Using non-convex approximations for efficient analysis of timed automata: Extended version

Frédéric Herbreteau, D. Kini, B. Srivathsan, Igor Walukiewicz

Using non-convex approximations for efficient analysis of timed automata: Extended version

F. Herbreteau[1], D. Kini[2], B. Srivathsan[1] and I. Walukiewicz[1]

[1] Université de Bordeaux, IPB, Université Bordeaux 1, CNRS, LaBRI UMR5800
LaBRI Bât A30, 351 crs Libération, 33405 Talence, France
[2] Indian Institute of Technology Bombay, Dept. of Computer Science and Engg.,
Powai, Mumbai 400076, India

Abstract. The reachability problem for timed automata asks if there exists a path from its initial state to a given target state. The standard solution to this problem involves computing the zone graph of the automaton, which in principle could be infinite. In order to make the graph finite, zones are approximated using an extrapolation operator. For reasons of efficiency it is required that an extrapolation of a zone is always a zone; and in particular that it is convex.
In this paper, we propose to solve the reachability problem without such extrapolation operators. To ensure termination, we provide an efficient algorithm to check if a zone is included in the so called region closure of another. Although theoretically better, closure cannot be used in the standard algorithm since a closure of a zone may not be convex.
The structure of this new algorithm permits to calculate approximating parameters on-the-fly during exploration of the zone graph, as opposed to the current methods which do it by a static analysis of the automaton prior to the exploration. This allows for further improvements in the algorithm. Promising experimental results are presented.

1 Introduction

Timed automata [1] are obtained from finite automata by adding clocks that can be reset and whose values can be compared with constants. The crucial property of timed automata is that their reachability problem is decidable: one can check if a given target state is reachable from the initial state. Reachability algorithm is at the core of verification tools like Uppaal [4] or Kronos [7], and is used in industrial case studies [11, 5]. The standard solution to this problem involves computing, so called, zone graph of the automaton, and the use of approximation to ensure termination.

We propose a new algorithm for the problem using a very simple approximation operation: region closures of zones. Additionally, we show that parameters for approximations can be calculated on-the-fly, and that this calculation can benefit from the information gathered during the analysis. In certain cases this allows to limit the search space substantially. A critical point of our algorithm is the avoidance of storing closures explicitly. We show how to check if a zone is

included in the closure of another zone at the same cost as checking the inclusion between two zones. Thanks to this efficient inclusion test our algorithm can work directly on the zone graph.

The first solution to the reachability problem has been proposed in the paper introducing timed automata [1]. It is based on *regions*: equivalence classes of clock valuations. Their definition is parametrized by a threshold up to which the clock values should be considered. Subsequent research has shown that the region abstraction is very inefficient. Another method using *zones* instead of regions has been proposed which can be implemented efficiently using DBMs [10]. It is used at present in all timed-verification tools.

The number of reachable zones can be infinite, so one needs an abstraction operator to get a finite approximation. The simplest is to approximate a zone with a set of regions it intersects: so called *closure* of a zone. Unfortunately closure may not always be convex and no efficient representation of closures is known. For this reason implementations use another approximation that is also based on (refined) regions.

Another important step in efficient implementation is calculation of thresholds for approximations. Definition of an approximation depends on the regions that in turn depend on the choice of a threshold. A safe choice is to take the maximal constant appearing in the transitions of the automaton. A considerable gain in efficiency can be obtained by analyzing the graph of the automaton and calculating thresholds specific for each clock and state of the automaton [2]. An even more efficient approach is the so called LU-approximation that distinguishes between upper and lower bounds [3]. This is the method used in the current implementation of UPPAAL.

We present a different approach to the reachability testing. Our algorithm works directly on zones: the reachability tree it constructs is labeled with zones. In order to ensure termination we use the simple zone closure operator. We do not explore a zone if it is contained in the closure of another zone in the search tree. Since we have no efficient way to represent closures, we compute them each time an inclusion test is performed. We show that we can do this at no additional cost: we provide an algorithm of the same complexity as that for comparing two zones. Moreover the inclusion test can accommodate LU-approximation at no additional cost. Another important feature of our algorithm is that its structure permits to compute thresholds for closure or LU approximations on-the-fly. We show how to use information gathered during reachability analysis to calculate better thresholds. This way of calculating thresholds adapts well to a very common case of analysis of parallel compositions of timed automata.

The plan of the paper is as follows. The next section presents the basic notions and recalls some of their properties. Section 3 describes the new algorithm. The efficient inclusion test used by the algorithm is presented in Section 4. Some results obtained with a prototype implementation are presented in the last section.

2 Deciding State Reachability in Timed Automata

2.1 Timed automata and the reachability problem

Let X be a set of clocks, i.e., variables that range over $\mathbb{R}_{\geq 0}$, the set of non-negative real numbers. *Clock constraints* are conjunctions of comparisons of clocks with integer constants, e.g. $(x \leq 3 \wedge y > 0)$. More formally, a clock constraint is a conjunction of constraints $x \# c$ for $x \in X$, $\# \in \{<, \leq, =, \geq, >\}$ and $c \in \mathbb{N}$. Let $\Phi(X)$ denote the set of clock constraints over clock variables X.

A *clock valuation* over X is a function $\nu : X \to \mathbb{R}_{\geq 0}$. We denote $\mathbb{R}_{\geq 0}^X$ the set of clock valuations over X, and $\mathbf{0}$ the valuation that associates 0 to every clock in X. We write $\nu \vDash \phi$ when ν satisfies $\phi \in \Phi(X)$, i.e. when every constraint in ϕ holds after replacing every x by $\nu(x)$. For $\delta \in \mathbb{R}_{\geq 0}$, let $\nu + \delta$ be the valuation that associates $\nu(x) + \delta$ to every clock x. For $R \subseteq X$, let $[R]\nu$ be the valuation that sets x to 0 if $x \in R$, and that sets x to $\nu(x)$ otherwise.

A *Timed Automaton (TA)* is a tuple $\mathcal{A} = (Q, q_0, X, T, Acc)$ where Q is a finite set of states, $q_0 \in Q$ is the initial state, X is a finite set of clocks, $Acc \subseteq Q$ is a set of accepting states, and $T \subseteq Q \times \Phi(X) \times 2^X \times Q$ is a finite set of transitions (q, g, R, q') where g is a *guard*, and R is the set of clocks that are *reset* on the transition. Examples of TA are depicted in Figure 3. The class of TA we consider is usually known as diagonal-free TA since clock comparisons like $x - y \leq 1$ are disallowed. Notice that since we are interested in state reachability, considering timed automata without state invariants does not entail any loss of generality.

A *configuration* of $\mathcal{A}$ is a pair $(q, \nu) \in Q \times \mathbb{R}_{\geq 0}^X$; $(q_0, \mathbf{0})$ is the *initial configuration*. We denote $(q, \nu) \xrightarrow{\delta, t} (q', \nu')$ if there exists $\delta \in \mathbb{R}_{\geq 0}$ and a transition $t = (q, g, R, q')$ in $\mathcal{A}$ such that $\nu + \delta \vDash g$, and $\nu' = [R]\nu$. Then (q', ν') is called a *successor* of (q, ν). A *run* of $\mathcal{A}$ is a finite sequence of transitions:

$$(q_0, \nu_0) \xrightarrow{\delta_0, t_0} (q_1, \nu_1) \xrightarrow{\delta_1, t_1} \cdots (q_n, \nu_n)$$

starting from $(q_0, \nu_0) = (q_0, \mathbf{0})$. A run is *accepting* if it ends in a configuration (q_n, ν_n) with $q_n \in Acc$.

We are interested in deciding if a given timed automaton $\mathcal{A}$ has an accepting run which in turn consists in deciding if there exists a reachable configuration (q, ν) with $q \in Acc$. This problem is known to be PSPACE-complete [1, 8].

One cannot directly use classical graph algorithms like a depth-first search to solve the problem, as a configuration (q, ν) may have uncountably many successors. We introduce a symbolic semantics for timed automata that permits to enumerate the successors of a node.

2.2 Symbolic semantics for timed automata

A symbolic semantics for timed automata considers sets of (uncountably many) valuations instead of valuations separately. A *zone* is a set of valuations defined by a conjunction of two kinds of constraints: comparison of difference between two clocks with an integer like $x - y \# c$, or comparison of a single clock with an

integer like $x\#c$, where $\# \in \{<, \leq, =, \geq, >\}$ and $c \in \mathbb{N}$. For instance $(x - y \geq 1) \wedge (y < 2)$ is a zone.

The transition relation on valuations is transferred to zones as follows. We have $(q, Z) \xrightarrow{t} (q', Z')$ if Z' is the set of valuations ν' such that $(q, \nu) \xrightarrow{\delta,t} (q', \nu')$ for some $\nu \in Z$ and $\delta \in \mathbb{R}_{\geq 0}$. The node (q', Z') is called a successor of (q, Z). It can be checked that if Z is a zone, then Z' is also a zone. The *zone graph* of $\mathcal{A}$, denoted $ZG(\mathcal{A})$, has nodes of the form (q, Z) with initial node $(q_0, \{\mathbf{0}\})$, and edges defined as above. Immediately from the definition of $ZG(\mathcal{A})$ we can deduce that $\mathcal{A}$ has an accepting run iff there is a node (q, Z) reachable in $ZG(\mathcal{A})$ such that $q \in Acc$.

