http://www.lsv.ens–cachan.fr/Publis/
In *Proc. 9th Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems (TACAS'2003) Warsaw, Poland, Apr. 2003, volume 2619 of Lecture Notes in Computer Science, pages 254–277. Springer, 2003.*

Static Guard Analysis in Timed Automata Verification

Gerd Behrmann[1], Patricia Bouyer[2*], Emmanuel Fleury[1], Kim G. Larsen[1]

[1] BRICS[***], Aalborg University Denmark
Fredrik Bajers Vej 7, 9220 Aalborg Ø – Denmark
Email: {behrmann, fleury, kgl}@cs.auc.dk

[2] LSV, CNRS UMR 8643, ENS de Cachan
61, av. du Prés. Wilson, 94235 Cachan Cedex – France
Email: bouyer@lsv.ens-cachan.fr

Abstract. By definition Timed Automata have an infinite state-space, thus for verification purposes, an exact finite abstraction is required. We propose a *location-based finite zone abstraction*, which computes an abstraction based on the *relevant* guards for a particular state of the model (as opposed to *all* guards). We show that the location-based zone abstraction is sound and complete with respect to location reachability; that it generalises *active-clock reduction*, in the sense that an inactive clock has no relevant guards at all; that it *enlarges* the class of timed automata, that can be verified. We generalise the new abstraction to the case of *networks* of timed automata, and experimentally demonstrate a potentially exponential speedup compared to the usual abstraction.

1 Introduction

Since their introduction by Alur and Dill [3], timed automata have become one of the most well-studied and well-established models for real-time systems. By their definition timed automata models describe infinite state-spaces. Thus, to enable algorithmic verification, exact finite abstractions are required. Here, the original region-graph construction of Alur and Dill provides a "universal" such abstraction. However, whereas indispensable as a key to decidability for several timed automata related decision problems, the enormous size of the region-graph construction makes it highly impractical for tool-implementation. In fact, most real-time verification tools (*e.g.* UPPAAL [20, 6], KRONOS [12] and CMC [19]) apply abstractions based on so-called zones and in a highly model-dependent manner in order to be as coarse (and hence small and efficient) as possible.

To insure finiteness, it is essential for the (region- as well as zone-based) abstractions to take into account the maximum constants to which clocks are compared. In particular, the abstractions identify states which are identical except for the values of clocks exceeding the (relevant) maximum constants. Obviously, the smaller we choose

---

* Supported by a BRICS grant. The work has been mainly carried out while the author had a post-doctoral position at Aalborg University.
*** Basic Research in Computer Science (`www.brics.dk`), funded by the Danish National Research Foundation.

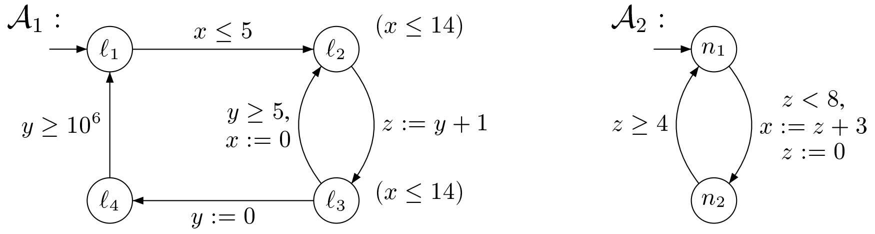


Fig. 1. *Network of Timed Automaton* $\mathcal{A}_1 \parallel \mathcal{A}_2$.

these maximum constants, the coarse the abstraction will be. So far, the maximum constants have been determined by a global examination of *all* guards in the model. In this paper we propose a coarser *location-based* abstraction, using location-dependent (and smaller) maximum constants based on the *relevant* guards for a particular state of the timed automata.

Consider the timed automata network in Fig. 1. Here, global examination identifies 10^6 as a maximum constant for y. However, the guard $y \geq 10^6$ is clearly irrelevant in ℓ_2 and ℓ_3 as any path from these locations to the guard must pass through a reset of y. Thus, it should be possible to choose valid maximum constants, $\max_2$ and $\max_3$, for y in ℓ_2 and ℓ_3 substantially smaller than 10^6. Obviously, $\max_2, \max_3 \geq 5$, due to the relevant guard $y \geq 5$. However, 5 may not necessarily be a valid maximum constant for y in ℓ_2 and ℓ_3: the combination of the update $z := y + 1$ with the guard $z < 8$ in $\mathcal{A}_2$ suggests the relevance of the derived guard $y < 8 - 1$. Thus $\max_2, \max_3 \geq 7$. In fact realizing that also the update $x := z + 3$ and the invariant $x \leq 14$ are relevant in ℓ_2 and ℓ_3 yields $\max_2, \max_3 = 10$ as smallest valid maximal constants for y in ℓ_2 and ℓ_3.

In this paper, we will offer efficient methods for identification of location-dependent maximal constants based on a static analysis for determining and combining *relevant* guards and updates for a given location. In particular, we prove that the maximal constants determined are valid in the sense that the resulting relevant guard abstraction is exact with respect to location reachability. Furthermore, to keep the static analysis as light as possible, we offer a computational method allowing valid location-dependent maximal constants of a network to be derived from the component automata. We further experimentally demonstrate that the use of location-dependent maximal constants in abstractions provides a potential exponential speedup[1].

The closest related work to our static analysis method for relevant guards and updates is the *active-clock reduction* technique for timed automata presented in [15]. In fact, active-clock reduction is a special case of our relevant guard abstraction, in the sense that an inactive clock has no relevant guards at all. Also related is the work in [13] on using precomputed influence information.

2 The Model and its Semantics

The model used in this paper is that of networks of timed automata. Let X be a set of non-negative real-valued variables called *clocks*, and $\mathcal{A}\mathrm{ct}$ a set of actions and co-actions (denoted $a!$ and $a?$) and the non-synchronising action (denoted τ). The set of *guards*, denoted by $\mathcal{G}(X)$, and the set of updates, denoted by $\mathcal{U}(X)$, are generated by

[1] *E.g.* an exponential speedup is obtained for the reachability of the network in Fig. 1.

the grammars

$$g ::= x \bowtie c \mid g_1 \wedge g_2 \qquad \mathsf{up} ::= x := c \mid x := y + c \mid \mathsf{up}_1 \wedge \mathsf{up}_2 \ ,$$

where $x, y \in X$, $c \in \mathbb{N}$, $\bowtie \in \{<, \leq, =, \geq, >\}$, $\mathsf{base}(\mathsf{up}_1) \cap \mathsf{base}(\mathsf{up}_2) = \emptyset$, and base is a function such that $\mathsf{base}(x := c) = \mathsf{base}(x := y + c) = \{x\}$ and $\mathsf{base}(\mathsf{up}_1 \wedge \mathsf{up}_2) = \mathsf{base}(\mathsf{up}_1) \cup \mathsf{base}(\mathsf{up}_2)$. For the static analysis to follow we also define the function used such that $\mathsf{used}(x \bowtie c) = \{x\}$, $\mathsf{used}(x := y + c) = \{y\}$ and $\mathsf{used}(\varphi_1 \wedge \varphi_2) = \mathsf{used}(\varphi_1) \cup \mathsf{used}(\varphi_2)$, where φ_i is either a guard or an update.

