# 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

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 an initial state to a 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 in current algorithms extrapolation of a zone is always a zone; and in particular 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.
An additional benefit of the proposed approach is that it 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 algorithms are at the core of verification tools like Uppaal [4] or RED [15], and are used in industrial case studies [11, 6]. The standard solution constructs a search tree whose nodes are approximations of zones. In this paper we give an efficient algorithm for checking if a zone is included in an approximation of another zone. This enables a reachability algorithm to work with search trees whose nodes are just unapproximated zones. This has numerous advantages: one can use non-convex approximations, and one can compute approximating parameters on the fly.

The first solution to the reachability problem has used *regions*, which are equivalence classes of clock valuations. Subsequent research has shown that the

region abstraction is very inefficient and an other method using *zones* instead of regions has been proposed. This can be implemented efficiently using DBMs [10] and is used at present in almost 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, the 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 convex approximation that is also based on (refined) regions.

We propose a new algorithm for the reachability problem using closures of zones. To this effect we provide an efficient algorithm for checking if a zone is included in a closure of another zone. In consequence we can work with non-convex approximations without a need to store them explicitly.

Thresholds for approximations are very important for efficient implementation. Good thresholds give substantial gains in time and space. The simplest approach is to take as a threshold the maximal constant appearing in a transition 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 show that we can accommodate closure on top of the LU-approximation at no extra cost too.

Since our algorithm never stores approximations, we can compute thresholds on-the-fly. This means that our computation of thresholds does not take into account unreachable states. In consequence in some cases we get much better LU-thresholds than those obtained by static analysis. This happens in particular in a very common context of analysis of parallel compositions of timed automata.

*Related work* The topic of this paper is approximation of zones and efficient handling of them. We show that it is possible to use non-convex approximations and that it can be done efficiently. In particular, we improve on state of the art approximations [3]. Every forward algorithm needs approximations, so our work can apply to tools like RED or UPPAAL.

Recent work [14] reports on backward analysis approach using general linear constraints. This approach does not use approximations and relies on SMV solver to simplify the constraints. Comparing forward and backward methods would require a substantial test suite, and is not the subject of this paper.

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 for efficient inclusion test between a zone and a closure of another zone. The algorithm constructing the search tree and calculating approximations on-the-fly is presented in Section 4. Some results obtained with a prototype implementation are presented in the last section. For reasons of space all the missing proofs are presented in the appendix.

## 2 Preliminaries

*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. A *clock constraint* is a conjunction of constraints $x\#c$ for $x \in X$, $\# \in \{<, \leq, =, \geq, >\}$ and $c \in \mathbb{N}$, e.g. $(x \leq 3 \wedge y > 0)$. 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 $\nu$ satisfies $\phi \in \Phi(X)$, i.e. when every constraint in $\phi$ 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. An example of TA is depicted in Figure 1. 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 $(q', \nu')$ is called a *successor* of $(q, \nu)$. 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 $(q_n, \nu_n)$ with $q_n \in Acc$. The *reachability problem* is to decide if a given automaton has an accepting run. This problem is known to be PSPACE-complete [1, 8].

*Symbolic semantics for timed automata.* The reachability problem is solved using so-called symbolic semantics. It 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 $\nu'$ 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 infer that $\mathcal{A}$ has an accepting run iff there is a node $(q, Z)$ reachable in $ZG(\mathcal{A})$ with $q \in Acc$.

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

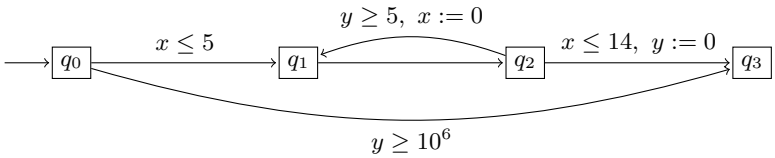


**Fig. 1.** Timed automaton $\mathcal{A}$.

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 intersect $S$. This gives the finite abstraction.

Let us consider a *bound function* associating to each clock $x$ of $\mathcal{A}$ a bound $\alpha_x \in \mathbb{N}$. A *region* [1] with respect to $\alpha$ is the set of valuations specified as follows:

1. for each clock $x \in X$, one constraint from the set:
   $\{x = c \mid c = 0, \ldots, \alpha_x\} \cup \{c - 1 < x < c \mid c = 1, \ldots, \alpha_x\} \cup \{x > \alpha_x\}$
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 $S$ [7]. 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 $\nu'$ 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 $\alpha$. 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.** *[7] $\mathcal{A}$ has an accepting run iff there is a reachable node $(q, S)$ in $SG_\alpha(\mathcal{A})$ with $q \in Acc$ and $\alpha_{\mathcal{A}} \leq \alpha$.*

For efficiency it is important to get good bound function $\alpha$. 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 $\alpha$ yield a coarser, hence smaller, symbolic graph $SG_\alpha(\mathcal{A})$. Note that current implementations do not use closure but some convex under-approximation of it that makes the graph even bigger.

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. Consider for instance the automaton $\mathcal{A}$ in Figure 1. Looking at the guards, we get that $\alpha_x = 14$ and $\alpha_y = 10^6$. Yet, a closer look at the automaton reveals that in the state $q_2$ it is enough to take the bound $\alpha_y(q_2) = 5$. This observation from [2] points out that one can often get very big gains by associating a bound function $\alpha(q)$ to each state $q$ in $\mathcal{A}$ that is later used for the abstraction of nodes of the form $(q, Closure_{\alpha(q)}(S))$. In op. cit. an algorithm for inferring bounds based on static analysis of the structure of the automaton is proposed. In Section 4 we will show how to calculate these bounds on-the-fly during the exploration of the automaton's state space.

## 3 Efficient testing of inclusion in a closure of a zone

The tests of the form $Z \subseteq Closure_\alpha(Z')$ will be at the core of the new algorithm we propose. 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 with neither the need to represent nor to compute the closure.

We start by examining the question as to how one decides if a region $R$ intersects a zone $Z$. The important point is that it is enough to verify that the projection on every pair of variables is nonempty. This is the cornerstone for the efficient inclusion testing algorithm that even extends to LU-approximations.

*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, 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 [5].

*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 $<$.
*Minus* $-(\preccurlyeq, c) = (\preccurlyeq, -c)$.
*Order* $(\preccurlyeq_1, c_1) < (\preccurlyeq_2, c_2)$ if either $c_1 < c_2$ or ($c_1 = c_2$ and $\preccurlyeq_1 = <$ and $\preccurlyeq_2 = \leq$).
*Floor* $\lfloor (<, c) \rfloor = (\leq, c - 1)$ and $\lfloor (\leq, c) \rfloor = (\leq, c)$.

This arithmetic lets us 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 useful proposition is a folklore.

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

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 $Z$.

We can now state a necessary and sufficient condition for $R \cap Z$ to be empty in terms of cycles in distance graphs. We denote by $Z_{xy}$ the weight of the edge $x \xrightarrow{\preccurlyeq_{xy} c_{xy}} y$ in the canonical distance graph representing $Z$. Similarly for $R$.

**Proposition 2.** *Let $R$ be a region and let $Z$ be a zone. The intersection $R \cap Z$ is empty iff there exist variables $x, y$ such that $Z_{yx} + R_{xy} \leq (<, 0)$.*

A variant of this fact has been proven as an intermediate step of Proposition 2 in [7]. We give a different proof that allows us to derive an efficient inclusion test presented below.

*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 $\mathcal{O}(|X|^2)$ complexity, since this is the complexity of current algorithms used for checking inclusion of two zones. Proposition 2 can be used to efficiently test the inclusion $R \subseteq \mathit{Closure}(Z')$. It remains to understand what are the regions intersecting the zone $Z$ and then to consider all possible cases. The next lemma basically says that every consistent instantiation of an edge in $G_Z$ leads to a region intersecting $Z$.