Now, every node (q, Z) has finitely many successors: at most one successor of (q, Z) per transition in $\mathcal{A}$. Still a reachability algorithm may not terminate as the number of reachable nodes in $ZG(\mathcal{A})$ may not be finite [9]. The next step is thus to define an abstract semantics of $\mathcal{A}$ as a finite graph. The basic idea is to define a finite partition of the set of valuations $\mathbb{R}^X_{\geq 0}$. Then, instead of considering nodes (q, S) with set of valuations S (e.g. zones Z), one considers a union of the parts of $\mathbb{R}^X_{\geq 0}$ that include S, hence the finite abstraction.

Let $\alpha : X \to \mathbb{N}$ associate a *bound* to each clock of $\mathcal{A}$. A *region* [1] with respect to α is the set of valuations specified as follows:

1. for each clock $x \in X$, one constraint from the set:
   $\{x = c \mid c = 1, \ldots, \alpha(x)\} \cup \{c - 1 < x < c \mid c = 1, \ldots, \alpha(x)\} \cup \{x > \alpha(x)\}$
2. for each pair of clocks x, y having interval constraints: $c - 1 < x < c$ and $d - 1 < y < d$, it is specified if $fract(x)$ is less than, equal to or greater than $fract(y)$.

It can be checked that the set of regions is a finite partition of $\mathbb{R}^X_{\geq 0}$.

The *closure abstraction* of a set of valuations S, denoted $Closure_\alpha(S)$, is the set of regions that intersect S [6]. A simulation graph, denoted $SG_\alpha(\mathcal{A})$, has nodes of the form (q, S) where q is a state of $\mathcal{A}$ and $S \subseteq \mathbb{R}^X_{\geq 0}$ is a set of valuations. The initial node of $SG_\alpha(\mathcal{A})$ is $(q_0, \{\mathbf{0}\})$. There is an edge $(q, S) \xrightarrow{t} (q', Closure_\alpha(S'))$ in $SG_\alpha(\mathcal{A})$ iff S' is the set of valuations ν' such that $(q, \nu) \xrightarrow{\delta,t} (q', \nu')$ for some $\nu \in S$ and $\delta \in \mathbb{R}_{\geq 0}$. Notice that the reachable part of $SG_\alpha(\mathcal{A})$ is finite since the number of regions is finite.

The definition of the graph $SG_\alpha(\mathcal{A})$ is parametrized by a bound function α. It is well-known that if we take $\alpha_\mathcal{A}$ associating to each clock x the maximal integer c such that $x\#c$ appears in some guard of $\mathcal{A}$ then $SG_\alpha(\mathcal{A})$ preserves the reachability properties.

Theorem 1 ([6]). *$\mathcal{A}$ has an accepting run iff there is a reachable node (q, S) in $SG_\alpha(\mathcal{A})$ with $q \in Acc$ and $\alpha_\mathcal{A} \leq \alpha$.*

Hence, in order to solve the emptiness problem for $\mathcal{A}$, one can first compute $\alpha_\mathcal{A}$ and then run a reachability algorithm on $SG_{\alpha_\mathcal{A}}(\mathcal{A})$, the graph obtained by choosing $\alpha_\mathcal{A}$ as the maximal bound. This is not what is done in the standard algorithm because $Closure_\alpha(Z)$ may not be convex and no efficient representations of closures are known.

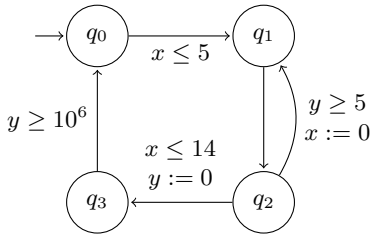

Fig. 1. Timed automaton $\mathcal{A}_4$.

The other issue related to efficiency concerns the bound function α. The nodes of $SG_\alpha(\mathcal{A})$ are unions of regions. Hence the size of $SG_\alpha(\mathcal{A})$ depends on the number of regions which is $\mathcal{O}\big(|X|!.2^{|X|}.\prod_{x\in X}(2.\alpha(x)+2)\big)$ [1]. It follows that smaller values for α yield a coarser, hence smaller, symbolic graph, whereas bigger bounds yield a more precise but bigger graph $SG_\alpha(\mathcal{A})$. In Section 3.2 we discuss this issue in detail.

2.3 Static Guard Analysis: An Example

It has been observed in [2] that instead of considering a global bound function $\alpha_\mathcal{A}$ for all states in $\mathcal{A}$, one can use different functions in each state of the automaton. Thus one bounds function α_q is associated to each state q in $\mathcal{A}$. This bound function is later used for the abstraction of the nodes in $SG(\mathcal{A})$ that have state q: nodes of the form $(q, \mathit{Closure}_{\alpha_q}(S))$.

Consider for instance the automaton $\mathcal{A}_4$ in Figure 1. Looking at the guards, we get that $\alpha_{\mathcal{A}_4}(x) = 14$ and $\alpha_{\mathcal{A}_4}(y) = 10^6$. Consider the valuations $\nu : x \mapsto 10, y \mapsto 6$ and $\nu' : x \mapsto 10, y \mapsto 10^7$. Notice that ν and ν' do not belong to the same region since $\nu(y) < \alpha_{\mathcal{A}_4}(y)$ whereas $\nu'(y) > \alpha_{\mathcal{A}_4}(y)$. However, starting from (q_2, ν) or (q_2, ν'), one can fire the same sequences of transitions. Indeed, in both cases, the transition from q_2 to q_1 is enabled as well as the transition back to q_2. The transition from q_2 to q_3 is also enabled. Furthermore, the difference between $\nu(y)$ and $\nu'(y)$ is no longer relevant once the transition to q_3 is taken as y is reset on that transition. Hence, in state q_2, it is relevant to distinguish the valuations such that y is less than 5 from those where y is at least 5, since the transition from q_2 to q_1 is either disabled or enabled. However, it is irrelevant to consider a maximal bound bigger than 5 for clock y.

The state-based bounds functions are computed as follows. Let m_x^q be a non-negative integer variable associated to each state $q \in Q$ and each clock $x \in X$. Then for each transition (q, g, R, q') in $\mathcal{A}$, one defines the following system of inequalities: (1) $m_x^q \geq c$ if $x\#c$ appears in g; and (2) $m_x^q \geq m_x^{q'}$ if $x \notin R$. The first inequation states that the bounds in q is first defined by the guards of the transitions out of q. The second inequation entails that the bound on a clock x propagates backward along the edges unless x is reset. Indeed if x is not reset on

the transition from q to q', then a bound on x is satisfied in q' only if it satisfied in q.

Let $\mathcal{S}_\mathcal{A}$ denote the conjunction of the inequations defined for all the transitions in $\mathcal{A}$. The value of the bound function α_q for clock x can be chosen as any solution for m_x^q in $\mathcal{S}_\mathcal{A}$. The minimal solution gives the coarsest abstraction.

For instance, the minimal solution for $\mathcal{A}_4$ in Figure 1 has $\alpha_{q_0}(y) = \alpha_{q_1}(y) = \alpha_{q_2}(y) = 5$ and $\alpha_{q_3}(y) = 10^6$ yielding a substantial reduction on the size of the abstract symbolic semantics. This reduction preserves reachability properties.

3 A New Algorithm for Reachability

Our objective is to compute a finite prefix of the reachability tree of the zone graph $ZG(\mathcal{A})$ that is sufficient to decide the state reachability property. Finiteness is ensured by not exploring (q, Z) if there exists a (q, Z') such that $Z \subseteq Closure_\alpha(Z')$, for a suitable α. In this section, we first describe a simple algorithm based on the closure and then we will address the issue of finding tighter bounds for the clock values.

3.1 The basic algorithm

Given a timed automaton $\mathcal{A}$ we first calculate the bound function $\alpha_\mathcal{A}$ as described in Theorem 1. Each node in the tree that we compute is of the form (q, Z), where q is a state of the automaton, and Z is an unapproximated zone. The root node is (q_0, Z_0), which is the initial node of $ZG(\mathcal{A})$. The algorithm performs a depth first search: at a node (q, Z), a transition $t = (q, g, r, q')$ not yet considered for exploration is picked and the successor (q', Z') is computed where $(q, Z) \xrightarrow{t} (q', Z')$ in $ZG(\mathcal{A})$. If q' is a final state and Z' is not empty then the algorithm terminates. Otherwise the search continues from (q', Z') unless there is already a node (q', Z'') with $Z' \subseteq Closure_{\alpha_\mathcal{A}}(Z'')$ in the current tree.

The correctness of the algorithm is straightforward. It follows from the fact that if $Z' \subseteq Closure_{\alpha_\mathcal{A}}(Z'')$ then all the states reachable from (q', Z') are reachable from (q', Z'') and hence it is not necessary to explore the tree from (q', Z'). Termination of the algorithm is ensured since there are finitely many sets of the form $Closure_{\alpha_\mathcal{A}}(Z)$. Indeed, the algorithm will construct a prefix of the reachability tree of $SG_\alpha(\mathcal{A})$ as described in Theorem 1.