Definition 1. *A* timed automaton *over $(\mathcal{A}\mathrm{ct}, X)$ is a tuple (L, ℓ^0, I, E), where L is a set of locations, $\ell^0 \in L$ is an initial location, $I : L \to \mathcal{G}(X)$ assigns invariants to locations, and E is a set of edges such that $E \subseteq L \times \mathcal{G}(X) \times \mathcal{A}\mathrm{ct} \times \mathcal{U}(X) \times L$. A* network of timed automata *$\mathcal{A}_1 \parallel \cdots \parallel \mathcal{A}_n$ over $(\mathcal{A}\mathrm{ct}, X)$ is defined as the parallel composition of n timed automata over $(\mathcal{A}\mathrm{ct}, X)$. Let $\mathcal{A}_i = (L_i, \ell_i^0, I_i, E_i)$ for $1 \leq i \leq n$. We write $\ell \xrightarrow{g,a,u}_i \ell'$ iff $(\ell, g, a, u, \ell') \in E_i$ and $I(\boldsymbol{\ell}) = \bigwedge_{1 \leq i \leq n} I_i(\ell_i)$, where $\boldsymbol{\ell}$ is our standard notation for a vector $\boldsymbol{\ell} = (\ell_1, \ldots, \ell_n)$.*

The semantics of a network of timed automata is defined in terms of a transition system over states of the network. Intuitively, there are three kinds of transitions: delay transitions, internal transitions, and synchronisations. Before formally stating the semantics, we introduce a few definitions. A clock valuation $\sigma \in \mathbb{R}_{\geq 0}^X$ is a function which assigns values to clocks. If $d \in \mathbb{R}_{>0}$ is a delay, then $\sigma + d$ denotes the clock valuation such that for each clock x, $(\sigma + d)(x) = \sigma(x) + d$. If $\mathsf{up} \in \mathcal{U}(X)$ is an update, then $\mathsf{up}(\sigma)$ denotes the valuation which maps x to c if $x := c$ is in up, maps x to $\sigma(y) + c$ if $x := y + c$ is in up, and agrees with σ in all other cases. We write $\sigma \models g$ if and only if the clock valuation σ satisfies the guard g (defined in the natural way).

Definition 2. *The semantics of a network of timed automata $\mathcal{A}_1 \parallel \cdots \parallel \mathcal{A}_n$ is defined by a transition system $(S, s_0, \longrightarrow)$, where $S = (L_1 \times \cdots \times L_n) \times \mathbb{R}_{\geq 0}^X$ is the set of states, $s_0 = (\boldsymbol{\ell}^0, \sigma^0)$ is the initial state, $\forall x \in X$: $\sigma^0(x) = 0$, and $\longrightarrow \subseteq S \times S$ is the set of transitions defined by:*

$$\frac{\forall 0 \leq d' \leq d : \ \sigma + d' \models I(\boldsymbol{\ell})}{(\boldsymbol{\ell}, \sigma) \xrightarrow{d} (\boldsymbol{\ell}, \sigma + d)} \qquad \textit{if } d \in \mathbb{R}_{>0}$$

$$\frac{\ell_i \xrightarrow{g,\tau,\mathsf{up}}_i \ell_i' \qquad \sigma \models g \qquad \mathsf{up}(\sigma) \models I(\boldsymbol{\ell}[\ell_i'/\ell_i])}{(\boldsymbol{\ell}, \sigma) \xrightarrow{\tau} (\boldsymbol{\ell}[\ell_i'/\ell_i], \mathsf{up}(\sigma))}$$

$$\frac{\begin{array}{l}\ell_i \xrightarrow{g_i,a!,\mathsf{up}_i}_i \ell_i' \\ \ell_j \xrightarrow{g_j,a?,\mathsf{up}_j}_j \ell_j'\end{array} \qquad \sigma \models g_i \wedge g_j \qquad \sigma' \models I(\boldsymbol{\ell}')}{(\boldsymbol{\ell}, \sigma) \xrightarrow{\tau} (\boldsymbol{\ell}', \sigma')} \qquad \begin{array}{c}\textit{if } i \neq j, \\ \boldsymbol{\ell}' = \boldsymbol{\ell}[\ell_i'/\ell_i, \ell_j'/\ell_j], \\ \sigma' = (\mathsf{up}_i \wedge \mathsf{up}_j)(\sigma)\end{array}$$

In contrast to UPPAAL and KRONOS, we allow updates of the form $x := y + c$. In the next section, we restrict ourselves to individual timed automata (*i.e.* consider only τ actions). Extensions to networks of timed automata will be considered in section 6.

3 Forward Analysis

A standard forward breadth-first or depth-first state-space exploration based directly on the transition relation '$\longrightarrow$' defined in section 2 is unlikely to terminate, since the state space is uncountably infinite. The classical approach used to prove decidability of the reachability problem for timed automata involves the construction of a region graph [3]. Intuitively, regions are sets of clock valuations indistinguishable by arbitrary guards and behaving identical under delay and update operations. In practice, tools like UPPAAL and KRONOS use zones rather than regions to build a coarser representation of the state-space. Zones are sets of clock valuations definable by conjunctions of constraints of the forms $x \bowtie c$ and $x - y \bowtie c$, where x and y are clocks and c is an integer. Using zones makes the state-space countable, but not finite. In order to make it finite, an abstraction of the state-space is needed. In the following, we will use Z to refer to zones and W to arbitrary sets of valuations.

In this section we will recall the symbolic semantics of timed automata, the DBM representation of zones, and the DBM based abstraction used in order to obtain a finite state-space.

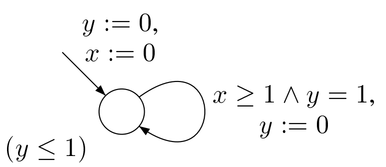
Symbolic Semantics. For universality, the definition of the symbolic semantics uses arbitrary sets of clock valuations, rather than zones.

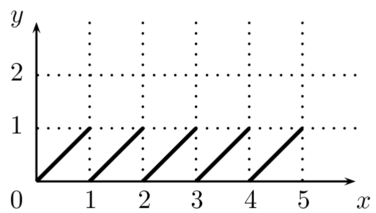
Definition 3. *The symbolic semantics of a timed automaton $\mathcal{A} = (L, \ell^0, I, E)$ is based on the abstract transition system $(S, s_0, \Longrightarrow)$, where $S = L \times \mathcal{P}(\mathbb{R}^X_{\geq 0})^2$, $s_0 = (\ell^0, \{\sigma^0\})$, and '$\Longrightarrow$' is defined by the following two rules:*

DELAY: $(\ell, W) \Longrightarrow (\ell, W')$,
where $W' = \big\{\sigma + d \mid \sigma \in W \wedge d \geq 0 \wedge \forall\, 0 \leq d' \leq d : \sigma + d' \models I(\ell)\big\}$

ACTION: $(\ell, W) \Longrightarrow (\ell', W')$*if there exists a transition* $\ell \xrightarrow{g,a,\mathsf{up}} \ell'$ *in* $\mathcal{A}$,
such that $W' = \big\{\mathsf{up}(\sigma) \mid \sigma \in W \wedge \sigma \models g \wedge \mathsf{up}(\sigma) \models I(\ell')\big\}$.

Example 1. The timed automaton in the following figure illustrates that although the symbolic semantics results in a countable state-space, it is not necessarily finite. Whenever one time unit has passed, the loop will be taken and y will be reset to zero. However x keeps growing thus resulting in an infinite state-space.