**Lemma 1.** *Let $G$ be a distance graph in canonical form, with all cycles positive. Let $x, y$ be two variables, and let $x \xrightarrow{\preccurlyeq_{xy} c_{xy}} y$ and $y \xrightarrow{\preccurlyeq_{yx} c_{yx}} x$ be edges in $G$. For every $d \in \mathbb{R}$ such that $d \preccurlyeq_{xy} c_{xy}$ and $-d \preccurlyeq_{yx} c_{yx}$ there exists a valuation $v \in [\![G]\!]$ with $v(y) - v(x) = d$.*

Thanks to this lemma it is enough to look at edges of $G_Z$ one by one to see what regions we can get. This insight is used to get the desired efficient inclusion test

**Theorem 2.** *Let $Z, Z'$ be zones. Then, $Z \not\subseteq \mathit{Closure}_\alpha(Z')$ iff there exist variables $x$, $y$, both different from $x_0$, such that one of the following conditions hold:*

1. *$Z'_{0x} < Z_{0x}$ and $Z'_{0x} \leq (\leq, \alpha_x)$, or*
2. *$Z'_{x0} < Z_{x0}$ and $Z_{x0} \geq (\leq, -\alpha_x)$, or*
3. *$Z_{x0} \geq (\leq, -\alpha_x)$ and $Z'_{xy} < Z_{xy}$ and $Z'_{xy} \leq (\leq, \alpha_y) + \lfloor Z_{x0} \rfloor$.*

*Comparison with the algorithm for $Z \subseteq Z'$.* Given two zones $Z$ and $Z'$, the procedure for checking $Z \subseteq Z'$ works on two graphs $G_Z$ and $G_{Z'}$ that are in canonical form. This form reduces the inclusion test to comparing the edges of the graphs one by one. Note that our algorithm for $Z \subseteq \mathit{Closure}_\alpha(Z')$ does not do worse. It works on $G_Z$ and $G_{Z'}$ too. The edge by edge checks are only marginally more complicated. The overall procedure is still $\mathcal{O}(|X|^2)$.

*Handling LU-approximation.* In [3] the authors propose to distinguish between maximal constants used in upper and lower bounds comparisons: for each clock $x$, $L_x \in \mathbb{N} \cup \{-\infty\}$ represents the maximal constant $c$ such that there exists a constraint $x > c$ or $x \geq c$ in a guard of a transition in the automaton; dually, $U_x \in \mathbb{N} \cup \{-\infty\}$ represents the maximal constant $c$ such that there is a constraint $x < c$ or $x \leq c$ in a guard of a transition. If such a $c$ does not exist, then it is considered to be $-\infty$. 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 now explain how to extend our inclusion test to handle LU approximation, namely given $Z$ and $Z'$ how to directly check $Z \subseteq Closure_\alpha(Extra^+_{LU}(Z'))$ efficiently. Observe that for each $x$, the maximal constant $\alpha_x$ is the maximum of $L_x$ and $U_x$. In the sequel, this is denoted $Z \subseteq Closure^+_{LU}(Z')$. For this we need to understand first when a region intersecting $Z$ intersects $Extra^+_{LU}(Z')$. Therefore, we study the conditions that a region $R$ should satisfy if it intersects $Extra^+_{LU}(Z)$ for a zone $Z$.

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 $x_j \xrightarrow{\prec_{i,j} c_{i,j}} x_i$; hence to $Z_{ji}$ in our notation. Let $Z^+$ denote $Extra^+_{LU}(Z)$ and $G_{Z^+}$ its distance graph. We have:

$$Z^+_{xy} = \begin{cases} (<, \infty) & \text{if } Z_{xy} > (\leq, L_y) \\ (<, \infty) & \text{if } -Z_{y0} > (\leq, L_y) \\ (<, \infty) & \text{if } -Z_{x0} > (\leq, U_x), y \neq 0 \\ (<, -U(x)) & \text{if } -Z_{x0} > (\leq, U_x), y = 0 \\ Z_{xy} & \text{otherwise.} \end{cases} \tag{1}$$

From this definition it will be important for us to note that $G_{Z^+}$ is $G_Z$ with some weights put to $(<, \infty)$ and some weights on the edges to $x_0$ put to $(<, -U_x)$. Note that $Extra^+_{LU}(Z')$ is not in the canonical form. If we put $Extra^+_{LU}(Z')$ into the canonical form then we could just use Theorem 2. We cannot afford to do this since canonization can take cubic time [5]. The following theorem implies that we can do the test without canonizing $Extra^+_{LU}(Z')$. Hence we can get a simple quadratic test also in this case.

**Theorem 3.** *Let $Z, Z'$ be zones. Let $Z'^+$ denote $Extra^+_{LU}(Z')$ obtained from $Z'$ using Equation 1 for each edge. Note that $Z'^+$ is not necessarily in canonical form. Then, we get that $Z \nsubseteq Closure_\alpha(Z'^+)$ iff there exist variables $x$, $y$ different form $x_0$ such that one of the following conditions hold:*

1. $Z'^+_{0x} < Z_{0x}$ *and* $Z'^+_{0x} \leq (\leq, \alpha_x)$, *or*
2. $Z'^+_{x0} < Z_{x0}$ *and* $Z_{x0} \geq (\leq, -\alpha_x)$, *or*
3. $Z_{x0} \geq (-\alpha_x, \leq)$ *and* $Z'^+_{xy} < Z_{xy}$ *and* $Z'^+_{xy} \leq (\leq, \alpha_y) + \lfloor Z_{x0} \rfloor$.

## 4 A New Algorithm for Reachability

Our goal is to decide if a final state of a given timed automaton is reachable. We do it by computing a finite prefix of the reachability tree of the zone graph $ZG(\mathcal{A})$ that is sufficient to solve the reachability problem. Finiteness is ensured by not exploring a node $(q, Z)$ if there exists a $(q, Z')$ such that $Z \subseteq Closure_\alpha(Z')$, for a suitable $\alpha$. We will first describe a simple algorithm based on the closure and then we will address the issue of finding tighter bounds for the clock values.

*The basic algorithm.* Given a timed automaton $\mathcal{A}$ we first calculate the bound function $\alpha_{\mathcal{A}}$ as described just before 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 algorithm does not use the classical extrapolation operator named $Extra^+_M$ in [3] and $Extra^+_\alpha$ hereafter, but the coarser $Closure_\alpha$ operator [7]. 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 [7]). This is as efficient as testing $Z \subseteq Z'$ thanks to the algorithm presented in the previous section.

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 $Z$ [7, 3]. Other important optimizations of the standard algorithm concern finer computation of bounding functions $\alpha$. We now show that the structure of the proposed algorithm allows to improve this too.