The above simple algorithm does not use the classical extrapolation operator named $Extra^+_M$ in [3] and $Extra^+_\alpha$ hereafter, but the coarser $Closure_\alpha$ operator [6]. This is possible since the algorithm does not need to represent $Closure_\alpha(Z)$, which is in general not a zone. Instead of storing $Closure_\alpha(Z)$ the algorithm just stores Z and performs tests $Z' \subseteq Closure_\alpha(Z)$ each time it is needed (in contrast to Algorithm 2 in [6]). At first sight this may seem to be a source of big inefficiency. That this is not the case is the main point we wish to make here. In section 4 we show checking $Z' \subseteq Closure_\alpha(Z)$ can be done in time $\mathcal{O}(X^2)$, which is the complexity of the inclusion test $Z' \subseteq Z$ used in the standard algorithm.

```
1  function main():
2    push((q0, Z0, α0), stack)
3    while (stack ≠ ∅) do
4      (q, Z, α) := top(stack); pop(stack)
5      explore(q, Z, α)
6      resolve()
7    return "empty"
8
9  function explore(q, Z, α):
10   if (q is accepting)
11     exit "not empty"
12   if (∃ (q, Z', α') nottentative
13             and s.t. Z ⊆ Closure_α'(Z'))
14     mark (q, Z, α) tentative wrt (q, Z', α')
15     α := α'; propagate(parent(q, Z, α))
16   else
17     propagate(q, Z, α)
18     for each (q_s, Z_s, α_s) in children(q, Z, α) do
19       if (Z_s ≠ ∅)
20         explore(q_s, Z_s, α_s)
21
22 function resolve():
23   for each (q, Z, α) tentative wrt (q, Z', α') do
24     if (Z ⊈ Closure_α'(Z'))
25       mark (q, Z, α) nottentative
26       α := −∞; propagate(parent(q, Z, α))
27       push((q, Z, α), stack)
28
29 function propagate(q, Z, α):
30   α := max_{(q,Z,α) --g;R--> (q_s,Z_s,α_s)} (max_edge(g, R, α_s))
31   if (α has changed)
32     for each (q_t, Z_t, α_t) tentative wrt (q, Z, α) do
33       α_t := α; propagate(parent(q_t, Z_t, α_t))
34     if ((q, Z, α) ≠ (q0, Z0, α0))
35       propagate(parent(q, Z, α))
36
37 function max_edge(g, R, α):
38   let α_R = λx. if x ∈ R then −∞ else α(x)
39   let α_g = λx. if x#c in g then c else −∞
40   return (λx. max(α_R(x), α_g(x)))
```

Fig. 2. Reachability algorithm with on-the-fly bound computation and non-convex abstraction.

Since $Closure_\alpha$ is a coarser abstraction, this simple algorithm already covers some of the optimizations of the standard algorithm. For example the $Extra^+_\alpha(Z)$ abstraction proposed in [3] is subsumed since $Extra^+_\alpha(Z) \subseteq Closure_\alpha(Z)$ for any zone Z [6, 3]. Other important optimizations of the standard algorithm concern finer computation of bounding functions α. We now show that the structure of the proposed algorithm allows to improve this too.

3.2 Tightening the bounds

Instead of considering a global bound function $\alpha_{\mathcal{A}}$, it has been observed in [2] that associating a bound function α_q to each state q of an automaton is more efficient. Consequently the bound α_q is used for the abstraction at nodes (q, Z) having state q. An example is presented in in the extended version of this paper 2.3.

In [2] state-based bound functions α_q are obtained by a static analysis of the automaton. In this section we show how to modify our basic algorithm to compute bound functions on-the-fly for each node (q, Z) in the reachability tree that is constructed. An obvious gain is that we will never consider constraints coming from unreachable transitions. We comment more on advantages of this approach in Section 5.

Our modified algorithm is given in Figure 2. It computes a tree whose nodes are triples (q, Z, α) where (q, Z) is a node of $ZG(\mathcal{A})$ and α is a bound function, which by default maps each clock to $-\infty$. The root node is (q_0, Z_0, α_0) where (q_0, Z_0) is the initial node of $ZG(\mathcal{A})$. Each node (q, Z, α) has as many child nodes (q_s, Z_s, α_s) as there are successors (q_s, Z_s) of (q, Z) in $ZG(\mathcal{A})$. Notice that this includes successors with an empty zone Z_s, which are however not further unfolded. These nodes must be included as explained below. Each node is further

marked either *tentative* or *nottentative*. The leaf nodes (q, Z, α) of the tree are either deadlock nodes (either there is no transition out of state q or Z is empty), or *tentative* nodes. All the other nodes are marked *nottentative*.

Our algorithm repeatedly alternates an exploration and a resolution phase as described below.

Exploration phase. Our algorithm computes the tree described above starting from the root node. Before exploring a node $n = (q, Z, \alpha)$ the function `explore` checks if q is accepting and Z is not empty; if it is so then $\mathcal{A}$ has an accepting run. Otherwise the algorithm checks if there exists a *nottentative* node $n' = (q', Z', \alpha')$ in the current tree such that $q = q'$ and $Z \subseteq \mathit{Closure}_{\alpha'}(Z')$. If yes, n becomes a *tentative* node and its exploration is temporarily stopped as each state reachable from n is also reachable from n'. If none of this holds, the successors of the node are explored. The exploration terminates since $\mathit{Closure}_\alpha$ has a finite range.

When the exploration algorithm gets to a new node, it propagates the bounds from this node to all its predecessors. The goal is to compute a suitable bounding function α in each node. We now describe how this is achieved. The bounding function in a node (q, Z, α) should be such that for every clock x and every successor (q_s, Z_s, α_s) the value $\alpha(x)$ is bigger than the constants x is compared to in the guard of the transition leading to the successor; and it should also be greater than $\alpha_s(x)$ if x is not reset by the transition (as in function `max_edge`). In consequence, each time the value of α_s changes, the value of α should be verified and eventually changed (as in function `propagate`). This is analogous to the inequations seen in the static guard analysis [2], however now applied to the zone graph, on-the-fly. Hence, the bounds associated to each node (q, Z, α) never exceed those that are computed by the static guard analysis.

This argument cannot be applied on tentative nodes as their successor nodes are not computed. However, if $n = (q, Z, \alpha)$ is tentative with respect to $n' = (q, Z', \alpha')$, then every state reachable from n is also reachable from n'. As a consequence, we know that α cannot be bigger than α'. Hence, when n is marked *tentative*, we assign the bound function α' to n and propagate α' upward.

The goal of these propagations is to maintain the following invariants. For every node $n = (q, Z, \alpha)$:

1. if n is *nottentative*, then α is the maximum of the α_s for all successor nodes (q_s, Z_s, α_s) of n (taking into account the clock resets as explained above);
2. if n is *tentative* with respect to (q', Z', α'), then α is equal to α'.

A delicate point about this procedure is handling of tentative nodes. When a node n is marked *tentative*, we have $\alpha = \alpha'$. However the value of α' may be updated when the tree is further explored. Thus each time we update the bounds function of a node, it is not only propagated upward in the tree but also to the nodes that are tentative with respect to n'.

This algorithm terminates as the bound functions in each node never decrease and are bounded. From the invariants above, we get that in every node, α is a solution to the equations in [2] applied on $ZG(\mathcal{A})$.

It could seem that our algorithm is inefficient due to the high number of propagations of bounds that it involves. The experiments, as reported in Section 5, show that the present very simple approach to bound propagation is good enough. Since we propagate the bounds as soon as they are modified, most of the time, the value of α does not change in line 30 of function `propagate`. In general, bounds are only propagated on very short distances in the tree, mostly along one single edge. For this reason we do not concentrate on optimizing the function `propagate`. In the implementation we use the presented function augmented with a minor "optimization" that avoids calculating maximum over all successors in line 30 when it is not needed.

Resolution phase. Finally, as the bounds may have changed since n has been marked tentative the function `resolve` checks for the consistency of *tentative* nodes. If $Z \subseteq Closure_{\alpha'}(Z')$ is not true anymore, n needs to be explored. Hence it is viewed as a new node: the bounds are set to $-\infty$ and n is pushed on the *stack* for further consideration in the function `main`. Setting α to $-\infty$ is safe as α will be computed and propagated when n is explored. We perform also a small optimization and propagate this bound upward, thereby making some bounds decrease.

The resolution phase may provide new nodes to be explored. The algorithm terminates when it is not the case, that is when all tentative nodes remain tentative. We can then conclude that no accepting state is reachable.

3.3 Correctness of the algorithm

Theorem 2. *An accepting state is reachable in $ZG(\mathcal{A})$ iff the algorithm reaches a node with an accepting state and a non-empty zone.*

The right-to-left direction follows by a straightforward induction on the length of the path. The left-to-right direction is shown using the following lemmas.

Let $Post(S, t)$ stand for the set of all valuations of clocks reachable by t from valuations in S. We will need the following observation.

Lemma 1 ([6]). *For every zone Z, transition t and a bound function α:*

$$Post(Closure_\alpha(Z), t) \subseteq Closure_\alpha(Post(Z, t)).$$

Lemma 2. *Suppose that algorithm concludes that the final state is not reachable. Consider the tree it has constructed. For every (q, Z) reachable from (q_0, Z_0) in $ZG(\mathcal{A})$, there is a non tentative node (q, Z', α') in the tree, such that $Z \subseteq Closure_{\alpha'}(Z')$.*

Proof. The hypothesis is vacuously true for (q_0, Z_0). Assume that the hypothesis is true for a node $(q, Z) \in ZG(\mathcal{A})$. We now prove that the lemma is true for every successor of (q, Z).

From hypothesis, there exists a non tentative node (q, Z_L, α) in the constructed tree such that $Z \subseteq Closure_\alpha(Z_L)$. Let $t = (q, g, r, q')$ be a transition of $\mathcal{A}$ and let $(q, Z) \xrightarrow{t} (q', Z') \in ZG(\mathcal{A})$.

The transition t is enabled from (q, Z_L, α) because $Z \subseteq Closure_\alpha(Z_L)$, and, due to constraint propagation, for every clock x, $\alpha(x)$ is greater than the maximum constant it is compared to in the guard g. So we have

$$(q, Z_L, \alpha) \xrightarrow{t} (q', Z'_L, \alpha')$$

in the constructed tree.

