Software Tools for Technology Transfer manuscript No.
(will be inserted by the editor)

Lower and Upper Bounds in Zone Based Abstractions of Timed Automata

Gerd Behrmann[1], **Patricia Bouyer**[2⋆], **Kim G. Larsen**[1], **Radek Pelánek**[3⋆⋆]

[1] BRICS, Aalborg University, Denmark
e-mail: {behrmann,kgl}@cs.auc.dk
[2] LSV, CNRS & ENS de Cachan, UMR 8643, France
e-mail: bouyer@lsv.ens-cachan.fr
[3] Masaryk University Brno, Czech Republic
e-mail: xpelanek@informatics.muni.cz

The date of receipt and acceptance will be inserted by the editor

Abstract. The semantics of timed automata is defined using an infinite-state transition system. For verification purposes, one usually uses zone based abstractions w.r.t. the maximal constants to which clocks of the timed automaton are compared. We show that by distinguishing maximal lower and upper bounds, significantly coarser abstractions can be obtained. We show soundness and completeness of the new abstractions w.r.t. reachability. We demonstrate how information about lower and upper bounds can be used to optimise the algorithm for bringing a difference bound matrix into normal form. Finally, we experimentally demonstrate that the new techniques dramatically increases the scalability of the real-time model checker UPPAAL.

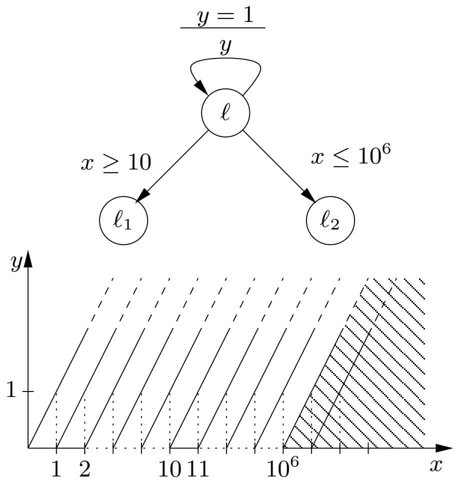


Fig. 1. A small timed automaton. The state space of the automaton when in location ℓ is shown. The area to the right is the abstraction of the last zone.

1 Introduction

Since their introduction by Alur and Dill in [AD90, AD94], timed automata have become one of the most well established models for real-time systems with well-studied underlying theory and development of mature model-checking tools such as UPPAAL [LPY97] and KRONOS [BDM+98]. By their very definition timed automata describe (uncountable) infinite state-spaces. Thus, algorithmic verification relies on the existence of exact finite abstractions. In the original work by Alur and Dill, the so-called region-graph construction provided a "universal" such abstraction. However, whereas well-suited for establishing decidability of problems related to timed automata, the region-graph construction is highly impractical from a tool-implementation point of view. Instead, most real-time verification tools apply abstractions based on so-called zones, which in practise provide much coarser (and hence smaller) abstractions.

To ensure finiteness, it is essential that the given abstraction (region as well as zone based) takes into account the actual constants with which clocks are compared. In particular, the abstraction could identify states which are identical except for the clock values which exceed the *maximum* such constants. Obviously, the smaller we may choose these maximum constants, the coarser the resulting abstraction will be. Allowing clocks to be assigned different (maximum) constants is an obvious first step in this direction, and in [BBFL03] this idea has been (successfully) taken further by allowing the maximum constants not only to depend of the particular clock but also of the particular location of the timed automaton. In all cases the *exactness* is established by proving that the abstraction respects *bisimilarity*, *i.e.* states identified by the abstraction are bisimilar.

⋆ Partially supported by ACI Cortos. Work partly done while visiting CISS, Aalborg University.
⋆⋆ Partially supported by GA ČR grant no. 201/03/0509.

