2 \leq 1$ or $x_2 \leq x_1 + 1$. As explained in the paper, when computing normal forms of DBMs, it may be the case that, as in Floyd's algorithm, a minimal path can be obtained in two different ways. Of course, for splitting, we choose the set of guards $\{x_4 \geq x_3 + 2\}$ which is the smallest one.