Since $Z \subseteq Closure_\alpha(Z_L)$, we have $Post(Z, t) \subseteq Post(Closure_\alpha(Z_L), t)$, that is $Z' \subseteq Post(Closure_\alpha(Z_L), t)$. From Lemma 1, $Z' \subseteq Closure_\alpha(Post(Z_L, t))$, that is $Z' \subseteq Closure_\alpha(Z'_L)$. We now need to check if we can replace α with α'. But $Closure_\alpha(Z'_L) \subseteq Closure_{\alpha'}(Z'_L)$ since by definition of constant propagation $\alpha(x) \geq \alpha'(x)$ for all clocks x not reset by t, and for clocks x that are reset, Z'_L entails $x = 0$, therefore irrespective of α or α' the regions that intersect with Z'_L should satisfy $x = 0$.

If $n' = (q', Z'_L, \alpha')$ is non tentative, we are done and n' is the node in the constructed tree corresponding to (q', Z'). If n' is tentative, by definition, we know that there exists a non tentative node (q', Z''_L, α'') such that $\alpha'' = \alpha'$ and $Z'_L \subseteq Closure_{\alpha'}(Z''_L)$. Thus $Z' \subseteq Closure_{\alpha'}(Z''_L)$. In this case (q', Z''_L, α'') is the node corresponding to (q', Z').

□

3.4 Handling LU approximations.

In [3] the authors proposed to distinguish between maximal constants used in upper and lower bounds comparisons. They have introduced an extrapolation operator $Extra^+_{LU}(Z)$ that takes into account this information. This is probably the best presently known convex abstraction of zones.

We will present $Extra^+_{LU}(Z)$ in Section 4.3. Here let us just say that in our algorithm we can easily propagate LU bounds instead of just maximal bounds. We can also replace the test $Z \subseteq Closure_{\alpha'}(Z')$ by $Z \subseteq Closure_{\alpha'}(Extra^+_{L'U'}(Z'))$, where $L'U'$ are the bounds calculated for Z' and $\alpha'(x) = \max(L'(x), U'(x))$ for every clock x. We show in Section 4.3 that this test can be done efficiently too. The proof of the correctness of the resulting algorithm follows the same lines as above.

Correctness of the algorithm with LU-approximation The proof of the correctness of the algorithm using $Z \subseteq Closure^+_{LU}(Z')$ test is similar to that using $Z \subseteq Closure_\alpha(Z')$ test. We call it LU-algorithm for short. Since $Extra^+_{LU}$ is difficult to handle, we do a small detour through another approximation $\mathfrak{a}_{\preccurlyeq LU}(Z)$ introduced in [3]. We recall its definition here.

Definition 1 (LU-preorder). *Fix integers L and U. Let ν and ν' be two valuations. Then, we say $\nu' \preccurlyeq_{LU} \nu$ if for each clock x:*

- *either* $\nu'(x) = \nu(x)$,
- *or* $L(x) < \nu'(x) < \nu(x)$,
- *or* $U(x) < \nu(x) < \nu'(x)$.

This LU-preorder can be extended to define abstractions of sets of valuations.

Definition 2 ($\mathfrak{a}_{\preccurlyeq LU}$, abstraction w.r.t $\preccurlyeq_{LU}$). *Let W be a set of valuations. Then,*

$$\mathfrak{a}_{\preccurlyeq LU}(W) = \{\nu \mid \exists \nu' \in W,\ \nu' \preccurlyeq_{LU} \nu\}$$

It is shown in [3] that this is a sound, complete and finite abstraction, coarser than *Closure*. The soundness of this abstraction follows from the lemma given below.

Lemma 3. *Let ν and ν' be valuations such that $\nu' \preccurlyeq_{LU} \nu$. Let q be a state of $\mathcal{A}$ and t a transition. Then, $(q,\nu) \xrightarrow{\delta,t} (q_1,\nu_1)$ implies that there exists a delay δ' and a valuation ν'_1 such that $(q,\nu') \xrightarrow{\delta',t} (q_1,\nu'_1)$ and $\nu'_1 \preccurlyeq_{LU} \nu_1$.*

The relation between $\mathfrak{a}_{\preccurlyeq LU}(Z)$ and $Extra^+_{LU}(Z)$ is summarized by the following.

Lemma 4. *For all zones Z,*

$$Extra^+_{LU}(Z) \text{ is a zone} \tag{1}$$

$$Extra^+_{LU}(Z) \subseteq \mathfrak{a}_{\preccurlyeq LU}(Z) \tag{2}$$

We are now in a position to prove the correctness of LU-algorithm.

Theorem 3. *An accepting state is reachable in $ZG(\mathcal{A})$ iff the LU-algorithm reaches a node with an accepting state and a non-empty zone.*

The right to left direction is straightforward, so we concentrate on the opposite direction.

Lemma 5. *For every zone Z, and transition t:*

$$Post(\mathfrak{a}_{\preccurlyeq LU}(Z),t) \subseteq \mathfrak{a}_{\preccurlyeq LU}(Post(Z,t))$$

Proof. Pick $\nu_1 \in Post(\mathfrak{a}_{\preccurlyeq LU}(Z),t)$. There exists a valuation $\nu \in \mathfrak{a}_{\preccurlyeq LU}(Z)$ such that $\nu \xrightarrow{t} \nu_1$. By definition of $\mathfrak{a}_{\preccurlyeq LU}$, there exists a valuation $\nu' \in Z$ such that $\nu' \preccurlyeq_{LU} \nu$. From Lemma 3, $\nu' \xrightarrow{t} \nu'_1$ such that $\nu'_1 \preccurlyeq_{LU} \nu_1$. Hence $\nu_1 \in \mathfrak{a}_{\preccurlyeq LU}(Post(Z,t))$. □

The left to right implication of the theorem follows form the next lemma.

Lemma 6. *For every (q,Z) reachable in $ZG(\mathcal{A})$, there exists a non tentative node (q,Z_1,L_1U_1) in the tree constructed by the LU-algorithm, such that $Z \subseteq \mathfrak{a}_{\preccurlyeq_{L_1U_1}}(Z_1)$.*

Proof. The hypothesis is vacuously true for (q_0, Z_0). Assume that the hypothesis is true for a node $(q, Z) \in ZG(\mathcal{A})$. We prove that the lemma is true for every successor of (q, Z).

From hypothesis, there exists a node (q, Z_1, L_1, U_1) in the tree constructed by the LU-algorithm such that $Z \subseteq \mathfrak{a}_{\preccurlyeq_{L_1U_1}}(Z_1)$. Let $t = (q, g, r, q')$ be a transition of $\mathcal{A}$ and let $(q, Z) \xrightarrow{t} (q', Z') \in ZG(\mathcal{A})$. There are two cases.

(q, Z_1) *is not tentative* Since $Z \subseteq \mathfrak{a}_{\preccurlyeq_{L_1U_1}}(Z_1)$, the transition t is enabled from $\mathfrak{a}_{\preccurlyeq_{L_1U_1}}(Z_1)$. From Lemma 3, t is enabled from Z_1 too. Since $Z \subseteq \mathfrak{a}_{\preccurlyeq_{L_1U_1}}(Z_1)$, we have $Post(Z, t) \subseteq Post(\mathfrak{a}_{\preccurlyeq_{L_1U_1}}(Z_1), t)$, that is $Z' \subseteq Post(\mathfrak{a}_{\preccurlyeq_{L_1U_1}}(Z_1), t)$. From Lemma 5, $Z' \subseteq \mathfrak{a}_{\preccurlyeq_{L_1U_1}}(Post(Z_1, t))$. We can take Z_1' as $Post(Z_1, t)$ and then let $(q', Z_1', L_1'U_1')$ be the successor node in the tree computed by LU-algorithm. It remains to show that $Z' \subseteq \mathfrak{a}_{\preccurlyeq_{L_1'U_1'}}(Z_1')$ is the node corresponding to (q', Z'). This follows because by definition $L_1(x) \geq L_1'(x)$, $U_1(x) \geq U_1'(x)$ for all clocks x that are not reset by the transition t and for the clocks reset by t, Z_1' entails $x = 0$.

(q, Z_1) *is tentative* If it is a tentative node, we know that there exists a non-tentative node (q, Z_2, L_2U_2) in the tree constructed by the LU-algorithm such that $Z_1 \subseteq Closure_{L_2U_2}(Z_2)$, that is, $Z_1 \subseteq \mathfrak{a}_{L_2U_2}(Z_2)$. The rest of the argument is the same as in the previous case with (q, Z_2, L_2U_2) instead of (q, Z_1, L_1U_1).

□

4 An $\mathcal{O}(|X|^2)$ Algorithm for Inclusion Testing

The tests of the form $Z \subseteq Closure_\alpha(Z')$ are at the core of the new algorithm we have presented. This is an important difference with respect to the standard algorithm that makes the tests of the form $Z \subseteq Z'$. The latter tests are done in $\mathcal{O}(|X|^2)$ time, where $|X|$ is the number of clocks. We present in this section a simple algorithm that can do the tests $Z \subseteq Closure_\alpha(Z')$ at the same complexity.

We will start by examining the question how to decide if a region R intersects a zone Z. The important point is that it is enough to verify that projection on every pair of variables is nonempty. This fact has been proven in [6] but as an intermediate step of some other argument and it is not even stated there explicitly as a lemma. We give a different proof of this fact, that allows us to derive the efficient inclusion testing algorithm also for LU-approximation.