To obtain a finite graph, we suggest to apply some abstraction $\mathfrak{a} : \mathcal{P}(\mathbb{R}^X_{\geq 0}) \hookrightarrow \mathcal{P}(\mathbb{R}^X_{\geq 0})$, such that $W \subseteq \mathfrak{a}(W)$. The abstract transition system '$\Longrightarrow_{\mathfrak{a}}$' will then be given by the following induction rules:

[2] $\mathcal{P}(A)$ denotes the power set of A.

$$\frac{(\ell, W) \Longrightarrow (\ell', W')}{(\ell, W) \Longrightarrow_{\mathfrak{a}} \big(\ell', \mathfrak{a}(W')\big)} \qquad \text{if } W = \mathfrak{a}(W)$$

A simple way to assure that the reachability graph induced by '$\Longrightarrow_{\mathfrak{a}}$' is finite is to establish that there is only finitely many abstractions of sets of valuations, that is, the set $\{\mathfrak{a}(W) \mid \mathfrak{a} \text{ defined on } W\}$ is finite. In this case $\mathfrak{a}$ is said to be a *finite abstraction*. Moreover, $\Longrightarrow_{\mathfrak{a}}$ is said *sound* and *complete* whenever:

SOUND: $(\ell^0, \{\sigma^0\}) \Longrightarrow_{\mathfrak{a}}^* (\ell, W)$ implies $\exists \sigma \in W : (\ell^0, \sigma^0) \longrightarrow^* (\ell, \sigma)$
COMPLETE: $(\ell^0, \sigma^0) \longrightarrow^* (\ell, \sigma)$ implies $\exists W : \sigma \in W \wedge (\ell^0, \{\sigma^0\}) \Longrightarrow_{\mathfrak{a}}^* (\ell, W)$

Completeness follows trivially from the definition of abstraction. Of course, if $\mathfrak{a}$ and $\mathfrak{b}$ are two abstractions such that for any set of valuations W, $\mathfrak{a}(W) \subseteq \mathfrak{b}(W)$, we prefer to use abstraction $\mathfrak{b}$, because the graph induced by it, is *a priori* smaller than the one induced by $\mathfrak{a}$. Our aim is thus to propose an abstraction which is finite and which induces a sound abstract transition system. We also require that this abstraction is *effective*, in the sense that it can be efficiently computed.

Zones. A First step in finding an effective abstraction is realizing that W will always be a zone for any $(\ell^0, \{\sigma^0\}) \Longrightarrow^* (\ell, W)$. Zones can be represented using *Difference Bound Matrices* (DBM). We will briefly recall the definition of DBMs, but refer to [16, 14, 5, 8] for more details.

A DBM is a square matrix $M = \langle m_{i,j}, \prec_{i,j} \rangle_{0 \le i,j \le n}$ such that $m_{i,j} \in \mathbb{Z}$ and $\prec_{i,j} \in \{<, \le\}$ or $m_{i,j} = \infty$ and $\prec_{i,j} = <$. M represents the zone $[\![M]\!]$ which is defined by $[\![M]\!] = \{\sigma \mid \forall 0 \le i,j \le n : \sigma(x_i) - \sigma(x_j) \prec_{i,j} m_{i,j}\}$, where $\{x_i \mid 1 \le i \le n\}$ is the set of clocks, and x_0 is a clock which is always 0, (*i.e.* $\forall \sigma$: $\sigma(x_0) = 0$). DBMs are not a canonical representation of zones, but a normal form can be computed by considering the DBM as an adjacency matrix of a weighted directed graph and computing all shortest paths. In particular, if $M = \langle m_{i,j}, \prec_{i,j} \rangle_{0 \le i,j \le n}$ is a DBM in normal form, then it satisfies the *triangular inequality*, that is, for every $0 \le i,j,k \le n$, we have that $(m_{i,j}, \prec_{i,j}) \le (m_{i,k}, \prec_{i,k}) + (m_{k,j}, \prec_{k,j})$ where comparisons and additions are defined in a natural way (see [8]). All operations needed to compute '$\Longrightarrow$' can be implemented by manipulating the DBM.

The 'Maximum Constants' Abstraction. The abstraction currently in use in model-checkers, is based on the idea that the automaton is only sensitive to changes on a clock if its value is below a certain constant. That is, for each clock there is a maximum constant and once the value of a clock has passed this constant, its exact value is no longer relevant – only the fact that it is larger than the maximum constant matters. Transforming a DBM to reflect this idea is often referred to as *extrapolation* or *normalisation* [15].

Definition 4 (Extrapolation). *Given a set of clocks $\{x_i \mid 1 \le i \le n\}$, a maximum constant k_i for each clock x_i, and a DBM $M = \langle m_{i,j}, \prec_{i,j} \rangle_{0 \le i,j \le n}$, the* extrapolation *of M is $M' = \langle m'_{i,j}, \prec'_{i,j} \rangle_{0 \leq i,j \leq n}$ such that:*

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

Let $\Lambda_{\boldsymbol{k}}$ denote the extrapolation operator corresponding to the tuple of constants $\boldsymbol{k} = (k_1, \ldots, k_n)$. In an abuse of notation, we use $\Lambda_{\boldsymbol{k}}$ as an abstraction function.

The actual choice of constants $\boldsymbol{k}$ is based on the guards, invariants, and updates of the system. For the simple subset of timed automata without guards on clock differences and with clock updates limited to reset to a constant, the constant k_i for clock x_i is simply the maximum constant in any guard or invariant that x_i is ever compared to. In the general case, finding $\boldsymbol{k}$ requires solving a system of simple linear Diophantine inequalities [10, 8], having one variable, p_x, for each clock x. The inequalities are on the form $p_x \leq p_y + c$ whenever there is an update $x := y + c$ in the automaton, or $p_x \geq d$ whenever there is a guard $x \bowtie d$ in the automaton. The system of inequalities has the nice property that whenever it has a solution $\boldsymbol{k}$, then it is safe to use $\Lambda_{\boldsymbol{k}}$ as an abstraction. The graph induced by the transition relation '$\Longrightarrow_{\Lambda_{\boldsymbol{k}}}$' is obviously finite and has an effective implementation. Its soundness has been proved in [9].

4 Location-Dependent Abstractions

The symbolic transition relation '$\Longrightarrow_{\Lambda_{\boldsymbol{k}}}$' defined in the previous section is based on a location-independent abstraction of zones. That is, for a given symbolic state (ℓ, Z) the abstraction applied to Z is independent of ℓ, but merely uses global information of the constants to which clocks are compared throughout the automaton. However, for a given location not all of these comparisons may be of relevance: if all paths in the automaton from a location ℓ to a guard $x \bowtie c$ passes through an update of x, then intuitively the guard is not relevant or active in ℓ.

To illustrate this, consider the timed automaton in Fig. 2. Here the guard $y \geq 10^6$ is irrelevant in ℓ_2 as any path from ℓ_2 to the guard necessarily must pass through the reset of y. In contrast the guard $y \geq 5$ is relevant in ℓ_2. It is easy to see that for any pair $(\ell_2, v_x, v_y), (\ell_2, v_x, w_y)$, where $v_y, w_y > 5$, the set of reachable locations is identical, stipulating the irrelevance of the guard $y \geq 10^6$. Now using '$\Longrightarrow_{\Lambda_{\boldsymbol{k}}}$', observing that

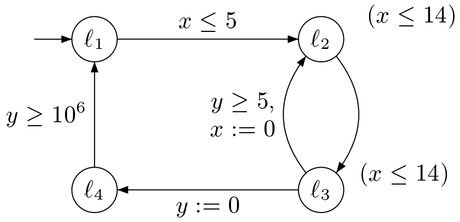


Fig. 2. Timed Automaton A.