Consider now the timed automaton of Fig. 1. Clearly 10^6 is the maximum constant for x and 1 is the maximum constant for y. Thus, abstractions based on maximum constants will distinguish all states where $x \leq 10^6$ and $y \leq 1$. In particular, a forward computation of the full state space will – regardless of the search-order – create an excessive number of abstract (symbolic) states including all abstract states of the form $(\ell, x - y = k)$ where $0 \leq k \leq 10^6$ as well as $(\ell, x - y > 10^6)$. However, assuming that we are only interested in *reachability* properties (as is often the case in UPPAAL), the application of so-called downwards closure with respect to *simulation* will lead to an exact abstraction which could potentially be substantially coarser than closure under bisimilarity. Observing that 10^6 is an *upper* bound on the edge from ℓ to ℓ_2 in Fig. 1, it is clear that for any state where $x \geq 10$, increasing x will only lead to "smaller" states with respect to simulation preorder. In particular, applying this downward closure results in the radically smaller collection of abstract states, namely $(\ell, x - y = k)$ where $0 \leq k \leq 10$ and $(\ell, x - y > 10)$.

The fact that 10^6 is an *upper* bound in the example of Fig. 1 is crucial for the reduction we obtained above. In this paper we present new, substantially coarser yet still exact abstractions which are based on *two* maximum constants obtained by distinguishing lower and upper bounds. In all cases the exactness (w.r.t. reachability) is established by proving that the abstraction respects downwards closure w.r.t. simulation, *i.e.* for each state in the abstraction there is an original state simulating it. The variety of abstractions comes from the additional requirements to *effective* representation and *efficient* computation and manipulation. In particular we insist that zones can form the basis of our abstractions; in fact the suggested abstractions are defined in terms of low-complexity transformations of the difference bound matrix (DBM) representation of zones. Furthermore, we demonstrate how information about lower and upper bounds can be used to optimise the algorithm for bringing a DBM into normal form. Finally, we experimentally demonstrate the significant speedups obtained by our new abstractions, to be comparable with the convex hull over-approximation supported by UPPAAL. Here, the distinction between lower and upper bounds is combined with the orthogonal idea of location-dependency of [BBFL03].

Related Work: This paper is a full and extended version of [BBLP04]. In particular, proof of the correctness of our abstraction techniques is given in Appendix. Also, full information on how to exploit lower and upper bounds to obtain smaller asymmetric DBMs and faster successor computation is given. Finally, we show that applying our abstraction techniques to jobshop scheduling problems extends the notion of dominating point proposed in [AM01, AAM04].

The classical abstraction method for timed automata takes into account the maximum constants to which the various clocks are compared. This method — termed extrapolation or normalization — is described in [DT98] and rigorously proved in [Bou03]. In [BBFL03] the maximum constrants used in the abstraction do not only depend on the clocks but also on the particular locations of the timed automata. [DT98] describes a number of additional abstraction techniques including active clock reduction and the over-approximating convex hull abstraction. In [BBFL03] it is shown that active clock reduction is obtained as a special case of the location-dependent abstraction. In this paper we compare the performance of our new (exact) abstraction technique to that of convex hull approximation.

Most of the above zone based abstractions — including the ones examined in this paper — require that guards are restricted to conjunctions of simple lower or upper bounds (strict or non-strict) on individual clocks. Thus, constraints on clock differences are generally not allowed. We refer the reader to [Bou03] to learn more about the intricacies of zone based abstractions in the presence of difference constraints, and to [BY03] for a solution based on zone splitting.

2 Preliminaries

Although we perform our experiments in UPPAAL, we describe the theory on the basic timed automaton model. Variables, committed locations, networks, urgency, and other things supported by UPPAAL are not important with respect to presented ideas and the technique can easily be extended for these "richer" models. Let X be a set of non-negative real-valued variables called *clocks*. The set of guards $G(X)$ is defined by the grammar $g := x \bowtie c \mid g \wedge g$, where $x \in X, c \in \mathbb{N}$ and $\bowtie \in \{<, \leq, \geq, >\}$.