4.1 When is $R \cap Z$ empty

It will be very convenient to represent zones by *distance graphs*. Such a graph has clocks as vertices, with additional special clock x_0 representing constant 0. For readability, we will often write 0 instead of x_0. Between every two vertices there is an edge with a weight of the form $(\preccurlyeq, c)$ where $c \in \mathbb{Z} \cup \{\infty\}$ and $\preccurlyeq$ is either $\leq$ or $<$. An edge $x \xrightarrow{\preccurlyeq c} y$ represents a constraint $y - x \preccurlyeq c$: or in words,

that the distance from x to y is bounded by c. Let $[\![G]\!]$ be the set of valuations of clock variables satisfying all the constraints given by the edges of G with the restriction that the value of x_0 is 0.

An arithmetic over the weights $(\preccurlyeq, c)$ could be defined as follows.

Equality $(\preccurlyeq_1, c_1) = (\preccurlyeq_2, c_2)$ if $c_1 = c_2$ and $\preccurlyeq_1 = \preccurlyeq_2$.
Addition $(\preccurlyeq_1, c_1) + (\preccurlyeq_2, c_2) = (\preccurlyeq, c_1 + c_2)$ where $\preccurlyeq = <$ iff either $\preccurlyeq_1$ or $\preccurlyeq_2$ is $<$.
Order $(\preccurlyeq_1, c_1) < (\preccurlyeq_2, c_2)$ if either $c_1 < c_2$ or if $c_1 = c_2$ then $\preccurlyeq_1 = <$ and $\preccurlyeq_2 = \leq$.
Similarly for $\leq$.

This arithmetic allows us to talk about the weight of a path as a weight of the sum of its edges. A cycle in a distance graph G is said to be *negative* if the sum of the weights of its edges is at most $(<, 0)$; otherwise the cycle is *positive*. The following proposition is a folklore.

Proposition 1. *A distance graph G has only positive cycles iff $[\![G]\!] \neq \emptyset$.*

Proof. The proposition states that a distance graph G has only positive cycles iff $[\![G]\!] \neq \emptyset$.

If there is a valuation $v \in [\![G]\!]$ then we replace every edge $x \xrightarrow{\preccurlyeq c} y$ by $x \xrightarrow{\leq d} y$ where $d = v(y) - v(x)$. We have $d \preccurlyeq_{xy} c_{xy}$. Since every cycle in the new graph has value 0, every cycle in G is positive.

For the other direction suppose that every cycle in G is positive. Let $\overline{G}$ be the canonical form of G. Clearly $[\![G]\!] = [\![\overline{G}]\!]$, i.e., the constraints defined by G and by $\overline{G}$ are equivalent. It is also evident that all the cycles in $\overline{G}$ are positive. We exhibit a valuation in $[\![\overline{G}]\!]$. Choose a variable x and a value $\lambda_x \in \mathbb{R}$ such that

$$\lambda_x - 0 \preccurlyeq_{0x} c_{0x} \quad \text{and} \quad 0 - \lambda_x \preccurlyeq_{x0} c_{x0}$$

Such a value exists since the cycle $x_0 \to x \to x_0$ is positive. Let $\overline{G}_x$ be $\overline{G}$ where we put the edges $0 \xrightarrow{\leq \lambda_x} x$ and $x \xrightarrow{\leq -\lambda_x} 0$ instead of the ones present in $\overline{G}$. We have that all valuations $[\![\overline{G}_x]\!]$ have the value of x set to λ_x. We will below show that all cycles in $\overline{G}_x$ are positive. Assuming this we examine the graph G_{-x} obtained from $\overline{G}_x$ by removing the vertex x. Since all its cycles are positive, there is a valuation in $[\![G_{-x}]\!]$. If we additionally set the value of x to be λ_x, we obtain a valuation from $[\![\overline{G}_x]\!]$ and we are done.

It remains to show that $[\![\overline{G}_x]\!]$ does not have a negative cycle. For contradiction suppose that there is a negative cycle in $\overline{G}_x$. Clearly, 0 should be on this cycle: $0 \longrightarrow x_1 \ldots \longrightarrow x_n \longrightarrow 0$. Now since there where no negative cycle in $\overline{G}$, this cycle must use an edge that we have added. So either x_1 or x_n is x. In the later case the path from 0 to x can be reduced to an edge $0 \longrightarrow x$. But the value of the cycle $0 \longrightarrow x \longrightarrow 0$ in $\overline{G}_x$ is 0. So the only remaining possibility is that $x_1 = x$. Then the path from x_2 to 0 can be reduced to one edge. So we are only interested in cycles of the form $0 \longrightarrow x \longrightarrow y \longrightarrow 0$. The value of such a cycle is $d + c_{xy} + c_{y0}$. From canonicity of $\overline{G}$, we know that $c_{xy} + c_{y0} \geq c_{x0}$, that is, $-c_{x0} + c_{xy} + c_{y0} \geq 0$. But since $d \geq -c_{x0}$, the value of the cycle is positive. □

Let R be a region wrt. a bound function $\alpha : X \to \mathbb{N}$. A variable x is *bounded in* R if a constraint $x \leq c$ holds in R for some constant c; otherwise the variable is called *unbounded in* R. Observe that if x_1, x_2 are bounded then we have

$$x_1 - x_2 = c \quad \text{or} \quad c - 1 < x_1 - x_2 < c \quad \text{in } R.$$

If y is unbounded then we have $y > \alpha(y)$ in R.

A distance graph is in *canonical form* if the weight of the edge from x to y is the lower bound of the weights of paths from x to y. A *distance graph of a region* R, denoted G_R, is the canonical graph representing all the constraints defining R. Similarly G_Z for a zone Z.

Before proceeding we need one more piece of notation. For two distance graphs G_1, G_2, we denote by $\min(G_1, G_2)$ the distance graph where each edge has the weight equal to the minimum of the corresponding weights in G_1 and G_2. It should be clear that $[\![\min(G_1, G_2)]\!] = [\![G_1]\!] \cap [\![G_2]\!]$, that is the valuations satisfying the constraints given by $\min(G_1, G_2)$ are exactly those satisfying all the constraints from G_1 as well as from G_2. This holds even if the two graphs are not in the canonical form.

We are now ready to examine the conditions when $R \cap Z$ is empty. We start with the following simple lemma.

Lemma 7. *Let G_R be the distance graph of a region and let x_1, x_2 be two variables bounded in R. For every distance graph G: if in $\min(G_R, G)$ the weight of the edge $x_1 \longrightarrow x_2$ comes from G then $x_1 \longrightarrow x_2 \longrightarrow x_1$ is a negative cycle in $\min(G_R, G)$.*

Proof. Suppose that the edge $x_1 \xrightarrow{\preccurlyeq c} x_2$ is as required by the assumption of the lemma. In R we can have either $x_2 - x_1 = d$ or $d - 1 < x_2 - x_1 < d$.

In the first case we have an edge $x_1 \xrightarrow{\leq d} x_2$ in G_R. Since the corresponding edge in G is smaller we have $c < d$ or $c = d$ and $\preccurlyeq$ is the strict inequality. Put together with the edge $x_2 \xrightarrow{\leq -d} x_1$ present in R we get a negative cycle.

In the second case we have $c < d$, and an edge $x_2 \xrightarrow{< -d+1} x_1$ in R. Put together with $x_1 \xrightarrow{\preccurlyeq c} x_2$ this gives a negative cycle. □

Proposition 2. *Let R be a region and let Z be a zone. The intersection $R \cap Z$ is empty iff $\min(G_R, G_z)$ has a negative cycle involving at most two clocks and possibly x_0.*

Proof. If $\min(G_R, G_Z)$ has a negative cycle then $R \cap Z$ is empty by Proposition 1.

For the other direction suppose that $R \cap Z$ is empty. By Proposition 1 the graph $\min(G_R, G_Z)$ has a negative cycle. An immediate case is when in this graph an edge between two variables bound in R comes from G_Z. From Lemma 7 we obtain a negative cycle on these two variables. So in what follows we suppose that in $\min(G_R, G_Z)$ all the edges between variables bounded in R come from G_R. Hence every negative cycle should contain an unbounded variable.

Let y be a variable unbounded in R that is a part of the negative cycle. Consider y with its successor and its predecessor on the cycle: $x \longrightarrow y \longrightarrow x'$.

Observe that in R every edge to y has value ∞. So the weight of the edge $x \longrightarrow y$ is from Z. If the weight of the outgoing edge is also from Z then we could have eliminated y from the cycle by choosing $x \longrightarrow x'$ from Z. Hence the weight of $y \longrightarrow x'$ comes from R. Since y is unbounded in R, the weight of this edge is $d - \alpha(y)$, where d is the value on the edge $0 \stackrel{\preccurlyeq d}{\longrightarrow} x'$ in R. This is because we can rewrite inequation $x' - y < d - \alpha(y)$ as $y - x' > \alpha(y) - d$, and we know that $\alpha(y)$ is the smallest possible value for y, while d is the supremum on the possible values of x. But then instead of the edge $y \longrightarrow x'$ we can take $y \longrightarrow x_0 \longrightarrow x'$ in $\min(G_R, G_Z)$ which has smaller value since we have $y \stackrel{<-\alpha(y)}{\longrightarrow} x_0$ in G_R.