$k_x = 5$ and $k_y = 10^6$, iteration of the cycle between ℓ_2 and ℓ_3 leads to the following sequence of symbolic states for the timed automaton in Fig. 2:

$$
\begin{array}{l}
(\ell_1, x = 0 \wedge y = 0) \\
\Longrightarrow_{\Lambda_k} (\ell_1, x = y) \Longrightarrow_{\Lambda_k} (\ell_2, 5 \leq x, y \leq 14 \,\wedge\, x = y) \\
\Longrightarrow^2_{\Lambda_k} (\ell_2, x \in [0, 14] \,\wedge\, y \in [5, 28] \,\wedge\, y - x \in [5, 14]) \\
\ldots\ldots \\
\Longrightarrow^n_{\Lambda_k} (\ell_2, x \in [0, 14] \,\wedge\, y \in [5, 14n] \,\wedge\, y - x \in [5, 14n - 14]) \\
\ldots\ldots \\
\Longrightarrow^*_{\Lambda_k} (\ell_2, 0 \leq x \leq 14 \,\wedge\, 5 \leq y \,\wedge\, 5 \leq y - x);
\end{array}
$$

where $n \leq \lceil 10^6/14 \rceil + 1$. We observe that the symbolic states in this sequence are strictly increasing, and hence termination of a forward symbolic exploration using '$\Longrightarrow_{\Lambda_k}$' occur only extremely slowly. Obviously, we look for a more abstract '$\Longrightarrow_{\Lambda_k}$' which will take into account the irrelevance of the guard $y \geq 10^6$ in location ℓ_2. In the remainder of this section we suggest three such abstractions, all ignoring location-irrelevant guards and sound and complete *w.r.t.* location reachability, but differing in efficiency of representation. Let us consider for the rest of this section a given timed automaton $\mathcal{A} = (L, \ell^0, I, E)$.

Location-based Bisimulation. In Fig. 2, states of the form (ℓ_2, v_x, v_y) with $v_y > 5$ and $v_x \leq 14$, do not only agree on location-reachability but are *location-bisimilar*. We define *location-based bisimularity*, $\equiv$, as the largest relation *s.t.*:

- $(\ell, \sigma) \equiv (\ell', \sigma')$ implies $\ell = \ell'$,
- if $(\ell_1, \sigma_1) \equiv (\ell_2, \sigma_2)$ and $(\ell_1, \sigma_1) \xrightarrow{\tau} (\ell_1', \sigma_1')$,
  then $\exists \sigma_2'$, such that $(\ell_2, \sigma_2) \xrightarrow{\tau} (\ell_2', \sigma_2')$ and $(\ell_1', \sigma_1') \equiv (\ell_2', \sigma_2')$,
- if $(\ell_1, \sigma_1) \equiv (\ell_2, \sigma_2)$ and $(\ell_1, \sigma_1) \xrightarrow{d_1} (\ell_1', \sigma_1')$,
  then $\exists \sigma_2'$, $\exists d_2$, such that $(\ell_2, \sigma_2) \xrightarrow{d_2} (\ell_2', \sigma_2')$ and $(\ell_1', \sigma_1') \equiv (\ell_2', \sigma_2')$,
- *vice-versa* (for the two last conditions).

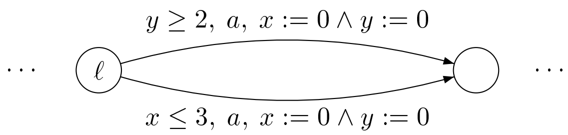
If W is a set of valuations, we denote by $W_{\equiv,\ell}$ the set $\{\sigma \mid \exists \sigma' \in W \text{ s.t. } (\ell, \sigma) \equiv (\ell, \sigma')\}$ and we define the induced symbolic transition relation $\Longrightarrow_{\equiv}$ by the rule:

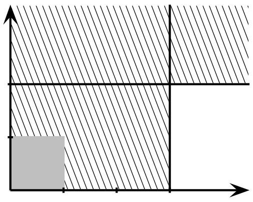
$$
\frac{(\ell, W) \Longrightarrow (\ell', W')}{(\ell, W) \Longrightarrow_{\equiv} (\ell', W'_{\equiv,\ell'})} \qquad \text{if } W = W_{\equiv,\ell}
$$

Lemma 1. *$\Longrightarrow_{\equiv}$ is* sound *and* complete *w.r.t. location-reachability.*

Though inducing a sound and complete symbolic transition relation, there is no simple way to represent $\equiv$-closures. In particular, as shown in the following example, for a given location ℓ and zone Z, the set $Z_{\equiv,\ell}$ might not be a zone.

Example 2. Assume that we reach state ℓ with the zone $Z = (0 \leq x, y \leq 1)$. The set $Z_{\equiv,\ell}$ is $\{\sigma \mid \sigma(x) \leq 3 \text{ or } \sigma(y) \geq 2\}$, which is not a zone (non-convex).





Location-Dependent Maximal Constants. Even with a suitable representation of the $\equiv$-closed sets (*e.g.* using lists of zones or structures as CDDs [4], DDDs [21]), the actual computation of the closure would be at least as difficult as solving the location-reachability problem itself [1, 2]. What we are looking for is an alternative abstraction obtained by a more efficient analysis of the relevance of guards. The method proposed in the following determines for each clock x and each location ℓ a maximum constant $\max_x^\ell$ beyond which the actual value of x in ℓ is irrelevant. Let $\{\max_x^\ell \mid x \in X, \ell \in L\}$ be a set of variables and define the system of inequalities, $\mathcal{S}_\mathcal{A}$, as follows:

Definition 5 (Location-dependent maximal constants.). *For each transition* $\ell \xrightarrow{g,\tau,\mathsf{up}} \ell'$ *of* $\mathcal{A}$*, we have the following inequalities in* $\mathcal{S}_\mathcal{A}$*:*

$$\begin{cases} \max_x^\ell \geq c, & \textit{if } (x \bowtie c) \textit{ is in the constraints } g \textit{ or } I(\ell), \\ \max_x^\ell \geq \max_y^{\ell'} - c, & \textit{if } (y := x + c) \textit{ is in } \mathsf{up} \textit{ and } x \leq d \textit{ or } x < d \textit{ is not in } g, \\ \max_x^\ell \geq \max_x^{\ell'}, & \textit{if } x \notin \mathsf{base}(\mathsf{up}). \end{cases}$$

Example 3. Consider the timed automaton $\mathcal{A}$ of Fig. 2. Then $\mathcal{S}_\mathcal{A}$ consists of the following inequalities:[3]

$$\left\{ \begin{array}{llll} \max_x^{\ell_1} \geq 5, \max_x^{\ell_2} & \max_x^{\ell_2} \geq 14, \max_x^{\ell_3} & \max_x^{\ell_3} \geq 14, \max_x^{\ell_4} & \max_x^{\ell_4} \geq \max_x^{\ell_1} \\ \max_y^{\ell_1} \geq \max_y^{\ell_2} & \max_y^{\ell_2} \geq \max_y^{\ell_3} & \max_y^{\ell_3} \geq 5 & \max_y^{\ell_4} \geq 10^6, \max_y^{\ell_1} \end{array} \right\}$$

Let $\alpha = (\max_x^\ell)_{x \in X, \ell \in L}$ be a solution of $\mathcal{S}_\mathcal{A}$. The equivalence $\cong_\alpha$ is defined by $(\ell, \sigma) \cong_\alpha (\ell', \sigma') \overset{\text{def}}{\iff} \ell = \ell'$ and $\forall x : \sigma(x) = \sigma'(x)$ or $\sigma(x), \sigma'(x) > \max_x^\ell$.

Lemma 2. *Whenever* $(\ell, \sigma) \cong_\alpha (\ell, \sigma')$ *then* $(\ell, \sigma) \equiv (\ell, \sigma')$.