Definition 1 (Timed Automata Syntax). A *timed automaton* is a tuple $\mathcal{A} = (L, X, \ell_0, E, I)$, where L is a finite set of locations, X is a finite set of clocks, $\ell_0 \in L$ is an initial location, $E \subseteq L \times G(X) \times 2^X \times L$ is a set of edges labelled by guards and a set of clocks to be reset, and $I : L \to G(X)$ assigns invariants to clocks.

A *clock valuation* is a function $\nu : X \to \mathbb{R}_{\geq 0}$. If $\delta \in \mathbb{R}_{\geq 0}$ then $\nu + \delta$ denotes the valuation such that for each clock $x \in X$, $(\nu + \delta)(x) = \nu(x) + \delta$. If $Y \subseteq X$ then $\nu[Y := 0]$ denotes the valuation such that for each clock $x \in X \setminus Y$, $\nu[Y := 0](x) = \nu(x)$ and for each clock $x \in Y$, $\nu[Y := 0](x) = 0$. The satisfaction relation $\nu \models g$ for $g \in G(X)$ is defined in the natural way.

Definition 2 (Timed Automata Semantics). The semantics of a timed automaton $\mathcal{A} = (L, X, \ell_0, E, I)$ is defined by a transition system $S_{\mathcal{A}} = (S, s_0, \longrightarrow)$, where $S = L \times \mathbb{R}^X_{\geq 0}$ is the set of states, $s_0 = (\ell_0, \nu_0)$ is the initial state, $\nu_0(x) = 0$ for all $x \in X$, and $\longrightarrow \subseteq S \times S$ is the set of transitions defined by:

– $(\ell, \nu) \longrightarrow (\ell, \nu + \delta)$ if $\forall 0 \leq \delta' \leq \delta : (\nu + \delta') \models I(l)$
– $(\ell, \nu) \longrightarrow (\ell', \nu[Y := 0])$ if there exists $(\ell, g, Y, \ell') \in E$ such that $\nu \models g$ and $\nu[Y := 0] \models I(\ell')$

The *reachability problem* for an automaton $\mathcal{A}$ and a location ℓ is to decide whether there is a state (ℓ, ν) reachable from (ℓ_0, ν_0) in the transition system $S_{\mathcal{A}}$. As usual, for verification purposes, we define a symbolic semantics for timed automata.

Definition 3 (Symbolic Semantics). The symbolic semantics of a timed automaton $\mathcal{A} = (L, X, \ell_0, E, I)$ is defined by the abstract transition system $(S, s_0, \Longrightarrow)$, where $S = L \times \mathcal{P}(\mathbb{R}^X_{\geq 0})^1$, $s_0 = (\ell_0, \{\nu_0\})$ and '$\Longrightarrow$' is defined by the following two rules:

Delay: $(\ell, W) \Longrightarrow (\ell, W')$, where $W' = \{\nu + d \mid \nu \in W \wedge d \geq 0 \wedge \forall 0 \leq d' \leq d : (\nu + d') \models I(\ell)\}$.
Action: $(\ell, W) \Longrightarrow (\ell', W')$ if there exists a transition $\ell \xrightarrow{g,Y} \ell'$ in $\mathcal{A}$, such that $W' = \{\nu' \mid \exists \nu \in W : \nu \models g \wedge \nu' = \nu[Y := 0] \wedge \nu' \models I(\ell')\}$.