*Computing clock bounds on-the-fly.* We can improve on the idea of Behrmann et al. [2] of computing a bound function $\alpha_q$ for each state $q$. We will compute these bounding functions on-the-fly and they will depend also on a zone and not just a state. 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.

```
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') nontentative
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) nontentative
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.

Our modified algorithm is given in Figure 2. It computes a tree whose nodes are triples $(q, Z, \alpha)$ where $(q, Z)$ is a node of $ZG(\mathcal{A})$ and $\alpha$ is a bound function. Each node $(q, Z, \alpha)$ has as many child nodes $(q_s, Z_s, \alpha_s)$ 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 for correctness of our constant propagation procedure. Each node is further marked either *tentative* or *nontentative*. The leaf nodes $(q, Z, \alpha)$ 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 *nontentative*.

Our algorithm starts from the root node $(q_0, Z_0, \alpha_0)$, consisting of the initial state, initial zone, and the function mapping each clock to $-\infty$. It repeatedly alternates an exploration and a resolution phase as described below.

*Exploration phase.* 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 *nontentative* 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 these 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 of these propagations is to maintain the following invariant. For every node $n = (q, Z, \alpha)$:

1. if $n$ is *nontentative*, then $\alpha$ is the maximum of the $\alpha_s$ from all successor nodes $(q_s, Z_s, \alpha_s)$ of $n$ (taking into account guards and resets as made precise in the function `max_edge`);

2. if $n$ is *tentative* with respect to $(q', Z', \alpha')$, then $\alpha$ is equal to $\alpha'$.

The result of propagation 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 $(q, Z, \alpha)$ never exceed those that are computed by the static guard analysis.

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 $\alpha'$ 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, $\alpha$ is a solution to the equations in [2] applied on $ZG(\mathcal{A})$.

It could seem that the algorithm will be forced to do a high number of propagations of bounds. The experiments 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 $\alpha$ 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 $\alpha$ to $-\infty$ is safe as $\alpha$ 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.

**Theorem 4.** *An accepting state is reachable in $ZG(\mathcal{A})$ iff the algorithm reaches a node with an accepting state and a non-empty zone.*

*Handling LU approximations.* Recall that $Extra^+_{LU}(Z)$ approximation used two bounds: $L_x$ and $U_x$ for each clock $x$. 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 $L'$ and $U'$ are the bounds calculated for $(q', Z')$ and $\alpha'_x = \max(L'_x, U'_x)$ for every clock $x$. As discussed in Section 3, this test can be done efficiently too. The proof of correctness of the resulting algorithm is only slightly more complicated.

| Model | Our algorithm | | UPPAAL's algorithm | | UPPAAL 4.1.3 (-n4 -C -o1) | |
|---|---|---|---|---|---|---|
| | nodes | s. | nodes | s. | nodes | s. |
| $\mathcal{A}_1$ | 2 | 0.00 | 10003 | 0.07 | 10003 | 0.07 |
| $\mathcal{A}_2$ | 7 | 0.00 | 3999 | 0.60 | 2003 | 0.01 |
| $\mathcal{A}_3$ | 3 | 0.00 | 10004 | 0.37 | 10004 | 0.32 |
| CSMA/CD7 | 5031 | 0.32 | 5923 | 0.27 | – | T.O. |
| CSMA/CD8 | 16588 | 1.36 | 19017 | 1.08 | – | T.O. |
| CSMA/CD9 | 54439 | 6.01 | 60783 | 4.19 | – | T.O. |
| FDDI10 | 459 | 0.02 | 525 | 0.06 | 12049 | 2.43 |
| FDDI20 | 1719 | 0.29 | 2045 | 0.78 | – | T.O. |
| FDDI30 | 3779 | 1.29 | 4565 | 4.50 | – | T.O. |
| Fischer7 | 7737 | 0.42 | 20021 | 0.53 | 18374 | 0.35 |
| Fischer8 | 25080 | 1.55 | 91506 | 2.48 | 85438 | 1.53 |
| Fischer9 | 81035 | 5.90 | 420627 | 12.54 | 398685 | 8.95 |
| Fischer10 | – | T.O. | – | T.O. | 1827009 | 53.44 |

**Table 1.** Experimental results: number of visited nodes and running time with a timeout (T.O.) of 60 seconds. Experiments done on a MacBook with 2.4GHz Intel Core Duo processor and 2GB of memory running MacOS X 10.6.7.

## 5 Experimental results

We have implemented the algorithm from Figure 2, and have tested it on classical benchmarks. The results are presented in Table 1, along with a comparison to UPPAAL and our implementation of UPPAAL's core algorithm that uses the $Extra^+_{LU}$ extrapolation [3] and computes bounds by static analysis [2]. Since we have not considered symmetry reduction [12] in our tool, we have not used it in UPPAAL either.

The comparison to UPPAAL is not meaningful for the CSMA/CD and the FDDI protocols. Indeed, UPPAAL runs out of time even if we significantly increase the time allowed; switching to breadth-first search has not helped either. We suspect that this is due to the order in which UPPAAL takes the transitions in the automaton. For this reason in columns 4 and 5, we provide results from our own implementation of UPPAAL's algorithm that takes transitions in the same order as the implementation of our algorithm. Although RED also uses approximations, it is even more difficult to draw a meaningful comparison with it, since it uses symbolic state representation unlike UPPAAL or our tool. Since this paper is about approximation methods, and not tool comparison, we leave more extensive comparisons as further work.

The results show that our algorithm provides important gains. Analyzing the results more closely we could see that both the use of closure, and on-the-fly computation of bounds are important. In Fischer's protocol our algorithm visits much less nodes. In the FDDI protocol, the size of DBMs is huge and nevertheless our inclusion test based on *Closure* is significantly better in the running time. The CSMA/CD case shows that the cost of bounds propagation does not always counterbalance the gains. However the overhead is not very high either. We comment further on the results below.

The first improvement comes from 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,

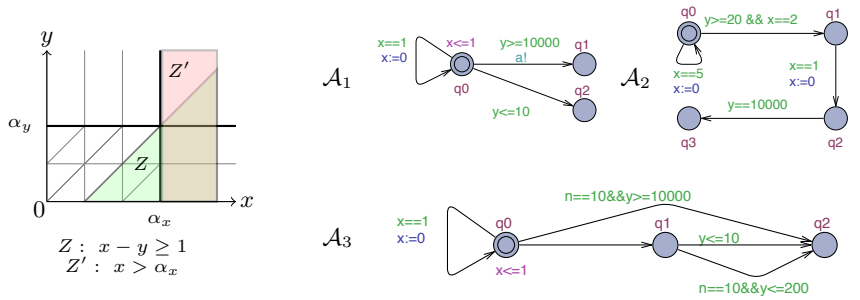


**Fig. 3.** Examples explaining gains obtained with the algorithm.

the 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 protocol. 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, $y$ is associated the bound 10 in every node $(q_0, Z)$. We observe that UPPAAL's algorithm visits 10003 nodes on $ZG(\mathcal{A}_1)$ whereas our algorithm only visits 2 nodes. The same situation occurs in the CSMA/CD example. However despite the improvement in the number of nodes (roughly 10%) the cost of computing the bounds impacts the running time negatively.