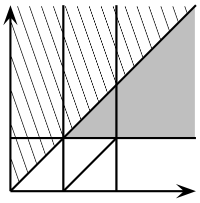
For W a set of valuations, let $W_{\cong_\alpha, \ell} = \{\sigma' \mid \exists \sigma \in W \text{ s.t. } (\ell, \sigma) \cong_\alpha (\ell, \sigma')\}$ and let '$\Longrightarrow_{\cong_\alpha}$' be the induced abstracted symbolic transition relation. From the previous Lemma 2 it follows that $W_{\cong_\alpha, \ell} \subseteq W_{\equiv, \ell}$ for any set of valuations W. As clearly $W \subseteq W_{\cong_\alpha, \ell}$, we have immediately:

Lemma 3. $\Longrightarrow_{\cong_\alpha}$ *is* sound *and* complete *w.r.t. location-reachability.*

Note that, if α and β are two solutions, it is clear that $W_{\cong_\beta, \ell} \subseteq W_{\cong_\alpha, \ell}$ whenever $\alpha \leq \beta$.[4] Thus, to maximize abstraction in the interest of early termination, we look for a smallest solution of $\mathcal{S}_\mathcal{A}$. For example, given the automaton $\mathcal{A}$ of Fig. 2, the smallest solution of $\mathcal{S}_\mathcal{A}$, as described in Example 3, is $\max_x^{\ell_i} = 14$ for $i \in \{1, 2, 3, 4\}$, $\max_y^{\ell_i} = 5$ for $i \in \{1, 2, 3\}$ and $\max_y^{\ell_4} = 10^6$.

[3] We write $a \geq b, c$ as a shorthand for the set of inequalities $a \geq b$, $a \geq c$.
[4] Here $\alpha \leq \beta$ is defined componentwise, *i.e.* $\alpha \leq \beta$ iff $\forall \ell \in L$, $\forall x \in X : \alpha_x^\ell \leq \beta_x^\ell$.

However, as for $\equiv$, there is no efficient way to represent $\cong_\alpha$-closures. Indeed, even if Z is a zone, it is not guaranteed that $Z_{\cong_\alpha,\ell}$ will be. For example, consider the zone Z (hashed) depicted in the figure on the right and let $\alpha^\ell = (2, 1)$. Then the set of valuations $Z_{\cong_\alpha,\ell}$ is obtained by adding the right part (gray). As a result, the union of these two is non-convex



Location-Dependent Abstraction Using DBMs. The (sound and complete) symbolic transition relations induced by the two abstractions considered so far, unfortunately do not preserve convexity of valuation-sets. In order to allow for valuation-sets to be represented *efficiently* as zones, we consider a slightly finer abstraction. In fact, we use a location-dependent version of the maximal constant abstraction on DBMs.

Given a timed automaton $\mathcal{A} = (L, \ell^0, I, E)$, let $\mathcal{S}_\mathcal{A}$ be the system of inequalities associated with $\mathcal{A}$ and $\alpha = (\max_x^\ell)_{x\in X,\ell\in L}$ be a solution to this system. For Z a zone, we define Z_α^ℓ as $\Lambda_{\alpha_{|\ell}}(Z)$ where $\alpha_{|\ell}$ is the tuple $(\max_x^\ell)_{x\in X}$ (see Definition 4 for Λ). The following non-trivial Lemma demonstrates that this zone-based abstraction is indeed finer than the two previously considered:

Lemma 4. *Let Z be a zone, $\ell \in L$ and α a solution to $\mathcal{S}_\mathcal{A}$, then $Z_\alpha^\ell \subseteq Z_{\cong_\alpha,\ell}$.*

The abstract transition system '$\Longrightarrow_\alpha$' is now induced in the obvious manner:

$$\frac{(\ell, Z) \Longrightarrow (\ell', Z')}{(\ell, Z) \Longrightarrow_\alpha (\ell', \Lambda_{\alpha_{|\ell'}}(Z'))} \qquad \text{if } \Lambda_{\alpha_{|\ell}}(Z) = Z$$

Note that, this transition system is well-defined (and consistent) because whenever Z is a zone, then also Z_α^ℓ is a zone. We may now state our main-theorem:

Theorem 1. *Let $\mathcal{A}$ be a timed automaton and $\mathcal{S}_\mathcal{A}$ the system of linear inequalities associated with $\mathcal{A}$. If α is a solution of $\mathcal{S}_\mathcal{A}$, then '$\Longrightarrow_\alpha$' is* sound *and* complete *w.r.t. location-reachability.*

Moreover, the symbolic reachability graph induced by '$\Longrightarrow_\alpha$' is obviously finite and is useful as the basis for a terminating, forward reachability algorithm. It should be noted that the use of location-dependent maximal constant abstraction *enlarges* the class of timed automata for which we may decide location-reachability compared with the previous method. For example, whereas the automaton of Fig. 3 may be analysed using the new location-dependent abstractions, the global maximal constant abstraction from [10] does not apply:

$$\mathcal{S}_\mathcal{B}^g = \begin{cases} \max_x \geq 2 \\ \max_x \leq \max_x - 1 \\ \max_x \geq 3 \end{cases} \qquad \mathcal{S}_\mathcal{B}^l = \begin{cases} \max_x^\ell \geq 2 \\ \max_x^{\ell'} \leq \max_x^\ell - 1 \\ \max_x^{\ell'} \geq 3 \end{cases}$$

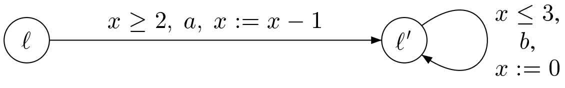


Fig. 3. Decrementing Timed Automaton, $\mathcal{B}$.

Here $\mathcal{S}^g_\mathcal{B}$ is the inequality system for global maximal constants and $\mathcal{S}^l_\mathcal{B}$ is the system for location-dependent maximal constants. Trivially, $\mathcal{S}^g_\mathcal{B}$ has no solutions, and $\mathcal{B}$ may consequently not be analysed using the methods of [10]. In contrast $\mathcal{S}^l_\mathcal{B}$ has the (minimal) solution α with $\max^\ell_x = 4$ and $\max^{\ell'}_x = 3$. Hence location-reachability is decidable for $\mathcal{B}$ using '$\Longrightarrow_\alpha$'.

5 Solving Simple Diophantine Constraints

We know from [17, 7] that the problem of solving a system of inequalities like $\mathcal{S}_\mathcal{A}$ is decidable, but the study done in these papers is much more general ([7] deals with general Presburger formulae) and the complexity is 3EXPTIME-complete. In this section, we provide, both, a polynomial and a linear algorithm for solving specific Diophantine inequality systems.

Minimal solutions. The interest of computing small solutions to $\mathcal{S}_\mathcal{A}$ appears clearly. Small solutions give large abstractions, and thus a smaller state-space to explore. The following lemma asserts that there is a unique minimal solution to the simple inequality systems we generate.[5]

Lemma 5. *Let $\mathcal{A}$ be a timed automaton and $\mathcal{S}_\mathcal{A}$ the inequality system associated to $\mathcal{A}$. If $\mathcal{S}_\mathcal{A}$ has a solution, then it has a unique minimal solution.*

Our aim is to provide efficient algorithms to find the minimal solution. We will reduce the problem to computing the longest paths in a digraph.

Reduction to a graph problem. We consider the system $\mathcal{S}_\mathcal{A}$ and we construct the directed graph $\mathcal{G}_\mathcal{A}$, having a vertex for each variable $\max^\ell_x$ in addition to the special vertex $\mathbf{0}$, and with the set of edges defined as follows:

- There is an edge $\max^\ell_x \xrightarrow{c} \mathbf{0}$ in $\mathcal{G}_\mathcal{A}$ if $\max^\ell_x \geq c$ is in $\mathcal{S}_\mathcal{A}$.
- There is an edge $\max^\ell_x \xrightarrow{c} \max^{\ell'}_y$ in $\mathcal{G}_\mathcal{A}$ if $\max^\ell_x \geq \max^{\ell'}_y + c$ is in $\mathcal{S}_\mathcal{A}$.