The symbolic semantics of a timed automaton may induce an infinite transition system. To obtain a finite graph one may, as suggested in [BBFL03], apply some abstraction $\mathfrak{a} : \mathcal{P}(\mathbb{R}^X_{\geq 0}) \longrightarrow \mathcal{P}(\mathbb{R}^X_{\geq 0})$, such that $W \subseteq \mathfrak{a}(W)$ and $\mathfrak{a}(\mathfrak{a}(W)) = \mathfrak{a}(W)$. The abstract transition system '$\Longrightarrow_{\mathfrak{a}}$' is then given by the following inference rule:

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

A simple way to ensure that the reachability graph induced by '$\Longrightarrow_{\mathfrak{a}}$' is finite is to establish that there is only a finite number of 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 to be *sound* and *complete* (w.r.t. reachability) whenever:

Sound: $(\ell_0, \{\nu_0\}) \Longrightarrow^*_{\mathfrak{a}} (\ell, W)$ implies $\exists \nu : \nu \in W$ and $(\ell_0, \nu_0) \longrightarrow^* (l, \nu)$.
Complete: $(\ell_0, \nu_0) \longrightarrow^* (\ell, \nu)$ implies $\exists W : \nu \in W$ and $(\ell_0, \{\nu_0\}) \Longrightarrow^*_{\mathfrak{a}} (\ell, W)$.

By language misuse, we say that an abstraction $\mathfrak{a}$ is *sound* (resp. *complete*) whenever '$\Longrightarrow_{\mathfrak{a}}$' is sound (resp. complete). 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, as coarse as possible, and which induces a sound abstract transition system. We also require that abstractions are *effectively* representable and may be *efficiently* computed and manipulated.

A first step in finding an effective abstraction is realising that W will always be a zone whenever $(\ell, \{\nu_0\}) \Longrightarrow^* (\ell, W)$. A *zone* is a conjunction of constraints of the form $x \bowtie c$ or $x - y \bowtie c$, where x and y are clocks, $c \in \mathbb{Z}$, and $\bowtie$ is one of $\{\leq, \leq, =, \geq, >\}$. Zones can be represented using *Difference Bound Matrices* (DBM). We will briefly recall the definition of DBMs, and refer to [Dil89, CGP99, Ben02, Bou04] for more details. A DBM is a square matrix $D = \langle c_{i,j}, \prec_{i,j} \rangle_{0 \leq i,j \leq n}$ such that $c_{i,j} \in \mathbb{Z}$ and $\prec_{i,j} \in \{<, \leq\}$ or $c_{i,j} = \infty$ and $\prec_{i,j} = <$. The DBM D represents the zone $[\![D]\!]$ which is defined by $[\![D]\!] = \{\nu \mid \forall 0 \leq i, j \leq n,\ \nu(x_i) - \nu(x_j) \prec_{i,j} c_{i,j}\}$, where $\{x_i \mid 1 \leq i \leq n\}$ is the set of clocks, and x_0 is a clock which is always 0, (*i.e.* for each valuation ν, $\nu(x_0) = 0$). 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 $D = \langle c_{i,j}, \prec_{i,j} \rangle_{0 \leq i,j \leq n}$ is a DBM in normal form, then it satisfies the *triangular inequality*, that is, for every $0 \leq i, j, k \leq n$, we have that $(c_{i,j}, \prec_{i,j}) \leq (c_{i,k}, \prec_{i,k}) + (c_{k,j}, \prec_{k,j})$, where comparisons and additions are defined in a natural way (see [Bou04]). All operations needed to compute '$\Longrightarrow$' can be implemented by manipulating the DBMs.

3 Maximum Bound Abstractions

The abstraction used in real-time model-checkers such as Uppaal [LPY97] and Kronos [BDM+98], is based on the idea that the behaviour of an automaton is only sensitive to changes of a clock if its value is below a certain constant. That is, for each clock there is a maximum constant such that 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* [Bou03, BBFL03] or *normalisation* [DT98]. In the following we will choose the term *extrapolation*.