If x is x_0 then we get a cycle of a required form since it contains only x_0 and y. Otherwise, let us more closely examine the whole negative cycle:

$$x_0 \longrightarrow x_{i_1} \longrightarrow \ldots \longrightarrow x_{i_k} \longrightarrow x \longrightarrow y \longrightarrow x_0 \ .$$

By the reasoning from the previous paragraph, all of $x_{i_1}, \ldots, x_{i_k}$ can be assumed to be bounded in R. Otherwise we could get a cycle visiting x_0 twice and we could remove a part of it with one unbounded variable and still have a negative cycle. By our assumption, all the edges from and to these variables come from R. This means that the path from x_0 to x can be replaced by an edge $x_0 \longrightarrow x$ from R. So finally, the negative cycle has the required form $x_0 \longrightarrow x \longrightarrow y \longrightarrow x_0$. □

Let us write R_{xy} for the weight of the edge $x \longrightarrow y$ in G_R. Similarly Z_{xy} will stand for the weight of this edge in G_Z. The following corollary lists all the cases we have encountered in the proof of the last proposition.

Corollary 1. *Let R be a region and let Z be a zone. The intersection $R \cap Z$ is empty iff one of the following conditions holds (below x, x' are variables bound in R and y a variable unbound in R):*

1. $Z_{0x} < R_{0x}$, *or*
2. $Z_{x0} < R_{x0}$, *or*
3. $Z_{xx'} < R_{xx'}$, *or*
4. $Z_{0y} \leq (\leq, \alpha(y))$, *or*
5. $Z_{xy} \leq (\leq, \alpha(y) - c)$ *where c is the constant of the edge R_{0x}.*

4.2 Efficient inclusion testing

Our goal is to efficiently perform the test $Z \subseteq \mathit{Closure}(Z')$ for two zones Z and Z'. We are aiming at quadratic complexity, since this is the complexity of the currently used algorithms for checking the inclusion of two zones. Corollary 1 shows that the inclusion $R \subseteq \mathit{Closure}(Z')$ can be tested efficiently. It remains to understand what are the regions intersecting the zone Z and then to consider all possible cases. The first lemma allows us to deduce what regions intersect Z.

Lemma 8. *Let G be a distance graph in canonical form, with all cycles positive. Let x, y be two variables such that $x \stackrel{\preccurlyeq_{xy} c_{xy}}{\longrightarrow} y$ and $y \stackrel{\preccurlyeq_{yx} c_{yx}}{\longrightarrow} x$ are edges in G. Let $d \in \mathbb{R}$ such that $d \preccurlyeq_{xy} c_{xy}$ and $-d \preccurlyeq_{yx} c_{yx}$. Then, there exists a valuation $v \in [\![G]\!]$ such that $v(y) - v(x) = d$.*

Proof. Let G_d be the distance graph where we have the edges $x \xrightarrow{\leq d} y$ and $y \xrightarrow{\leq d}$ for variables x and y and the rest of the edges come from G. We show that all cycles in G_d are positive. For contradiction, suppose there is a negative cycle N in G_d. Clearly, since G does not have negative cycle, N should contain the variables x and y. The value of the shortest path from x to y in G was given by $(\preccurlyeq_{xy}, c_{xy})$. Therefore, the shortest path value from x to y in G_d is given by d and the shortest path value from y to x is $-d$. Hence the sum of the weights in N is negative would imply that the value of the cycle $x \to y \to x$ is negative. However since, this is 0, such a negative cycle N cannot exist. The lemma follows from Proposition 1. □

Let Z, Z' be zones and let $G_Z, G_{Z'}$ be the respective distance graphs in canonical form. The above lemma basically says that every consistent instantiation of an edge in G_Z leads to a region intersecting Z. Hence it is enough to look at edges of G_Z one by one to see what regions we can get. This analysis is presented in Lemmas 9, 10 and 11 below. As the result we get:

Proposition 3. *Let Z, Z' be zones. Then, $Z \not\subseteq Closure_\alpha(Z')$ iff there exist variables x, y such that one of the following conditions hold:*

1. $Z'_{0x} < Z_{0x} \ \wedge \ Z'_{0x} \leq (\alpha(x), \leq)$ *or*
2. $Z'_{x0} < Z_{x0} \ \wedge \ Z_{x0} \geq (-\alpha(x), \leq)$ *or*
3. $Z_{x0} \geq (-\alpha(x), \leq) \ \wedge \ Z'_{xy} < Z_{xy} \ \wedge \ Z'_{xy} \leq (\alpha(y), \leq) + Z_{x0}$

When the number of clocks is at least two, the overall algorithm thus involves a check as given by Proposition 3 for every pair of clocks. When there is a single clock, only the first two conditions need to be checked. Therefore, the complexity of our algorithm is $\mathcal{O}(|X|^2)$.

Comparison with the algorithm for $Z \subseteq Z'$. Given two zones Z and Z', the procedure for checking $Z \subseteq Z'$ first constructs G_Z and $G_{Z'}$ using a variant of Floyd's algorithm for finding all pair shortest paths. The Floyd's algorithm runs in $\mathcal{O}(|X|^3)$. However, all the zones Z that we compute are the successors of zones Z' that are in canonical form. Due to the specific edges that are changed during the successor computation in our case, this can be done in $\mathcal{O}(|X|^2)$ [12]. Once G_Z and $G_{Z'}$ are computed, checking for the inclusion is done by comparing the graphs edge by edge. This costs $\mathcal{O}(|X|^2)$ once again. So the overall complexity for direct inclusion check is $\mathcal{O}(|X|^2)$.

Note that our algorithm for $Z \subseteq Closure_\alpha(Z')$ does not do worse. It requires to compute G_Z and $G_{Z'}$ too. The edge by edge checks are only marginally more complicated. So the overall procedure is still $\mathcal{O}(|X|^2)$.

Correctness of Proposition 3 Correctness follows from the following lemmas.

Lemma 9. *Let Z, Z' be zones. Then, there exists a region R that intersects Z and satisfies condition 1,2 or condition 4 of Corollary 1 with respect to Z' iff there exists a variable x such that one of the following is true:*

$$Z'_{0x} < Z_{0x} \ \wedge \ Z'_{0x} \leq (\alpha(x), \leq) \tag{3}$$

$$Z'_{x0} < Z_{x0} \ \wedge \ Z_{x0} \geq (-\alpha(x), \leq) \tag{4}$$

Proof. If there exists a region that satisfies one of the conditions 1,2 or 4, then it can be easily verified that one of the above equations is true. We give a proof of the other direction.

Suppose Equation 3 above is true; that is, there exists a variable x such that $Z'_{0x} < Z_{0x}$ and $Z'_{0x} \leq (\alpha(x), \leq)$. There exists a $d \in \mathbb{R}$ such that $Z'_{0x} < d$ and d is consistent with the constraints of Z. If $d \leq \alpha(x)$ then clearly there exists a region R intersecting Z and that satisfies the condition $Z'_{0x} < R_{0x}$ of corollary 1. Otherwise, if $d > \alpha(x)$, then there exists a region R intersecting Z where x is an unbounded variable and condition $Z'_{0x} \leq (\alpha(x), \leq)$ is satisfied.

Suppose Equation 4 is true; that is, there exists a variable x such that $Z'_{x0} < Z_{x0}$ and $Z_{x0} \geq (-\alpha(x), \leq)$. Therefore, there exists a value $d \in \mathbb{R}$ such that $d < \alpha(x)$ and satisfies the conditions of Z. This gives a region R intersecting Z where x is bounded and condition 2 of the corollary is not satisfied.

□

Lemma 10. *Let Z, Z' be zones. If there exists a region R that intersects Z and satisfies condition 3 or condition 5 of Corollary 1 with respect to Z' then there exist variables x, y such that*

$$Z_{x0} \geq (-\alpha(x), \leq) \ \wedge \ Z'_{xy} < Z_{xy} \ \wedge \ Z'_{xy} \leq (\alpha(y), \leq) + Z_{x0} \tag{5}$$

Proof. If there exists a region R that satisfies condition 3, then x and y are bounded in R and so $Z_{x0} \geq (-\alpha(x), \leq)$. We also have $Z'_{xy} < R_{xy} \leq Z_{xy}$, and so the second conjunct of Equation 5 is also true. Since y is bounded, we have $R_{0y} \leq (\alpha(y), \leq)$. Therefore:

$$Z'_{xy} < R_{xy} \leq R_{x0} + R_{0y} \leq Z_{x0} + (\alpha(y), \leq)$$

This shows that the third conjunct in Equation 5 is also true.

Now consider the case when R satisfies condition 5: that is $Z'_{xy} \leq (\alpha(y) - c, \leq)$ where c is the constant appearing in R_{0x}. Directly since x is bounded, we get $Z_{x0} \geq (-\alpha(x), \leq)$. Also since $(-c, \leq) \leq Z_{x0}$, we get the third conjunct: $Z'_{xy} \leq (\alpha(y), \leq) + Z_{x0}$. For the second conjunct, note that there exists a valuation $v \in Z$ such that $(v(x), \leq) R_{0x}$, $v(y) > (\alpha(y))$ and $v(y) - v(x) \leq Z_{xy}$. Therefore:

$$Z'_{xy} \leq (\alpha(y) - c, \leq) < (v(y) - c, \leq) \leq (v(y) - v(x), \leq) \leq Z_{xy}$$

This shows that the second conjunct in Equation 5 is also true.

□

Lemma 11. *Let Z, Z' be zones. There exists a region R that intersects Z and satisfies condition 3 or condition 5 of Corollary 1 with respect to Z' if there exist variables x, y such that*

$$Z_{x0} \geq (-\alpha(x), \leq) \ \wedge \ Z'_{xy} < Z_{xy} \ \wedge \ Z'_{xy} \leq (\alpha(y), \leq) + Z_{x0} \tag{6}$$

Proof. Let the edges in G_Z be of the form $x \xrightarrow{\preccurlyeq_{xy} c_{xy}} y$ and the ones in $G_{Z'}$ be of the form $x \xrightarrow{\preccurlyeq'_{xy} c'_{xy}} y$. We assume $\preccurlyeq_{x0} = \leq$ and $c_{x0} = -c$. The other case when the edge is $x \xrightarrow{<-c} 0$ is left as an exercise.