An edge labeled with $-\infty$ is equivalent to having no edge between two vertex. The relation between the graph $\mathcal{G}_\mathcal{A}$ and $\mathcal{S}_\mathcal{A}$ is stated by the following result:

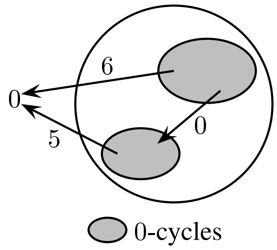
Proposition 1. *The system $\mathcal{S}_\mathcal{A}$ has a solution iff the graph $\mathcal{G}_\mathcal{A}$ has no positive cycle. Moreover, the minimal solution to the system $\mathcal{S}_\mathcal{A}$ corresponds to the longest paths in $\mathcal{G}_\mathcal{A}$ from each vertex "$\max^\ell_x$" to* $\mathbf{0}$.

The problem thus reduces to a graph problem: we compute the longest paths in a graph without positive cycles, which is equivalent (by inversing the edges) to compute shortest paths in graphs without negative cycles. We can thus use, for example, Floyd-Warshall's algorithm, which is polynomial ($\mathcal{O}(n^3)$) in the number of variables of the system, which is $n = |X| \cdot |L|$ where $|X|$ is the number of clocks of the automaton and $|L|$ the number of locations.

[5] This is not the case for general linear Diophantine inequality systems, see [17, 7].

A particular simpler case. We restrict to timed automata that use only updates of the form $x := c$ and $x := y$. The inequalities that we have to consider in this particular case are only of the two following forms: either $\max_x^\ell \geq \max_y^{\ell'}$ or $\max_x^\ell \geq c$ for a constant c. The graph that corresponds to such a system has a form like the one described below. Two vertex (different from **0**) are either not linked or the label of the edge is 0. There are moreover edges from non-**0** vertex to **0** that can be labeled by any positive integer.

In this (special) case, a simpler algorithm can be used:



- Compute the 0-cycles (*i.e.*, in this special case, the strongly connected components of the graph)
- The new graph, where we replace each 0-cycle by a single vertex, is a *dag*. Longest paths can be computed in linear time (in the number of strongly connected components, *i.e.* the number of variables of the system).

This algorithm has the nice property to be linear in the number of variables of the system (keep in mind that the number of variables is $|X| \cdot |L|$).

Active-Clock Reduction. A notion of active-clock reduction has been proposed in [15] for classical timed automata and has demonstrated a significant reduction in numerous case studies. This notion even makes sense for our more general model of timed automata, as defined in Section 2. Let $\mathcal{A}$ be a timed automaton. The active-clock reduction is computed as a (minimal) fix point by:

$$Act(\ell) = \bigcup_{(\ell \xrightarrow{g,a,u} \ell') \text{ in } \mathcal{A}} \mathsf{used}(g) \cup \mathsf{used}(u) \cup (Act(\ell') \setminus \mathsf{base}(u))$$

The following theorem states that active-clock reduction is a special case of our location-dependent abstraction technique.

Theorem 2. *Let $x \in X$, $\ell \in L$, and $(\max_x^\ell)_{x\in X,\ell\in L}$ be the minimal solution of $\mathcal{S}_\mathcal{A}$. Then $x \in Act(\ell)$ iff $\max_x^\ell > -\infty$.*

6 Dealing with Composition

Having so far developed a method for location-based abstraction for individual timed automata, it is now necessary to study its extension to the general case of *networks* of timed automata (see Section 2) in order to be applicable in tools such as UPPAAL. A simple solution would be to construct (at least logically) a single automaton having the same behaviour as the network and then apply our method to this product automaton. However, this approach would suffer from an exponential explosion in the size of the inequality system (e.g. the number of variables) to be solved w.r.t. the number of components of the network.

To avoid this explosion, we will develop *compositional* methods allowing (valid) location-dependent maximal constants of the product automata to be efficiently derived from location-dependent maximal constants of the components of the network. However, the fact that clocks may be shared (read and written) by several component automata of a network, complicates the combination of maximal constant information of components. To illustrate this reconsider the network of Fig. 1 from the introduction, where the clocks x and z are shared between $\mathcal{A}_1$ and $\mathcal{A}_2$. Considering the automaton $\mathcal{A}_2$ in isolation, the maximal constant(s) for z, $c_z^{n_i}$, seems to be 8 in both n_1 and n_2. However, the presence of the update $x := z + 3$ in $\mathcal{A}_2$ together with the invariants $(x \leq 14)$ in $\mathcal{A}_1$ requires that the value(s) of $c_z^{n_i}$ must take the maximal constants for x in $\mathcal{A}_1$ into account in order to be valid. How to make such transfer in a valid, yet efficient manner will be dealt with in this section.

For the sake of clarity, we fix some notation. For each network $\boldsymbol{\mathcal{A}}$, for each location vector $\boldsymbol{\ell}$ of this network and for each clock x, we denote by $\max_{\boldsymbol{\ell},x}^{\boldsymbol{\mathcal{A}}}$ the minimal solution for the system $\mathcal{S}_{\boldsymbol{\mathcal{A}}}$.

A First Simple Case. As a first simple case assume that $\boldsymbol{\mathcal{A}} = \mathcal{A}_1 \parallel \cdots \parallel \mathcal{A}_n$ and that each automaton $\mathcal{A}_i$ only uses updates of the form $x := c$, and no update of the form $x := y + c$. In addition, clocks may (or may not) be local in the sense that any clock is used (read and written) by at most one automaton. Now, consider the obvious combination of location-dependent maximal constants of components obtained by maximality, *i.e.* if c_i is the maximal constant for x in ℓ_i (of $\mathcal{A}_i$) then $\max\{c_1, \ldots, c_n\}$ will be the suggested maximal constant for x in $\boldsymbol{\ell}$ (of $\boldsymbol{\mathcal{A}}$) or formally: $\mathrm{Max}_{\boldsymbol{\ell},x}^{\boldsymbol{\mathcal{A}}} = \max\left\{\max_{\ell_i,x}^{\mathcal{A}_i} \mid i = 1 \ldots n\right\}$ $(*)$.
This combination yields a valid solution to $\mathcal{S}_{\boldsymbol{\mathcal{A}}}$ as stated in the following:

Proposition 2. *Let $\boldsymbol{\mathcal{A}}$ be a network only using updates of the form $x := c$, then* $\mathrm{Max}_{\boldsymbol{\ell},x}^{\boldsymbol{\mathcal{A}}} \geq \max_{\boldsymbol{\ell},x}^{\boldsymbol{\mathcal{A}}}$. *If moreover clocks are local, then* $\mathrm{Max}_{\boldsymbol{\ell},x}^{\boldsymbol{\mathcal{A}}} = \max_{\boldsymbol{\ell},x}^{\boldsymbol{\mathcal{A}}}$

The General Case. In the general case the component automata may have updates of the form $x := y + c$ as well as sharing of clocks. In particular, the case where one component, $\mathcal{A}_i$, contains a general update $x := y + c$ of x and another component, $\mathcal{A}_j$, a test $x \bowtie d$ on x requires a *transfer* of maximal constants for x (in certain locations) in $\mathcal{A}_j$ to maximal constants for y (in certain locations) in $\mathcal{A}_i$. Thus, the simple combination $(*)$ will not be valid. As an example consider the network of Fig. 1 and the corresponding inequality systems, $\mathcal{S}_{\mathcal{A}_1}$ and $\mathcal{S}_{\mathcal{A}_2}$[6]:

$$\mathcal{S}_{\mathcal{A}_1} = \left\{\begin{array}{ll} \max_x^{\ell_1} \geq 5, \max_x^{\ell_2} & \max_y^{\ell_1} \geq \max_y^{\ell_2} \\ \max_x^{\ell_2} \geq 14, \max_x^{\ell_3} & \max_y^{\ell_2} \geq \max_y^{\ell_3} \\ \max_x^{\ell_3} \geq 14, \max_x^{\ell_4} & \max_y^{\ell_3} \geq 5 \\ \max_x^{\ell_4} \geq \max_x^{\ell_1} & \max_y^{\ell_4} \geq 10^6, \max_y^{\ell_1} \end{array}\right\}, \quad \mathcal{S}_{\mathcal{A}_2} = \left\{\begin{array}{l} \max_z^{n_1} \geq 8 \\ \max_z^{n_2} \geq 4, \max_z^{n_1} \end{array}\right\}$$