Simulation & Bisimulation. The notion of bisimulation has so far been the semantic tool for establishing soundness of suggested abstractions. In this paper we shall exploit the more liberal notion of simulation [Mil71, TARB96] to allow for even coarser abstractions. Let us fix a timed automaton $\mathcal{A} = (L, X, \ell_0, E, I)$. We consider a $\preccurlyeq$ relation on $L \times \mathbb{R}^X_{\geq 0}$ satisfying the following transfer properties:

1. if $(\ell_1, \nu_1) \preccurlyeq (\ell_2, \nu_2)$ then $\ell_1 = \ell_2$
2. if $(\ell_1, \nu_1) \preccurlyeq (\ell_2, \nu_2)$ and $(\ell_1, \nu_1) \longrightarrow (\ell'_1, \nu'_1)$, then there exists (ℓ'_2, ν'_2) such that $(\ell_2, \nu_2) \longrightarrow (\ell'_2, \nu'_2)$ and $(\ell'_1, \nu'_1) \preccurlyeq (\ell'_2, \nu'_2)$
3. if $(\ell_1, \nu_1) \preccurlyeq (\ell_2, \nu_2)$ and $(\ell_1, \nu_1) \longrightarrow (\ell_1, \nu_1 + \delta)$, then there exists δ' such that $(\ell_2, \nu_2) \longrightarrow (\ell_2, \nu_2 + \delta')$ and $(\ell_1, \nu_1 + \delta) \preccurlyeq (\ell_2, \nu_2 + \delta')$

We call such a relation a (*location-based*) *simulation* relation or simply a *simulation* relation. A simulation relation $\preccurlyeq$ such that $\preccurlyeq^{-1}$ is also a simulation relation, is

[1] $\mathcal{P}(A)$ denotes the set of all subsets of A.

called a (location-based) *bisimulation relation*. Whenever $(\ell_1,\nu_1) \preccurlyeq (\ell_2,\nu_2)$, we say that (ℓ_1,ν_1) is simulated by (ℓ_2,ν_2) or that (ℓ_2,ν_2) simulates (ℓ_1,ν_1).

Proposition 1. *Let $\preccurlyeq$ be a simulation relation, as defined above. If $(\ell,\nu_1) \preccurlyeq (\ell,\nu_2)$ and if a discrete state ℓ' is reachable from (ℓ,ν_1), then it is also reachable from (ℓ,ν_2).*

Reachability is thus preserved by simulation as well as by bisimulation. However, in general the weaker notion of simulation preserves fewer properties than that of bisimulation. For example, deadlock properties as expressed in UPPAAL [2] are not preserved by simulation whereas it is preserved by bisimulation. In Fig. 1, $(\ell, x = 15, y = .5)$ simulates $(\ell, x = 115, y = .5)$ as well as $(\ell, x = 10^6 + 1, y = .5)$.

Classical Maximal Bounds. The classical abstraction for timed automata is based on maximal bounds, one for each clock of the automaton. Let $\mathcal{A} = (L, X, \ell_0, E, I)$ be a timed automaton. The *maximal bound* of a clock $x \in X$, denoted $M(x)$, is the maximal constant k such that there exists a guard or invariant containing $x \bowtie k$ in $\mathcal{A}$. Let ν and ν' be two valuations. We define the following relation: $\nu \equiv_M \nu' \stackrel{\text{def}}{\Longleftrightarrow} \forall x \in X$: either $\nu(x) = \nu'(x)$ or $(\nu(x) > M(x)$ and $\nu'(x) > M(x))$.

Lemma 1. *The relation $\mathcal{R} = \{((\ell,\nu),(\ell,\nu')) \mid \nu \equiv_M \nu'\}$ is a bisimulation relation.*

We can now define the abstraction $\mathfrak{a}_{\equiv_M}$ w.r.t. $\equiv_M$. Let W be a set of valuations, then $\mathfrak{a}_{\equiv_M}(W) = \{\nu \mid \exists \nu' \in W, \nu' \equiv_M \nu\}$.

Lemma 2. *The abstraction $\mathfrak{a}_{\equiv_M}$ is sound and complete.*

These two lemmas come from [BBFL03]. They will moreover be consequences of our main result.

Lower & Upper Maximal Bounds. The new abstractions introduced in the following will be substantially coarser than $\mathfrak{a}_{\equiv_M}$. It is no longer based on a single maximal bound per clock but rather on two maximal bounds per clock allowing lower and upper bounds to be distinguished.