Suppose Equation 5 is satisfied by Z and Z'. We claim that there exists a valuation $v \in Z$ such that

$$\begin{aligned} v(x) &= c \\ v(y) - v(x) &> c'_{xy}, \text{ if } \preccurlyeq'_{xy} = \leq \\ v(y) - v(x) &\geq c'_{xy}, \text{ if } \preccurlyeq'_{xy} = < \end{aligned}$$

Now, if $v(y) \leq \alpha(y)$, then the region R to which v belongs to has both x and y bounded and satisfies $(\preccurlyeq'_{xy}, c'_{xy}) < R_{xy}$. Hence condition 3 of Corollary 1, that is $Z'_{xy} < R_{xy}$ is satisfied. On the other hand, if $v(y) > \alpha(y)$, then the region R to which v belongs to has x bounded and y unbounded, and clearly intersects Z. From Equation 5, we have $Z'_{xy} < (\alpha(y), \leq) + Z_{x0}$. Therefore, $Z'_{xy} < (\alpha(y), \leq) + (\leq, -c)$ and hence $Z'_{xy} < (\alpha(y) - c, \leq)$. So, condition 5 of Corollary 1 is satisfied by R and Z'.

To prove the claim, we need to show that there exists a valuation that satisfies the mentioned properties. To this regard, we show that the following distance graph has only positive cycles. We enumerate the edges:

- $0 \xrightarrow{\leq c} x$,
- the edge from y to x has weight $min(Z_{yx}, (\leq, -c'_{xy}))$ or $min(Z_{yx}, (<, -c'_{xy}))$ depending on whether $\preccurlyeq'_{xy} = <$ or $\leq$,
- the rest of the edges are from Z

Assume for contradiction, there exists a negative cycle N in this distance graph. Clearly, N should contain an edge that was modified, that is either $0 \longrightarrow x$ or $y \longrightarrow x$. However since both are incoming to x only one of them would occur. If it is the edge $0 \longrightarrow x$, then the only possible cycle that could be negative is the cycle $0 \longrightarrow x \longrightarrow 0$. But, it has weight 0 and hence not negative. So it should be the edge $y \longrightarrow x$ that is present in N. However, the shortest path from x to y in this graph is given by the direct edge. Hence we just need to check for the cycle $x \longrightarrow y \longrightarrow x$.

For the cycle $x \longrightarrow y \longrightarrow x$ to be negative, the weight of $y \longrightarrow x$ should come from Z', in which case, it would be either $Z_{xy} + (<, -c'_{xy})$ if $Z'_{xy} = (\leq c'_{xy})$ or the other way. But from Equation 1, $Z'_{xy} < Z_{xy}$ and so $Z_{xy} + (<, -c'_{xy}) > (0, \leq)$, implying the cycle $x \longrightarrow y \longrightarrow x$ is positive.

□

4.3 Handling LU-approximation

In this section we show how to extend our inclusion test to handle LU approximation, namely how to handle the check $Z \subseteq \mathit{Closure}(\mathit{Extra}^+_{LU}(Z'))$ efficiently. In the sequel, this is denoted $Z \subseteq \mathit{Closure}^+_{LU}(Z')$. For this we need to understand first when a region intersecting Z intersects $\mathit{Extra}^+_{LU}(Z')$. Therefore, we first study the conditions that a region R should satisfy if it intersects $\mathit{Extra}^+_{LU}(Z)$ for a zone Z. We first describe a distance graph representation of $\mathit{Extra}^+_{LU}(Z)$ based on its definition.

We recall the definition given in [3] that has originally been presented using difference bound matrices (DBM). In a DBM $(c_{ij}, \prec_{i,j})$ stands for $x_i - x_j \prec_{i,j} c_{i,j}$. In the language of distance graphs, this corresponds to an edge $x_j \overset{\prec_{i,j} c_{i,j}}{\longrightarrow} x_i$; hence to Z_{ji} in our notation. Let Z^+ denote $\mathit{Extra}^+_{LU}(Z)$ and G_{Z^+} its distance graph. We have:

$$Z^+_{xy} = \begin{cases} \infty & \text{if } Z_{xy} > L(y) \\ \infty & \text{if } -Z_{y0} > L(y) \\ \infty & \text{if } -Z_{x0} > U(x), y \neq 0 \\ (<, -U(x)) & \text{if } -Z_{x0} > U(x), y = 0 \\ Z_{xy} & \text{otherwise.} \end{cases}$$

From this definition it will be important for us to note that G_{Z^+} is G_Z with some weights put to ∞ and some weights on the edges to x_0 put to $-U(x)$.

Lemma 12. *Let R be a region and Z be a zone. If $\min(G_R, G_{Z^+})$ has a negative cycle, it has a negative cycle involving only two clocks and possibly x_0.*

Proof. An easy case is when in $\min(G_R, G_{Z^+})$ a weight of an edge between two variables bound in R comes from G_{Z^+}. Using Lemma 7 we get a negative cycle on these two variables.

It remains to consider the opposite case. We need then to have an unbounded variable on the cycle. Let y be a variable unbounded in R that is part of the negative cycle. Consider y with its successor and its predecessor on the cycle: $x \longrightarrow y \longrightarrow x'$. Observe that in R every edge to y has value ∞. So the weight of the edge $x \longrightarrow y$ is from Z^+. By definition of Z^+, it is also from Z. If also the weight of the outgoing edge were from Z then we could have obtained a shorter negative cycle by choosing $x \longrightarrow x'$ from Z. Hence the weight of $y \longrightarrow x'$ comes from an edge modified in Z^+ or from R. In the first case it is $y \overset{<-U(y)}{\longrightarrow} 0$, in the second it is $y \overset{<-\alpha(y)}{\longrightarrow} 0$. However, note that since $U(y) \leq \alpha(y)$, we have $-\alpha(y) \leq -U(y)$ and therefore, in $\min(G_R, G_{Z^+})$ we could consider the edge to come from R, that is $y \overset{<-\alpha(y)}{\longrightarrow} 0$.

The same analysis as in the proof of Proposition 2 we get that the shortest cycle of this kind should be of the form $0 \longrightarrow y \longrightarrow 0$ or $0 \longrightarrow x \longrightarrow y \longrightarrow 0$; where y is an unbound variable and x is a bound variable. This cycle has the required form. □

Corollary 2. *Let R be a region and let Z be a zone. Let $Z^+ = Extra^+_{LU}(Z)$. The intersection $R \cap Z^+$ is empty iff one of the following conditions holds (below x, x' are variables bound in R and y a variable unbound in R):*

1. $Z^+_{0x} < R_{0x}$, *or*
2. $Z^+_{x0} < R_{x0}$, *or*
3. $Z^+_{xx'} < R_{xx'}$, *or*
4. $Z^+_{0y} \leq (\alpha(y), \leq)$, *or*
5. $Z^+_{xy} \leq (\alpha(y) - c, \leq)$ *where c is the constant appearing in R_{0x}.*

Efficient inclusion testing for LU approximations The conditions for a region R to not intersect the extrapolated zone Z^+ are obtained from Corollary 2. They are similar to the conditions for non-intersection of R with Z as given in Corollary 1, however with Z replaced by Z^+.

Let Z, Z' be two zones and let $G_Z, G_{Z'}$ be the respective distance graphs in canonical form. By extrapolating $G_{Z'}$ with respect to the $Extra^+_{LU}$ operator gives a zone Z^+ and a corresponding distance graph G_{Z^+}, which is not necessarily in canonical form. However, from Corollary 2, the check $Z \subseteq Closure^+_{LU}(Z')$ can be reduced to an edge by edge comparison with every region intersecting Z. Lemma 8 says that every consistent instantiation of an edge in G_Z yields a region intersecting Z. Hence, similar to the case of $Z \subseteq Closure_\alpha(Z')$, it is enough to look at edges of G_Z one by one to look at what regions we can possibly get. As a result we get an analogue of Proposition 3 with Z' replaced by Z'^+.

Proposition 4. *Let Z, Z' be zones. Writing Z'^+ for $Extra^+_{LU}(Z')$ we get that $Z \nsubseteq Closure_\alpha(Z'^+)$ iff there exist variables x, y such that one of the following conditions hold:*

1. $Z'^+_{0x} < Z_{0x} \ \wedge \ Z'^+_{0x} \leq (\alpha(x), \leq)$ *or*
2. $Z'^+_{x0} < Z_{x0} \ \wedge \ Z_{x0} \geq (-\alpha(x), \leq)$ *or*
3. $Z_{x0} \geq (-\alpha(x), \leq) \ \wedge \ Z'^+_{xy} < Z_{xy} \ \wedge \ Z'^+_{xy} \leq (\alpha(y), \leq) + Z_{x0}$

5 Experimental results

We have implemented different versions of our algorithm in a prototype tool and applied it to classical benchmarks. The results are presented in Table 1. The results show that both $Closure_\alpha$ and on-the-fly bounds computation yield improvements with respect to the existing algorithms. The best algorithm on these benchmarks is indeed $Closure^+_{LU}, otf$ that uses closures and computes LU-bounds on the fly. For comparison $Extra^+_{LU}, sa$ is the state-of-the-art algorithm used in UPPAAL. We give below some explanations for the origins of the gains that we observe.