Calculations give $\max_{\ell_2,z}^{\mathcal{A}_1} = -\infty$ and $\max_{n_1,z}^{\mathcal{A}_2} = 8$ and hence $\mathrm{Max}_{(\ell_2,n_1),z}^{\boldsymbol{\mathcal{A}}} = 8$ by $(*)$. However, this combination is invalid as it ignores the invariant $(x \leq 14)$ in $\mathcal{A}_1$, which combined with the update $x := z + 3$ in $\mathcal{A}_2$ will require $\max_{(\ell_2,n_1),z}^{\boldsymbol{\mathcal{A}}} \geq 11$.

[6] Note, that as $\mathcal{A}_2$ has no guards on x, there are no constraints on maximal constants involving x in $\mathcal{S}_{\mathcal{A}_2}$. Similar holds for $\mathcal{A}_1$ and z.

To obtain a valid inequality system, we take the transfer-update $x := z + 3$ (say) into account in the following way: for each location ℓ_i of $\mathcal{A}_1$ we add the constraint $\max_z^{n_1} \geq \max_x^{\ell_i} - 3$. Thus, to make the method compositional, we make no assumptions as to the location $\mathcal{A}_1$ might be in simultaneously with $\mathcal{A}_2$ being in location n_1. We add similar constraints to $\max_y^{\ell_2}$ to take the transfer-update $z := y+1$ into account. The two added transfer equation systems are thus: $\mathcal{T}_{1\to 2} = \{ \max_z^{n_1} \geq \max_x^{\ell_i} - 3 \;:\; 1 \leq i \leq 4 \}$, and $\mathcal{T}_{2\to 1} = \{ \max_y^{\ell_2} \geq \max_z^{\ell_i} - 1 : i = 1, 2 \}$.

Applying $(*)$ to the solutions found from the inequality system obtained by combining $\mathcal{S}_{\mathcal{A}_1}, \mathcal{S}_{\mathcal{A}_2}$ with $\mathcal{T}_{1\to 2}$ and $\mathcal{T}_{2\to 1}$ yields a valid solution to $\mathcal{S}_{\mathcal{A}_1 \| \mathcal{A}_2}$. In the remainder we formalize the method, state its correctness and complexity.

In general, for two different component automata $\mathcal{A}_i$ and $\mathcal{A}_j$ (with $i \neq j$), the transfer inequality system $\mathcal{T}_{i\to j}$ is obtained by adding for each update of the form $x := y + c$ in $\mathcal{A}_j$ with source-location ℓ a constraint $\max_{\ell,y}^{\mathcal{A}_j} \geq \max_{n,x}^{\mathcal{A}_i} - c$ for *any* location n of $\mathcal{A}_i$. Now, by combining the component inequality systems with the transfer inequality systems into $\mathcal{M} = \bigcup \{ \mathcal{S}_{\mathcal{A}_i} \;:\; 1 \leq i \leq n \} \cup \bigcup \{ \mathcal{T}_{i\to j} \;:\; i \neq j \}$ and defining: $\mathrm{MAX}_{\boldsymbol{\ell},x}^{\boldsymbol{\mathcal{A}}} = \max\{ \max_{\ell_i,x}^{\mathcal{M}} \mid i = 1 \ldots n \}$, where $(\max_{\ell_i,x}^{\mathcal{M}})$ is the minimal solution to $\mathcal{M}$, the following proposition holds:

Proposition 3. *Let $\boldsymbol{\mathcal{A}}$ be a network of timed automata. Then* $\mathrm{MAX}_{\boldsymbol{\ell},x}^{\boldsymbol{\mathcal{A}}} \geq \max_{\boldsymbol{\ell},x}^{\boldsymbol{\mathcal{A}}}$.

Thus from the (minimal) solution to $\mathcal{M}$ we may obtain valid location-dependent constants. It is important to note that the size of the system of inequalities $\mathcal{M}$ grows polynomial with the number of components of the network[7] thus allowing for our solution methods from the previous section to scale up. The computation of the maximal constants corresponding to particular location-vectors (and clocks) will be obtained from the minimal solution to $\mathcal{M}$ in an on-the-fly fashion.

7 Experiments with UPPAAL

A first prototype of the location-dependent abstraction technique has been implemented in UPPAAL. The fragment considered for this prototype can deal with networks of automata and resets to a constant ($x := c$). The algorithms are those described in section 5 and section 6. Our algorithm is expected to beat the standard approaches for timed automata in which we have a tremendous difference on clock constraints from one location to another one. In order to demonstrate this, let us consider a *naive example* (Fig. 4). In such an automaton, and considering a global approach of the maximum constants on clocks, the constant BIG plays a crucial role in the analysis of the system. The bigger the constant BIG is, the longer the analysis will last. Indeed, one can notice the fact that applying the location-dependent analysis on this automaton reduces the maximum constants of y to BIG in the initial location p and to 1 in the location q. These results have a huge impact on the analysis of the model. In Table 1, this naive example and the resources of its analysis are displayed for several values of the constant BIG. The *Global Method* refers to the classical approach, the *Active-clock Reduction* refers to the

[7] Assuming a fixed number of locations and clocks for each component the number of variables grows linearly and the number or inequalities grows quadraticly.

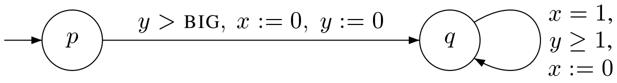


Fig. 4. Naive Example.

algorithm which only considers the clocks which are active in the locations [15], and finally, the *Local Constants* refers to our method. The time performance of our method is insensitive to the size of constant BIG.

Two Processes is the example from the introduction (see Fig. 1), but slightly modified: the updates of the form $x := y + c$ have been rewritten into $x := y$ and hard coded into the automaton[8]. The constant BIG is the constraint on the clock y valued 10^6. Once again, our abstraction is coarser than the ones traditionally applied, and therefore performs better on the verification.

Table 1. *Experimental Results (Intel PentiumIV@1.8GHz).*

	Constant BIG	*Global Method*	*Active-clock Reduction*	*Local Constants*
Naive Example	10^3	0.05s/1MB	0.05s/1MB	0.00s/1MB
	10^4	4.78s/3MB	4.83s/3MB	0.00s/1MB
	10^5	484s/13MB	480s/13MB	0.00s/1MB
	10^6	stopped	stopped	0.00s/1MB
Two Processes	10^3	3.24s/3MB	3.26s/3MB	0.01s/1MB
	10^4	5981s/9MB	5978s/9MB	0.37s/2MB
	10^5	stopped	stopped	72s/5MB
Asymmetric Fischer	10^3	0.01s/1MB	0.01s/1MB	0.01s/1MB
	10^4	2.20s/3MB	2.20s/3MB	0.85s/2MB
	10^5	333s/19MB	333s/19MB	160s/13MB
	10^6	33307s/122MB	33238s/122MB	16330s/65MB
Bang & Olufsen	25000	stopped	159s/243MB	123s/204MB

The next example, namely *Asymmetric Fischer*, refers to a classical two process Fischer example where the constants of one of the processes have been changed to the constant BIG. Experiments show a gain of 50% in time. The final example, referred to as *Bang & Olufsen*, is an industrial case study [18]. The Bang & Olufsen Power Down Protocol controls the transitions between stand-by mode and power-on mode in the company's products, where power consumption minimization is an important feature. The UPPAAL model of this protocol heavily uses a clock c and introduces a certain amount of guards with constants from 1 to 25000. The way the model is built introduces a lot of locations where the maximum constant of c can be reduced from 25000 to some lower constants. Without any modification of the model we have noticed an improvement of 25% in the speed and 20% of the memory usage.