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 every node $(q_0, Z)$. We observe that UPPAAL's algorithm visits 10004 nodes whereas our algorithm only visits 3 nodes. A similar situation occurs in the Fischer's protocol. We include the last row to underline that our implementation is not as mature as UPPAAL. We strongly think that UPPAAL could benefit from methods presented here.

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 situation observed on the FDDI protocol is explained in Figure 3. For the zone $Z$ in the figure, by definition $Extra^+_{LU}(Z) = Z$, and in consequence $Z' \not\subseteq Z$. However, $Z' \subseteq Closure_\alpha(Z)$. On FDDI and Fischer's protocols, our algorithm performs better due to the non-convex approximation.

## 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 indepen-

dent: 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 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. However in not-so-clean models coming from systems in practice one can expect the on-the-fly approach to 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. We believe that our approximations methods are compatible with partial order reductions [12, 13]. We hope that the two techniques can benefit from each other.

## 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*, 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. J. Bengtsson and W. Yi. Timed automata: Semantics, algorithms and tools. *Lectures on Concurrency and Petri Nets*, pages 87–124, 2004.
6. B. Bérard, B. Bouyer, and A. Petit. Analysing the pgm protocol with UPPAAL. *Int. Journal of Production Research*, 42(14):2773–2791, 2004.
7. P. Bouyer. Forward analysis of updatable timed automata. *Form. Methods in Syst. Des.*, 24(3):281–320, 2004.
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 *AVMFSS*, volume 407 of *LNCS*, 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*, pages 2–13, 1997.
12. M. Hendriks, G. Behrmann, K. G. Larsen, P. Niebert, and F. Vaandrager. Adding symmetry reduction to Uppaal. In *Int. Workshop on Formal Modeling and Analysis of Timed Systems*, volume 2791 of *LNCS*, pages 46–59. Springer, 2004.

13. J. Malinowski and P. Niebert. Sat based bounded model checking with partial order semantics for timed automata. In *TACAS*, volume 6015 of *LNCS*, pages 405–419, 2010.
14. Georges Morbé, Florian Pigorsch, and Christoph Scholl. Fully symbolic model checking for timed automata. In *CAV*, 2011. To Appear.
15. Farn Wang. Efficient verification of timed automata with bdd-like data structures. *Int. J. on Software Tools for Technology Transfer*, 6:77–97, 2004.

## A Proofs from Section 3

We provide all the proofs from the section presenting the efficient inclusion testing algorithm. For convenience, we recall the statements of the facts that are proven together with their original numbering. They are preceded with black arrow for readability.

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

*Proof.* 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 say that a variable $x$ is *fixed* in $\overline{G}$ if in this graph we have edges $0 \xrightarrow{\leq c_x} x$ and $x \xrightarrow{\leq -c_x} 0$ for some constant $c_x$. These edges mean that every valuation in $[\![\overline{G}]\!]$ should assign $c_x$ to $x$.

If all the variables in $\overline{G}$ are fixed then the value of every cycle in $\overline{G}$ is 0, and the valuation assigning $c_x$ to $x$ for every variable $x$ is the unique valuation in $[\![\overline{G}]\!]$. Hence, $[\![\overline{G}]\!]$, and in consequence $[\![G]\!]$ are not empty.

Otherwise there is a variable, say $y$, that is not fixed in $\overline{G}$. We will show how to fix it. Let us multiply all the constraints in $\overline{G}$ by 2. This means that we change each arrow $x_1 \xrightarrow{\preccurlyeq c} x_2$ to $x_1 \xrightarrow{\preccurlyeq 2c} x_2$. Let us call the resulting graph $H$. Clearly $H$ is in canonical form since $\overline{G}$ is. Moreover $[\![H]\!]$ is not empty iff $[\![\overline{G}]\!]$ is not empty. The gain of this transformation is that for our chosen variable $y$ we have in $H$ edges $0 \xrightarrow{\preccurlyeq c_{0y}} y$ and $y \xrightarrow{\preccurlyeq c_{y0}} 0$ with $c_{y0} + c_{0y} \geq 2$. This means that there is a natural number $d$ such that $(\leq, d) \leq (\preccurlyeq c_{0y})$ and $(\leq, -d) \leq (\preccurlyeq c_{y0})$. Let $H_d$ be $H$ with edges to and from $y$ changed to $0 \xrightarrow{\leq d} y$ and $y \xrightarrow{\leq -d} 0$, respectively. This is a distance graph where $y$ is fixed. We need to show that there is no negative cycle in this graph.

Suppose that there is a negative cycle in $H_d$. Clearly it has to pass through 0 and $y$ since there was no negative cycle in $H$. Suppose that it uses the edge $0 \xrightarrow{\leq d} y$, and suppose that the next used edge is $y \xrightarrow{\preccurlyeq c_{yx}} x$. The cycle cannot come back to $y$ before ending in 0 since then we could construct a smaller negative cycle. Hence all the other edges in the cycle come from $H$. Since $H$ is in the canonical form, a path from $x$ to 0 can be replaced by the edge from $x$ to 0,

and the value of the path will not increase. This means that our hypothetical negative cycle has the form $0 \xrightarrow{\leq d} y \xrightarrow{\preccurlyeq c_{yx}} x \xrightarrow{\preccurlyeq c_{x0}} 0$. By canonicity of $H$ we have $(\preccurlyeq_{yx}, c_{yx}) + (\preccurlyeq_{x0}, c_{x0}) \geq (\preccurlyeq_{y0}, c_{y0})$. Putting these two facts together we get

$$(\leq, 0) > (\leq, d) + (\preccurlyeq_{yx}, c_{yx}) + (\preccurlyeq_{x0}, c_{x0}) \geq (\leq, d) + (\preccurlyeq_{y0}, c_{y0})$$

but this contradicts the choice of $d$ which supposed that $(\leq, d) + (\preccurlyeq_{y0}, c_{y0})$ is positive. The proof when the hypothetical negative cycle passes through the edge $y \xrightarrow{\leq -d} 0$ is analogous.

Summarizing, starting from $G$ that has no negative cycles we have constructed a graph $H_d$ that has no negative cycles, and has one more variable fixed. We also know that if $[\![H_d]\!]$ is not empty then $[\![G]\!]$ is not empty. Repeatedly applying this construction we get a graph where all the variables are fixed and no cycle is negative. As we have seen above the semantics of such a graph is not empty. □

►**Proposition 2.** Let $R$ be a region and let $Z$ be a zone. The intersection $R \cap Z$ is empty iff there exist variables $x, y$ such that $Z_{yx} + R_{xy} \leq (<, 0)$.

Before proving the above proposition, we will start with some notions. 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 $R$.

For two distance graphs $G_1$, $G_2$ which are not necessarily in canonical form, 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 $G_2$. Even though this graph may be not in canonical form, it should be clear that it represents intersection of the two arguments, that is, $[\![\min(G_1, G_2)]\!] = [\![G_1]\!] \cap [\![G_2]\!]$; in other words, 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 $G_2$.