The first improvement concerns the computation of the maximal bounds used for the abstraction as demonstrated by the examples $\mathcal{A}_2$ (Figure 3), Fischer and CSMA/CD that correspond to three different situations. In the $\mathcal{A}_2$ example, the

Model	E^+_α,sa		Cl_α,sa		Cl_α,otf		E^+_{LU},sa		Cl^+_{LU},sa		Cl^+_{LU},otf	
	nodes	s.	nodes	s.	nodes	s.	nodes	s.	nodes	s.	nodes	s.
$\mathcal{A}_1$	10014	94.84	10014	30.23	24	0.00	10014	95.62	10014	48.33	3	0.00
$\mathcal{A}_2$	4001	6.12	4001	2.56	9	0.00	4001	6.16	4001	3.64	9	0.00
$\mathcal{A}_3$	20016	102.15	20016	35.93	36	0.00	20006	99.26	20006	51.82	4	0.00
Fi7	–	–	707498	422.02	410474	162.74	48535	6.24	48535	4.85	26405	2.76
Fi8	–	–	–	–	–	–	229890	63.97	229890	33.78	95353	12.49
Fi9	–	–	–	–	–	–	1024697	558.90	1024697	250.42	339211	55.29
Fi10	–	–	–	–	–	–	–	–	–	–	1191211	322.01
C7	–	–	279588	339.36	279588	278.02	23137	6.07	23137	6.74	18034	5.94
C8	–	–	–	–	–	–	86157	40.45	86157	37.18	65745	30.92
C9	–	–	–	–	–	–	317326	283.56	317326	201.02	238594	156.56
FD10	730	5.06	663	3.33	662	3.50	726	1.89	640	3.22	640	3.35
FD20	2850	206.94	2478	88.93	2477	93.54	2846	70.13	2430	86.27	2430	90.99
FD30	–	–	5443	636.14	5442	670.77	6366	670.31	5370	622.14	5370	655.89

Table 1. Experimental results: size of reachability trees. Dash means that the algorithm did not terminate within 15 minutes. Models: Fi is Fischer, C is CSMA/CD and FD is FDDI. Approximations: $Cl_x^{(+)}$ stands for $Closure_x^{(+)}$ and E_x^+ stands for $Extra_x^+$. Bounds computation: sa stands for "static analysis" whereas otf stands for "on-the-fly". Experiments done on an Intel Xeon 2.33GHz with Linux operating system.

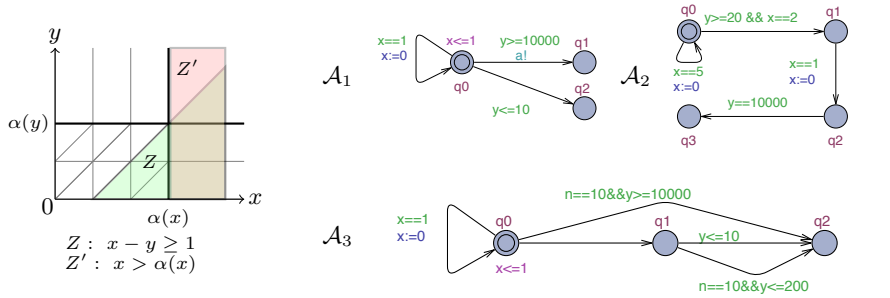

Fig. 3. Examples explaining gains obtained with the algorithm.

transition that yields the big bound 10^4 on y in q_0 is not reachable from any (q_0, Z), hence we just get the lower bound 20 on y in (q_0, Z), and a subsequent gain in performance.

The automaton $\mathcal{A}_1$ in Figure 3 illustrates the gain on the CSMA/CD. In the automaton $\mathcal{A}_1$, the transition from q_0 to q_1 is disabled as it must synchronize on letter $a!$. The static analysis algorithm [2] ignores this fact, hence it associates bound 10^4 to y in q_0. Since our algorithm computes the bounds on-the-fly, the bound for y associated to every node (q_0, Z) is 10. We observe that $Extra^+_{LU}, sa$ visits 10014 nodes on $ZG(\mathcal{A}_1)$ whereas our algorithm only visits 3 nodes. The same situation occurs in the CSMA/CD example and explains the gain given by algorithm $Closure^+_{LU}, otf$ over the algorithm $Extra^+_{LU}, sa$ used in UPPAAL.

The gains that we observe in the analysis of the Fischer's protocol are explained by the automaton $\mathcal{A}_3$ in Figure 3. $\mathcal{A}_3$ has a bounded integer variable n that is initialized to 0. Hence, the transitions from q_0 to q_2, and from q_1 to q_2, that check if n is equal to 10 are disabled. This is ignored by the static analysis algorithm that associates the bound 10^4 to clock y in q_0. Our algorithm however

associates the bound 10 to y in any node (q_0, Z). We observe that algorithm $Extra^+_{LU}, sa$ visits 20006 nodes whereas our algorithm only visits 4 nodes. A similar situation occurs in the Fischer's protocol.

Notice also that in both examples, computing the bounds on-the-fly is profitable to $Extra^+_{LU}$ as $Closure^+_{LU}, sa$ does not improve from $Extra^+_{LU}, sa$, whereas $Closure^+_{LU}, otf$ does.

The second kind of improvement comes from the $Closure_\alpha$ abstraction that particularly improves the analysis of the Fischer's and the FDDI protocols. The latter case is the most interesting as $Closure_\alpha, sa$ not only outperforms $Extra^+_M, sa$, but it also does better than $Extra^+_{LU}, sa$. Combining $Closure_\alpha$ and $Extra^+_{LU}$ improves the results even more. The situation observed on the FDDI protocol is explained in Figure 3. By its nature $Extra^+_\alpha$ abstraction does not modify a zone that has a point below maximal constants. This means that for a zone Z in the figure we have $Extra^+_\alpha(Z) = Extra^+_{LU}(Z) = Z$, and in consequence $Z' \not\subseteq Z$. However, $Z' \subseteq Closure_\alpha(Z)$. On the FDDI and the Fischer's protocols, our algorithm performs better thanks to the use of a non-convex approximation.

In our results we compare to $Extra^+_{LU}, sa$ algorithm used in UPPAAL but not to UPPAAL itself. On our examples $Extra^+_{LU}, sa$ gives similar number of zones as UPPAAL does. Curiously, for FDDI protocol UPPAAL visits much more nodes due to a different order of exploration. Of course, in general UPPAAL runs much faster than our straightforward implementation of $Extra^+_{LU}, sa$. For example we do nothing to speed up successor calculation. We also do not have an optimized memory management.

6 Conclusions

We have proposed a new algorithm for checking reachability properties of timed automata. The algorithm has two sources of improvement that are quite independent: the use of the $Closure_\alpha$ operator, and the computation of bound functions on-the-fly.

Apart from immediate gains presented in Table 1, we think that our approach opens some new perspectives on ways of improving analysis of timed systems. We show that the use of non-convex approximations can be efficient. We have used very simple approximations, but it may be well the case that there are more sophisticated approximations to be discovered. The structure of our algorithm permits to calculate bounding constants on the fly. One should note that standard benchmarks are very well understood and very well modeled. In particular they have no "superfluous" constraints or clocks. In the analysis of not-so-clean models coming from systems in practice one can expect that on-the-fly approach can be even more beneficial.

There are numerous directions for further research. One of them is to find other approximation operators. Methods for constraint propagation also deserve a closer look. Depending on the propagation strategy we can make our algorithm more depth-first or more breadth-first. One can expect that a depth-first

approach will be more efficient when a final state is reachable, while breath-first will be better when it is not reachable.

References

1. R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
2. G. Behrmann, P. Bouyer, E. Fleury, and K. G. Larsen. Static guard analysis in timed automata verification. In *TACAS'03*, volume 2619 of *LNCS*, pages 254–270. Springer, 2003.
3. G. Behrmann, P. Bouyer, K. G. Larsen, and R. Pelanek. Lower and upper bounds in zone-based abstractions of timed automata. *Int. Journal on Software Tools for Technology Transfer*, 8(3):204–215, 2006.
4. G. Behrmann, A. David, K. G Larsen, J. Haakansson, P. Pettersson, W. Yi, and M. Hendriks. Uppaal 4.0. In *QEST'06*, pages 125–126, 2006.
5. B. Bérard, B. Bouyer, and A. Petit. Analysing the pgm protocol with UPPAAL. *Int. Journal of Production Research*, 42(14):2773–2791, 2004.
6. P. Bouyer. Forward analysis of updatable timed automata. *Form. Methods in Syst. Des.*, 24(3):281–320, 2004.
7. M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, and S. Yovine. KRONOS: a mode-checking tool for real-time systems. In *CAV'98*, volume 1427 of *LNCS*, pages 546–550. Springer, 1998.
8. C. Courcoubetis and M. Yannakakis. Minimum and maximum delay problems in real-time systems. *Form. Methods Syst. Des.*, 1(4):385–415, 1992.
9. C. Daws and S. Tripakis. Model checking of real-time reachability properties using abstractions. In *TACAS'98*, volume 1384 of *LNCS*, pages 313–329. Springer, 1998.
10. D. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Automatic Verification Methods for Finite State Systems*, volume 407 of *Lecture Notes in Computer Science*, pages 197–212. Springer, 1989.
11. K. Havelund, A. Skou, K. Larsen, and K. Lund. Formal modeling and analysis of an audio/video protocol: An industrial case study using UPPAAL. In *RTSS'97*, pages 2–13, 1997.
12. J. Zhao, X. Li, and G. Zheng. A quadratic-time dbm-based successor algorithm for checking timed automata. *Inf. Process. Lett.*, 96(3):101–105, 2005.