[8] The technique used to transform timed automata with $x := c$ and $x := y$ into timed automata with only resets is described in [11].

Finally, it is crucial to point out that, as expected by the theory, our algorithm is performing as well as the active-clock reduction technique in all other examples that we tried (including the total suite of UPPAAL benchmarks). Location-dependent abstraction out performs active-clock reduction only when it deals with models in which there is a big difference on the value of the maximum constants from state to state (as demonstrated in this section), but has no effect on models which do not have this sort of property. We also emphasize the fact that our experiments did not exhibit any significant difference between the performance of the active-clock reduction and our method.

8 Conclusions and Further Work

In this paper, we have shown that the classical zone construction used to obtain a finite abstraction of a timed automaton is sensitive to large differences in the constants to which clocks are compared. We have contributed a *location-dependent zone abstraction*, which uses static analysis to identify the *relevant guards and invariants* in a given location. We have shown that this abstraction generalises the well-known active-clock reduction technique. In addition, we have extended the concept to the case of networks of timed automata and to the case of more general updates of clocks. Experiments have demonstrated, that our abstraction in some cases can result in an exponential speedup. On real-world cases, we either match or surpass the performance of active-clock reduction, depending on whether the system compares clocks to different constants or not.

There are a number of open questions, that need further work. Our experiments do not evaluate the quality of the heuristic described in Section 6 used for networks of timed automata, *i.e.*, whether computing the maximum constants based on the product automaton would yield significantly smaller constants. Also, we have not tested with systems containing non-trivial updates ($x := y+c$). This is partly due to lack of realistic systems using these kind of updates.

In UPPAAL, clocks (and clock differences) may be compared to expressions over bounded integer variables. Extending the active guard reduction technique to this case involves finding the smallest upper bound of an integer expression in a given location. Also, the idea of active-clock reduction could equally be applied to integer variables. In the sequential case, this has been studied in the field of compiler theory and is also related to slicing techniques (recently added to SPIN). Finally, we have not yet explored the fact, that we can verify a broader class of timed automata compared to the classic approach. A case-study demonstrating the usefulness of this claim is needed.

References

1. R. Alur, C. Courcoubetis, D. Dill, N. Halbwachs, and H. Wong-Toi. An Implementation of Three Algorithms for Timing Verification Based on Automata Emptiness. In *Proc. 13th IEEE Real-Time Systems Symp. (RTSS'92)*, pp. 157–166. IEEE Computer Society Press, 1992.
2. R. Alur, C. Courcoubetis, N. Halbwachs, D. Dill, and H. Wong-Toi. Minimization of Timed Transition Systems. In *Proc. 3rd Int. Conf. on Concurrency Theory (CONCUR'92)*, vol. 630 of *LNCS*, pp. 340–354. Springer, 1992.

3. R. Alur and D. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
4. G. Behrmann, K. G. Larsen, J. Pearson, C. Weise, and W. Yi. Efficient Timed Reachability Analysis Using Clock Difference Diagrams. In *Proc. 11th Int. Conf. on Computer Aided Verification (CAV'99)*, vol. 1633 of *LNCS*, pp. 341–353. Springer, 1999.
5. J. Bengtsson. *Clocks, DBMs ans States in Timed Systems*. PhD thesis, Dept. of Information Technology, Uppsala Univ., Uppsala, Sweden, 2002.
6. J. Bengtsson, K. G. Larsen, Fredrik Larsson, P. Pettersson, W. Yi, and Carsten Weise. New Generation of UPPAAL. In *Proc. Int. Workshop on Software Tools for Technology Transfer (STTT'98)*, BRICS Notes, pp. 43–52, 1998.
7. A. Boudet and H. Comon. Diophantine Equations, Presburger Arithmetic and Finite Automata. In *Proc. 21st Int. Coll. on Trees in Algebra and Programming (CAAP'96)*, vol. 1059 of *LNCS*, pp. 30–43. Springer, 1996.
8. P. Bouyer. Timed Automata May Cause Some Troubles. Research Report LSV–02–9, LSV, ENS de Cachan, France, 2002.
9. P. Bouyer. Untameable Timed Automata! In *Proc. 20th Annual Symp. on Theoretical Aspects of Computer Science (STACS'2003)*, 2003. To appear.
10. P. Bouyer, C. Dufourd, E. Fleury, and A. Petit. Are timed automata updatable? In *Proc. 12th Int. Conf. on Computer Aided Verification (CAV'2000)*, vol. 1855 of *LNCS*, pp. 464–479. Springer, 2000.
11. P. Bouyer, C. Dufourd, E. Fleury, and A. Petit. Expressiveness of updatable timed automata. In *Proc. 25th Int. Symp. on Mathematical Foundations of Computer Science (MFCS'2000)*, vol. 1893 of *LNCS*, pp. 232–242. Springer, 2000.
12. M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, and S. Yovine. KRONOS: a Model-Checking Tool for Real-Time Systems. In *Proc. 10th Int. Conf. on Computer Aided Verification (CAV'98)*, vol. 1427 of *LNCS*, pp. 546–550. Springer, 1998.
13. V. Braberman, D. Garbervetsky, and A. Olivero. Improving the Verification of Timed Systems Using Influence Information. In *Proc. 8th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'02)*, vol. 2280 of *LNCS*, pp. 21–36. Springer, 2002.
14. E. Clarke, O. Grumberg, and D. Peled. *Model-Checking*. The MIT Press, 1999.
15. C. Daws and S. Tripakis. Model-Checking of Real-Time Reachability Properties using Abstractions. In *Proc. 4th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'98)*, vol. 1384 of *LNCS*, pp. 313–329. Springer, 1998.
16. D. Dill. Timing Assumptions and Verification of Finite-State Concurrent Systems. In *Proc. of the Workshop on Automatic Verification Methods for Finite State Systems*, vol. 407 of *LNCS*, pp. 197–212. Springer, 1989.
17. E. Domenjoud. Solving Systems of Linear Diophantine Equations: an Algebraic Approach. In *Proc. 16th Int. Symp. on Mathematical Foundations of Computer Science (MFCS'91)*, vol. 520 of *LNCS*, pp. 141–150. Springer, 1991.
18. K. Havelund, A. Skou, K.G. Larsen, and K. Lund. Formal Modeling and Analysis of an Audio/Video Protocol: an Industrial Case Study using UPPAAL. In *Proc. 18th IEEE Real-Time Systems Symp. (RTSS'97)*, pp. 2–13. IEEE Computer Society Press, 1997.
19. F. Laroussinie and K.G. Larsen. CMC: a Tool for Compositional Model-Checking of Real-Time Systems. In *Proc. IFIP Int. Conf. on Formal Description Techniques & Protocol Specification, Testing, and Verification (FORTE-PSTV'98)*, pp. 439–456. Kluwer Academic, 1998.
20. K. G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a Nutshell. *Software Tools for Technology Transfer*, 1(1–2):134–152, 1997.
21. J. Møller, J. Lichtenberg, H.R. Andersen, and H. Hulgaard. Difference Decision Diagrams. In *Proc. 13th Int. Workshop on Computer Science Logic (CSL'99)*, vol. 1683 of *LNCS*, pp. 111–125. Springer, 1999.