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

**Lemma 2.** *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 edges $x_1 \xrightarrow{\leq d} x_2$ and $x_2 \xrightarrow{\leq -d} x_1$ in $G_R$. Since the edge $x_1 \xrightarrow{\preccurlyeq c} x_2$ comes from $G$ we have $c < d$ or $c = d$ and $\preccurlyeq$ is the strict inequality. We get a negative cycle $x_1 \xrightarrow{\preccurlyeq c} x_2 \xrightarrow{\leq -d} x_1$.

In the second case we have edges $x_2 \xrightarrow{<-d+1} x_1$ and $x_1 \xrightarrow{\leq d} x_2$ in $R$. Hence $c < d$ and $x_1 \xrightarrow{\preccurlyeq c} x_2 \xrightarrow{<-d+1} x_1$ gives a negative cycle. □

**Proof of Proposition 2** Let $G_R$, $G_Z$ be the canonical distance graphs representing the region $R$ and the zone $Z$ respectively. One direction is immediate: 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. Again, 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 2 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: $x \longrightarrow y \longrightarrow x'$. We will show that we can assume that $x'$ is $x_0$. Observe that in $G_R$ every edge to $y$ has value $\infty$. So the weight of the edge $x \longrightarrow y$ is from $G_Z$. If the weight of the outgoing edge is also from $G_Z$ then we could have eliminated $y$ from the cycle by choosing $x \longrightarrow x'$ from $G_Z$. Hence the weight of $y \longrightarrow x'$ comes from $G_R$. Since $y$ is unbounded in $R$, the weight of this edge is $d - \alpha_y$, where $d$ is the value on the edge $0 \xrightarrow{\preccurlyeq d} x'$ in $G_R$. 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 $y \xrightarrow{<-\alpha_y} x_0$ 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 $x_0 \longrightarrow x \longrightarrow y \longrightarrow x_0$ with the edges $x_0 \to x$ and $y \to x_0$ coming from $G_R$ and the edge $x \to y$ coming from $G_Z$. Since $G_R$ is canonical, we can reduce this cycle to $x \to y \to x$ with $x \to y$ coming from $G_Z$ and $y \to x$ coming from $G_R$. □

### A.1 Efficient inclusion testing

Given two zones $Z$ and $Z'$ and a bound function $\alpha$, we would like to know if $Z \not\subseteq \mathit{Closure}_\alpha(Z')$: that is, does there exist a region $R$ that intersects $Z$ but does not intersect $Z'$? From Proposition 2 this reduces to asking if there exists a region $R$ that intersects $Z$ and two variables $x, y$ such that $Z'_{yx} + R_{xy} < (\leq, 0)$. This

brings us to look for the least value of $R_{xy}$ from among the regions $R$ intersecting $Z$. We begin with the observation that every consistent instantiation of an edge in a canonical distance graph $G$ gives a valuation satisfying the constraints of $G$.

►**Lemma 1.** Let $G$ be a distance graph in canonical form, with all cycles positive. Let $x, y$ be two variables such that $x \xrightarrow{\preccurlyeq_{xy} c_{xy}} y$ and $y \xrightarrow{\preccurlyeq_{yx} c_{yx}} x$ 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.* Take $d$ as in the assumption of the lemma. Let $G_d$ be the distance graph where we have the edges $x \xrightarrow{\leq d} y$, and $y \xrightarrow{\leq -d} x$ 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. □

Recall that for a zone $Z$, we denote by $Z_{xy}$ the weight of the edge $x \xrightarrow{\preccurlyeq_{xy} c_{xy}} y$ in the canonical distance graph representing $Z$. We denote by $[v]$ the region to which $v$ belongs to; $[v]_{xy}$ denotes the value $(\preccurlyeq_{xy}, r_{xy})$ of the constraint $y - x \preccurlyeq_{xy} r_{xy}$ defining the region $[v]$. This is precisely the value of the edge $x \xrightarrow{\preccurlyeq_{xy} r_{xy}} y$ in the canonical distance graph representing the region $[v]$. We are interested in finding the least value of $[v]_{xy}$ from among the valuations $v \in Z$. Lemmas 3 and 4 describe this least value of $[v]_{xy}$ for different combinations of $x$ and $y$.

For a weight $(\preccurlyeq, c)$ we define $-(\preccurlyeq, c)$ as $(\preccurlyeq, -c)$. We now define a *ceiling* function $\lceil \cdot \rceil$ for weights.

**Definition 1.** *For a real $c$, let $\lceil c \rceil$ denote the smallest integer that is greater than or equal to $c$. We define the* ceiling *function $\lceil (\preccurlyeq, c) \rceil$ for a weight $(\preccurlyeq, c)$ depending on whether $\preccurlyeq$ equals $\leq$ or $<$, as follows:*

$$\lceil (\leq, c) \rceil = \begin{cases} (\leq, c) & \text{if } c \text{ is an integer} \\ (<, \lceil c \rceil) & \text{otherwise} \end{cases}$$

$$\lceil (<, c) \rceil = \begin{cases} (<, c+1) & \text{if } c \text{ is an integer} \\ (<, \lceil c \rceil) & \text{otherwise} \end{cases}$$

**Lemma 3.** *Let $Z$ be a non-empty zone and let $x$ be a variable different from $x_0$. Then, from among the regions $R$ that intersect $Z$:*

– *the least value of* $R_{0x}$ *is given by*

$$\begin{cases} (<, \infty) & \text{if } Z_{x0} < (\leq, -\alpha_x) \\ \lceil -Z_{x0} \rceil & \text{otherwise} \end{cases}$$

– *the least value of* $R_{x0}$ *is given by* $\max\{\lceil -Z_{0x} \rceil, (<, -\alpha_x)\}$.

*Proof.* Let $Z_{0x} = (\preccurlyeq_{0x}, c_{0x})$ and $Z_{x0} = (\preccurlyeq_{x0}, c_{x0})$.

For the least value of $R_{0x}$, first note that if $Z_{x0} < (\leq, -\alpha_x)$ then all valuations $v \in Z$ have $v_x > \alpha_x$ and by definition $[v]_{0x} = (<, \infty)$ for such valuations. If not, we know that for all valuations $-v_x \preccurlyeq_{x0} c_{x0}$, that is, $v_x \succcurlyeq_{x0} -c_{x0}$. If $\preccurlyeq_{x0}$ is $\leq$, then from Lemma 1 there exists a valuation with $v_x = -c_{x0}$ and this is the minimum value that can be attained. When $\preccurlyeq_{x0}$ is $<$, then we can find a positive $\epsilon < 1$ such that $c_{x0} - \epsilon \preccurlyeq_{x0} c_{x0}$ and $-c_{x0} + \epsilon \preccurlyeq_{0x} c_{0x}$. From Lemma 1, there exists a valuation with $v_x = -c_{x0} + \epsilon$ for which $[v]_{0x} = (<, -c_{x0} + 1)$. Since $\preccurlyeq_{x0}$ is a strict $<$, this is the minimum value for $R_{0x}$. This gives that the minimum value is $\lceil -Z_{x0} \rceil$.

Now we look at the minimum value for $R_{x0}$. If $(\preccurlyeq_{0x}, c_{0x}) \leq (\leq, \alpha_x)$, then all valuations $v$ have $v_x \leq \alpha_x$ and by an argument similar to above, the minimum value of $R_{x0}$ would be given by $\lceil -Z_{0x} \rceil$. Since $(\preccurlyeq_{0x}, c_{0x}) \leq (\leq, \alpha_x)$, we have $(\preccurlyeq_{0x}, -c_{0x}) \geq (\leq, -\alpha_x) > (<, -\alpha_x)$. If $(\preccurlyeq_{0x}, c_{0x}) > (\leq, \alpha_x)$, then from Lemma 1, there are valuations in $Z$ with $v_x > \alpha_x$ and for these valuations, $[v]_{x0} = (< , -\alpha_x)$. In this case the minimum value is given by $(<, -\alpha_x)$. Since $(\preccurlyeq_{0x}, c_{0x}) > (\leq, \alpha_x)$, we have $(\preccurlyeq_{0x}, -c_{0x}) < (<, -\alpha_x)$ and so $\lceil -Z_{0x} \rceil \leq (<, -\alpha_x)$. In each case, observe that we get $\max\{\lceil -Z_{0x} \rceil, (<, -\alpha_x)\}$ as the minimum value. □

**Lemma 4.** *Let* $Z$ *be a non-empty zone and let* $x, y$ *be variables none of them equal to* $x_0$*. Then, from among the regions* $R$ *that intersect* $Z$*, the least value of* $R_{xy}$ *is given by*

$$\begin{cases} (<, \infty) & \text{if } Z_{y0} < (\leq, -\alpha_y) \\ \max\{\lceil -Z_{yx} \rceil, \lceil -Z_{y0} \rceil - (\leq, \alpha_x)\} & \text{otherwise} \end{cases}$$

*Proof.* Let $G$ be the canonical distance graph representing the zone $Z$. We denote the weight of an edge $i \to j$ in $G$ by $(\preccurlyeq_{ij}, c_{ij})$. Recall that this means $Z_{ij} = (\preccurlyeq_{ij} , c_{ij})$. For clarity, for a valuation $v$, we write $v_x$ for $v(x)$.

We are interested in computing the smallest value of the $y-x$ constraint defining a region belonging to $Closure_\alpha(Z)$, that is, we need to find $\min\{[v]_{xy} \mid v \in Z\}$. Call this $\beta$. By definition of regions, we have for a valuation $v$:

$$[v]_{xy} = \begin{cases} (<, \infty) & \text{if } v_y > \alpha_y \\ \lceil (\leq, v_y - v_x) \rceil & \text{if } v_y \leq \alpha_y \text{ and } v_x \leq \alpha_x \\ (<, \lceil v_y \rceil - \alpha_x) & \text{if } v_y \leq \alpha_y \text{ and } v_x > \alpha_x \end{cases} \tag{2}$$

We now consider the first of the two cases from the statement of the lemma. Namely, $Z_{y0} < (\leq, -\alpha_y)$. This means that $0 - v_y \preccurlyeq_{y0} c_{y0}$ and $c_{y0} \leq -\alpha_y$;

moreover $\preccurlyeq_{y0}$ is the strict inequality if $c_{y0} = -\alpha_y$. In consequence, all valuations $v \in Z$, satisfy $v_y > \alpha_y$. Whence $\beta = (<, \infty)$.

We now consider the case when $Z_{y0} \geq (\leq, -\alpha_y)$. Let $G'$ be the graph in which the edge $0 \to y$ has weight $\min\{(\leq, \alpha_y), (\preccurlyeq_{0y}, c_{0y})\}$ and the rest of the edges are the same as that of $G$. This graph $G'$ represents the valuations of $Z$ that have $v_y \leq \alpha_y$: $[\![G']\!] = \{v \in Z \mid v_y \leq \alpha_y\}$. We show that this set is not empty. For this we check that $G'$ does not have negative cycles. Since $G$ does not have negative cycles, every negative cycle in $G'$ should include the newly modified edge $0 \to y$. Note that the shortest path value from $y$ to 0 does not change due to this modified edge. So the only possible negative cycle in $G'$ is $0 \to y \to 0$. But then we are considering the case when $Z_{y0} \geq (\leq, -\alpha_y)$, and so $Z_{y0} + (\leq, \alpha_y) \geq (\leq, 0)$. Hence this cycle cannot be negative either. In consequence all the cycles in $G'$ are positive and $[\![G']\!]$ is not empty.

To find $\beta$, it is sufficient to consider only the valuations in $[\![G']\!]$. As seen from Equation 2, among the valuations in $[\![G']\!]$, we need to differentiate between those with $v_x \leq \alpha_x$ and the ones with $v_x > \alpha_x$. We proceed as follows. We first compute $\min\{[v]_{xy} \mid v \in [\![G']\!] \text{ and } v_x \leq \alpha_x\}$. Call this $\beta_1$. Next, we compute $\min\{[v]_{xy} \mid v \in [\![G']\!] \text{ and } v_x > \alpha_x\}$ and set this as $\beta_2$. Our required value $\beta$ would then equal $\min\{\beta_1, \beta_2\}$.

To compute $\beta_1$, consider the following distance graph $G'_1$ which is obtained from $G'$ by just changing the edge $0 \to x$ to $\min\{(\leq, \alpha_x), (\preccurlyeq_{0x}, c_{0x})\}$ and keeping the remaining edges the same as in $G'$. The set of valuations $[\![G'_1]\!]$ equals $\{v \in [\![G']\!] \mid v_x \leq \alpha_x\}$. If $[\![G'_1]\!] = \emptyset$, we set $\beta_1$ to $(<, \infty)$ and proceed to calculate $\beta_2$. If not, we see that from Equation 2, for every $v \in [\![G'_1]\!]$, $[v]_{xy}$ is given by $\lceil (\leq, v_y - v_x) \rceil$. Let $(\preccurlyeq_1, w_1)$ be the shortest path from $y$ to $x$ in the graph $G'_1$. Then, we have for all $v \in [\![G'_1]\!]$, $v_x - v_y \preccurlyeq_1 w_1$, that is, $v_y - v_x \succcurlyeq_1 -w_1$. If $\preccurlyeq_1$ is $\leq$, then the least value of $[v]_{xy}$ would be $(\leq, -w_1)$ and if $\preccurlyeq_1$ is $<$, one can see that the least value of $[v]_{xy}$ is $(<, -w_1 + 1)$. This shows that $\beta_1 = \lceil (\preccurlyeq_1, -w_1) \rceil$. It now remains to calculate $(\preccurlyeq_1, w_1)$.

Recall that $G'_1$ has the same edges as in $G$ except possibly different edges $0 \to x$ and $0 \to y$. If the shortest path from $y$ to $x$ has changed in $G'_1$, then clearly it should be due to one of the above two edges. However note that the edge $0 \to y$ cannot belong to the shortest path from $y$ to $x$ since it would contain a cycle $y \to \ldots 0 \to y \to \ldots x$ that can be removed to give shorter path. Therefore, only the edge $0 \to x$ can potentially yield a shorter path: $y \to \ldots \to 0 \to x$. However, the shortest path from $y$ to 0 in $G'_1$ cannot change due to the added edges since that would form a cycle with 0 and we know that all cycles in $G'_1$ are positive. Therefore the shortest path from $y$ to 0 is the direct edge $y \to 0$, and the shortest path from $y$ to $x$ is the minimum of the direct edge $y \to x$ and the path $y \to 0 \to x$. We get: $(\preccurlyeq_1, w_1) = \min\{(\preccurlyeq_{yx}, c_{yx}), (\preccurlyeq_{y0}, c_{y0}) + (\leq, \alpha_x)\}$ which equals $\min\{Z_{yx}, Z_{y0} + (\leq, \alpha_x)\}$. Finally, from the argument in the above two paragraphs, we get:

$$\beta_1 = \begin{cases} (<, \infty) & \text{if } [\![G'_1]\!] = \emptyset \\ \lceil -Z_{yx} \rceil & \text{if } [\![G'_1]\!] \neq \emptyset \text{ and } Z_{yx} \leq Z_{y0} + (\leq, \alpha_x) \\ \lceil -Z_{y0} \rceil + (\leq, -\alpha_x) & \text{if } [\![G'_1]\!] \neq \emptyset \text{ and } Z_{yx} > Z_{y0} + (\leq, \alpha_x) \end{cases} \quad (3)$$

We now proceed to compute $\beta_2 = \min\{[v]_{xy} \mid v \in [\![G']\!] \text{ and } v_x > \alpha_x\}$. Let $G'_2$ be the graph which is obtained from $G'$ by modifying the edge $x \to 0$ to $\min\{Z_{x0}, (<, -\alpha_x)\}$ and keeping the rest of the edges the same as in $G'$. Clearly $[\![G'_2]\!] = \min\{v \in [\![G']\!] \mid v_x > \alpha_x\}$.

Again, if $[\![G'_2]\!]$ is empty, we set $\beta_2$ to $(<, \infty)$. Otherwise, from Equation 2, for each valuation $v \in [\![G'_2]\!]$, the value of $[v]_{xy}$ is given by $(<, \lceil v_y \rceil - \alpha_x)$. For the minimum value, we need the least value of $v_y$ from $v \in [\![G'_2]\!]$. Let $(\preccurlyeq_2, w_2)$ be the shortest path from $y$ to 0 in $G'_2$. Then, since $-v_y \preccurlyeq_2 w_2$, the least value of $\lceil v_y \rceil$ would be $-w_2$ if $\preccurlyeq_2 = \leq$ and equal to $\lceil -w_2 \rceil$ if $\preccurlyeq_2 = <$ and $\beta_2$ would respectively be $(<, -w_2 - \alpha_x)$ or $(<, -w_2 + 1 - \alpha_x)$. It now remains to calculate $(\preccurlyeq_2, w_2)$.

Recall that $G'_2$ is $G$ with $0 \to y$ and $x \to 0$ modified. The shortest path from $y$ to 0 cannot include the edge $0 \to y$ since it would need to contain a cycle, for the same reasons as in the $\beta_1$ case. So we get $(\preccurlyeq_2, w_2) = \min\{Z_{y0}, Z_{yx} + (<, -\alpha_x)\}$. If $Z_{y0} \leq Z_{yx} + (<, -\alpha_x)$, then we take $(\preccurlyeq_2, w_2)$ as $Z_{y0}$, otherwise we take it to be $Z_{yx} + (<, -\alpha_x)$. So, we get $\beta_2$ as the following:

$$\beta_2 = \begin{cases} (<, \infty) & \text{if } [\![G'_2]\!] = \emptyset \\ -Z_{yx} + (<, 1) & \text{if } [\![G'_2]\!] \neq \emptyset \text{ and } Z_{y0} \geq Z_{yx} + (<, -\alpha_x) \\ \lceil -Z_{y0} \rceil + (<, -\alpha_x) & \text{if } [\![G'_2]\!] \neq \emptyset \text{ and } Z_{y0} < Z_{yx} + (<, -\alpha_x) \end{cases} \quad (4)$$

However, we would like to write $\beta_2$ in terms of the cases used for $\beta_1$ in Equation 3 so that we can write $\beta$, which equals $\min\{\beta_1, \beta_2\}$, conveniently.

Let $\psi_1$ be the inequation: $Z_{yx} \leq Z_{y0} + (\leq, \alpha_x)$. From Equation 3, note that $\beta_1$ has been classified according to $\psi_1$ and $\neg\psi_1$ when $[\![G'_1]\!]$ is not empty. Similarly, let $\psi_2$ be the inequation: $Z_{y0} \geq Z_{yx} + (<, -\alpha_x)$. From Equation 4 we see that $\beta_2$ has been classified in terms of $\psi_2$ and $\neg\psi_2$ when $[\![G'_2]\!]$ is not empty. Notice the subtle difference between $\psi_1$ and $\psi_2$ in the weight component involving $\alpha_x$: in the former the inequality associated with $\alpha_x$ is $\leq$ and in the latter it is $<$. This necessitates a bit more of analysis before we can write $\beta_2$ in terms of $\psi_1$ and $\neg\psi_1$.

Suppose $\psi_1$ is true. So we have $(\preccurlyeq_{yx}, c_{yx}) \leq (\preccurlyeq_{y0}, c_{y0} + \alpha_x)$. This implies: $c_{yx} \leq c_{y0} + \alpha_x$. Therefore, $c_{y0} \geq c_{yx} - \alpha_x$. When $c_{y0} > c_{yx} - \alpha_x$, $\psi_2$ is clearly true. For the case when $c_{y0} = c_{yx} - \alpha_x$, note that in $\psi_2$ the right hand side is always of the form $(<, c_{yx} - \alpha_x)$, irrespective of the inequality in $Z_{yx}$ and so yet again, $\psi_2$ is true. We have thus shown that $\psi_1$ implies $\psi_2$.

Suppose $\neg\psi_1$ is true. We have $(\preccurlyeq_{yx}, c_{yx}) > (\preccurlyeq_{y0}, c_{y0} + \alpha_x)$. If $c_{yx} > c_{y0} + \alpha_x$, then clearly $c_{y0} < c_{yx} - \alpha_x$ implying that $\neg\psi_2$ holds. If $c_{yx} = c_{y0} + \alpha_x$, then we need to have $\preccurlyeq_{yx} = \leq$ and $\preccurlyeq_{y0} = <$. Although $\neg\psi_2$ does not hold now, we can safely take $\beta_2$ to be $\lceil -Z_{y0} \rceil + (<, -\alpha_x)$ as its value is in fact equal to $-Z_{yx} + (<, 1)$ in this case. Summarizing the above two paragraphs, we can rewrite $\beta_2$ as follows:

$$\beta_2 = \begin{cases} (<, \infty) & \text{if } [\![G'_2]\!] = \emptyset \\ -Z_{yx} + (<, 1) & \text{if } [\![G'_2]\!] \neq \emptyset \text{ and } Z_{xy} \leq Z_{y0} + (\leq, \alpha_x) \\ \lceil -Z_{y0} \rceil + (<, -\alpha_x) & \text{if } [\![G'_2]\!] \neq \emptyset \text{ and } Z_{xy} > Z_{y0} + (\leq, \alpha_x) \end{cases} \quad (5)$$

We are now in a position to determine $\beta$ as $\min\{\beta_1, \beta_2\}$. Recall that we are in the case where $Z_{y0} \leq (\leq, -\alpha_y)$ and we have established that $[\![G']\!]$ is non-empty. Now since $[\![G']\!] = [\![G'_1]\!] \cup [\![G'_2]\!]$ by construction, both of them cannot be simultaneously empty. Hence from Equations 3 and 5, we get $\beta$, the $\min\{\beta_1, \beta_2\}$ as:

$$\beta_= \begin{cases} \lceil -Z_{yx} \rceil & \text{if } Z_{xy} \leq Z_{y0} + (\leq, \alpha_x) \\ \lceil -Z_{y0} \rceil + (<, -\alpha_x) & \text{if } Z_{xy} > Z_{y0} + (\leq, \alpha_x) \end{cases} \quad (6)$$

There remains one last reasoning. To prove the lemma, we need to show that $\beta = \max\{\lceil -Z_{yx} \rceil, \lceil -Z_{y0} \rceil + (<, -\alpha_x)\}$. For this it is enough to show the following two implications:

$$Z_{yx} \leq Z_{y0} + (\leq, \alpha_x) \Rightarrow \lceil -Z_{yx} \rceil \geq \lceil -Z_{y0} \rceil + (<, -\alpha_x)$$
$$Z_{yx} > Z_{y0} + (\leq, \alpha_x) \Rightarrow \lceil -Z_{yx} \rceil \leq \lceil -Z_{y0} \rceil + (<, -\alpha_x)$$

We prove only the first implication. The second follows in a similar fashion. Let us consider the notation $(\preccurlyeq_{yx}, c_{yx})$ and $(\preccurlyeq_{y0}, c_{y0})$ for $Z_{yx}$ and $Z_{y0}$ respectively. So we have:

$$\begin{aligned} & (\preccurlyeq_{yx}, c_{yx}) \leq (\preccurlyeq_{y0}, c_{y0}) + (\leq, \alpha_x) \\ \Rightarrow\ & (\preccurlyeq_{yx}, c_{yx}) \leq (\preccurlyeq_{y0}, c_{y0} + \alpha_x) \end{aligned}$$

If the constant $c_{yx} < c_{y0} + \alpha_x$, then $-c_{yx} > -c_{y0} - \alpha_x$ and we clearly get that $\lceil -Z_{yx} \rceil \geq \lceil -Z_{y0} \rceil + (<, -\alpha_x)$. If the constant $c_{yx} = c_{y0} + \alpha_x$ and if $\preccurlyeq_{y0} = \leq$, then the required inequation is trivially true; if $\preccurlyeq_{y0} = <$, it implies that $\preccurlyeq_{yx} = <$ too and clearly $\lceil (<, -c_{yx}) \rceil$ equals $\lceil (<, -c_{y0}) \rceil + (<, -\alpha_x)$. □

►**Theorem 2.** Let $Z, Z'$ be zones. Then, $Z \nsubseteq \mathit{Closure}_\alpha(Z')$ iff there exist variables $x$, $y$ such that one of the following conditions hold:

1. $Z'_{0x} < Z_{0x}$ and $Z'_{0x} \leq (\alpha_x, \leq)$, or
2. $Z'_{x0} < Z_{x0}$ and $Z_{x0} \geq (-\alpha_x, \leq)$, or
3. $Z_{x0} \geq (-\alpha_x, \leq)$ and $Z'_{xy} < Z_{xy}$ and $Z'_{xy} \leq (\alpha_y, \leq) + \lfloor Z_{x0} \rfloor$

*Proof.* By definition of the *Closure* abstraction, $Z \nsubseteq \mathit{Closure}_\alpha(Z')$ iff there exists a region $R$ that intersects $Z$ but does not intersect $Z'$. Therefore, from Proposition 2, we need an $R$ that intersects $Z$ and satisfies $Z'_{yx} + R_{xy} < (\leq, 0)$ for some

variables $x, y$. This is equivalent to saying that for the least value of $R_{xy}$ that can be obtained from the zone $Z$, we have $Z'_{yx} + R_{xy} < (0, \leq)$. Depending on if $x$ is $x_0$ or $y$ is $x_0$ or both $x$ and $y$ are not $x_0$ we get the following three conditions that correspond to the three conditions given in the theorem.

Case 1: $Z'_{0x} + R_{x0} < (\leq, 0)$
From Lemma 3, the minimum value of $R_{x0}$ from among the regions intersecting $Z$ is given by $\max\{\lceil -Z_{0x} \rceil, (<, -\alpha_x)\}$. So we have:

$$\begin{aligned} & Z'_{0x} + \max\{\lceil -Z_{0x} \rceil, (<, -\alpha_x)\} < (\leq, 0) \\ & \quad \Rightarrow \; Z'_{0x} + \lceil -Z_{0x} \rceil < (\leq, 0) \;\text{ and }\; Z'_{0x} + (<, -\alpha_x) < (\leq, 0) \\ & \quad \Rightarrow \; Z'_{0x} < Z_{0x} \;\text{ and }\; Z'_{0x} \leq (\leq, \alpha_x) \end{aligned}$$

This gives Condition 1 of the theorem.

Case 2: $Z'_{x0} + R_{0x} < (\leq, 0)$
From Lemma 3, the minimum value of $R_{0x}$ is $(<, \infty)$ if $-Z_{x0} > (\leq, \alpha_x)$ and hence it cannot be part of a negative cycle. The edge $R_{0x}$ can yield a negative cycle only when $-Z_{x0} \leq (\leq, \alpha_x)$, in which case the least value of $R_{0x}$ is given by $\lceil -Z_{x0} \rceil$. So we have $Z'_{x0} + \lceil -Z_{x0} \rceil < (\leq, 0)$ which translates to $Z'_{x0} < Z_{x0}$. Therefore, this case is equivalent to saying $Z_{x0} \geq (\leq, -\alpha_x)$ and $Z'_{x0} < Z_{x0}$ which gives Condition 2 of the theorem.

Case 3: $Z'_{yx} + R_{xy} < (\leq, 0)$ From Lemma 4, we get that the minimum value of the edge $R_{xy}$ is $(<, \infty)$ if $-Z_{y0} > (\leq, \alpha_y)$. Similar to the case above, $R_{xy}$ cannot be part of a negative cycle if $-Z_{y0} > (\leq, \alpha_y)$. So we need to first check if $-Z_{y0} \leq (\leq, \alpha_y)$, that is, if $Z_{y0} \geq (\leq, -\alpha_y)$. Now, from Lemma 4, the minimum value of $R_{xy}$ is given by the $\max\{\lceil -Z_{yx} \rceil, \lceil -Z_{y0} \rceil - (\leq, \alpha_x)\}$. We get:

$$\begin{aligned} & Z'_{yx} + \max\{\lceil -Z_{yx} \rceil, \lceil -Z_{y0} \rceil - (\leq, \alpha_x)\} < (\leq, 0) \\ & \quad \Rightarrow Z'_{yx} + \lceil -Z_{yx} \rceil < (\leq, 0) \text{ and } Z'_{yx} + \lceil -Z_{y0} \rceil - (\leq, \alpha_x) < (\leq, 0) \\ & \quad \Rightarrow Z'_{yx} < Z_{yx} \text{ and } Z'_{yx} + \lceil -Z_{y0} \rceil - (\leq, -\alpha_x) < (\leq, 0) \end{aligned}$$

Let us look at the second inequality: $Z'_{yx} + \lceil -Z_{y0} \rceil - (\leq, -\alpha_x) < (\leq, 0)$. If $Z_{y0}$ is of the form $(\leq, c)$ with $c$ an integer, then $-Z_{y0} = (\leq, -c)$ and $\lceil -Z_{y0} \rceil$ is the same: $(\leq, -c)$. So we get:

$$\begin{aligned} Z'_{yx} + (\leq, -c) + (<, -\alpha_x) &< (\leq, 0) \\ \Leftrightarrow Z'_{yx} + (<, -c - \alpha_x) &< (\leq, 0) \\ \Leftrightarrow Z'_{yx} &\leq (\leq, c + \alpha_x) \end{aligned}$$

When $Z_{y0} = (<, c)$, then $\lceil -Z_{y0} \rceil = (<, -c + 1)$ and we get:

$$\begin{aligned} Z'_{yx} + (<, -c+1) + (<, -\alpha_x) &< (\leq, 0) \\ \Leftrightarrow Z'_{yx} + (<, -c+1-\alpha_x) &< (\leq, 0) \\ \Leftrightarrow Z'_{yx} &\leq (\leq, c-1+\alpha_x) \end{aligned}$$

This gives Condition 3 of the Theorem (symmetric in $x$ and $y$).

□

### A.2 Handling LU-approximation

Recall that for a zone $Z$, we denote by $Z^+$ the zone $Extra^+_{LU}(Z)$. Also note that $Z^+$ is not necessarily in canonical form.

**Proposition 3.** *Let $R$ be a region and $Z$ be a zone. Then, $R \cup Z^+$ is empty iff there exist variables $x, y$ such that $Z^+_{yx} + R_{xy} < (\leq, 0)$.*

*Proof.* Let $G_R$ be the canonical graph representing $R$ and let $G_Z$ be the canonical distance graph representing $Z$. Let $G_{Z^+}$ be the graph that representing $Z^+$. By definition, $G_{Z^+}$ is obtained from $G_Z$ by changing some edges to $(<, \infty)$ and some edges incident on $x_0$ to $(<, -U(x))$. Also, note that $G_{Z^+}$ is not necessarily in canonical form.

From Proposition 1, $R \cup Z^+$ is empty iff $\min(G_R, G_{Z^+})$ has a negative cycle. 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 2 we get a negative cycle of the required form 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 $\infty$. 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. □

**Efficient inclusion testing for LU approximations** 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 Proposition 3, the check $Z \subseteq Closure^+_{LU}(Z')$ can be reduced to an edge by edge comparison with every region intersecting $Z$. Lemmas 3 and 4 give the least value of the edge $R_{xy}$ for 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 Theorem 2 with $Z'$ replaced by $Z'^+$.

**Theorem 5.** *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. $Z'^+_{0x} < Z_{0x}$ *and* $Z'^+_{0x} \leq (\alpha_x, \leq)$, *or*
2. $Z'^+_{x0} < Z_{x0}$ *and* $Z_{x0} \geq (-\alpha_x, \leq)$, *or*
3. $Z_{x0} \geq (-\alpha_x, \leq)$ *and* $Z'^+_{xy} < Z_{xy}$ *and* $Z'^+_{xy} \leq (\alpha_y, \leq) + Z_{x0}$

# B Proofs from Section 4

## B.1 Correctness of the algorithm with *Closure* approximation

Here we show the proof of

**Theorem 4** 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 5 ([7]).** *For every zone $Z$, transition $t$ and a bound function $\alpha$:*

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

**Lemma 6.** *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 $(q, Z', \alpha')$ 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 $(q, Z_L, \alpha)$ 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 $(q, Z_L, \alpha)$ 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 5, $Z' \subseteq Closure_\alpha(Post(Z_L, t))$, that is $Z' \subseteq Closure_\alpha(Z'_L)$. We now need to check if we can replace $\alpha$ with $\alpha'$. 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 $\alpha$ or $\alpha'$ 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 then by definition we know that there exists a non tentative node $(q', Z''_L, \alpha'')$ such that $\alpha'' = \alpha'$ and $Z'_L \subseteq Closure_{\alpha'}(Z''_L)$. Thus $Z' \subseteq Closure_{\alpha'}(Z''_L)$. In this case $(q', Z''_L, \alpha'')$ is the node corresponding to $(q', Z')$.

□

### B.2 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 2 (LU-preorder).** *Fix integers $L$ and $U$. Let $\nu$ and $\nu'$ 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 3 ($\mathfrak{a}_{\preccurlyeq LU}$, 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 7.** *Let $q$ be a state of $\mathcal{A}$ and $t = (q, g, R, q_1)$ a transition. Assume that for a clock $x$: $L(x) \geq c$ for all $c$ such that $x \geq c$ or $x > c$ occurs in $g$; and $U(x) \geq d$ for all $d$ such that $x \leq d$ or $x < d$ occurs in $g$. Let $\nu$ and $\nu'$ be valuations such that $\nu' \preccurlyeq_{LU} \nu$. Then, $(q, \nu) \xrightarrow{\delta, t} (q_1, \nu_1)$ implies that there exists a delay $\delta'$ and a valuation $\nu'_1$ 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 8.** *For all zones $Z$,*

$$Extra^+_{LU}(Z) \text{ is a zone} \tag{7}$$
$$Extra^+_{LU}(Z) \subseteq \mathfrak{a}_{\preccurlyeq LU}(Z) \tag{8}$$

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

**Theorem 6.** *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 9.** *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 7, $\nu' \xrightarrow{t} \nu'_1$ 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 from the next lemma and from the following invariant on the nodes of the tree that is computed. For every node $n = (q, Z, L, U)$:

1. if $n$ is *nottentative*, then $L, U$ are respectively the maximum of the $L_s, U_s$ from all successor nodes $(q_s, Z_s, L_s, U_s)$ of $n$ (taking into account guards and clock resets, even if $Z_s$ is empty);
2. if $n$ is *tentative* with respect to $(q', Z', L', U')$, then $L$ and $U$ are equal to $L'$ and $U'$ respectively.

**Lemma 10.** *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 7, $t$ 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 9, $Z' \subseteq \mathfrak{a}_{\preccurlyeq_{L_1U_1}}(Post(Z_1,t))$. 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 $(q, Z_1, L_1U_1)$.

□