Definition 4. Let $\mathcal{A} = (L, X, \ell_0, E, I)$ be a timed automaton. The *maximal lower bound* denoted $L(x)$, (resp. *maximal upper bound* $U(x)$) of clock $x \in X$ is the maximal constant k such that there exists a constraint $x > k$ or $x \geq k$ (resp. $x < k$ or $x \leq k$) in a guard of some transition or in an invariant of some location of $\mathcal{A}$. If such a constant does not exist, we set $L(x)$ (resp. $U(x)$) to $-\infty$.

[2] There is a deadlock whenever there exists a state (ℓ,ν) such that no further discrete transition can be taken.

Let us fix for the rest of this section a timed automaton $\mathcal{A}$ and bounds $L(x)$, $U(x)$ for each clock $x \in X$ as above. The idea of distinguishing lower and upper bounds is the following: if we know that the clock x is between 2 and 4, and if we want to check that the constraint $x \leq 5$ can be satisfied, the only relevant information is that the value of x is greater than 2, and not that $x \leq 4$. In other terms, checking the emptiness of the intersection between a non-empty interval $[c,d]$ and $]-\infty, 5]$ is equivalent to checking whether $c > 5$; the value of d is not useful. Formally, we define the LU-preorder as follows.

Definition 5 (LU-preorder $\preccurlyeq_{LU}$). Let ν and ν' be two valuations. Then $\nu' \preccurlyeq_{LU} \nu$ if and only if for each clock x:

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

Lemma 3. *The relation $\mathcal{R} = \{((\ell,\nu),(\ell,\nu')) \mid \nu' \preccurlyeq_{LU} \nu\}$ is a simulation relation.*

Proof: The only non-trivial part in proving that $\mathcal{R}$ indeed satisfies the three transfer properties of a simulation relation is to establish that if g is a clock constraint, then "$\nu \models g$ implies $\nu' \models g$". Consider the constraint $x \leq c$. If $\nu(x) = \nu'(x)$, then we are done. If $L(x) < \nu'(x) < \nu(x)$, then $\nu(x) \leq c$ implies $\nu'(x) \leq c$. If $U(x) < \nu(x) < \nu'(x)$, then it is not possible that $\nu \models x \leq c$ (because $c \leq U(x)$). Consider now the constraint $x \geq c$. If $\nu(x) = \nu'(x)$, then we are done. If $U(x) < \nu(x) < \nu'(x)$, then $\nu(x) \geq c$ implies $\nu'(x) \geq c$. If $L(x) < \nu'(x) < \nu(x)$, then it is not possible that ν satisfies the constraint $x \geq c$ because $c \leq L(x)$. ∎

Using the above LU-preorder, we can now define a first abstraction based on the lower and upper bounds.

Definition 6 ($\mathfrak{a}_{\preccurlyeq_{LU}}$, abstraction w.r.t. $\preccurlyeq_{LU}$). Let W be a set of valuations. We define the abstraction w.r.t. $\preccurlyeq_{LU}$ as $\mathfrak{a}_{\preccurlyeq_{LU}}(W) = \{\nu \mid \exists \nu' \in W,\ \nu' \preccurlyeq_{LU} \nu\}$.

Before going further, we illustrate this abstraction in Fig. 2. There are several cases, depending on the relative positions of the two values $L(x)$ and $U(x)$ and of the valuation we are looking at. We represent with a plain line the value of $\mathfrak{a}_{\preccurlyeq_{LU}}(\{\nu_1\})$ and with a dashed line the value of $\mathfrak{a}_{\equiv_M}(\{\nu_2\})$, where the maximal bound $M(x)$ corresponds to the maximum of $L(x)$ and $U(x)$. In each case, we indicate the "quality" of the new abstraction compared with the "old" one. We notice that the new abstraction is coarser in three cases and matches the old abstraction in the fourth case.

Lemma 4. *Let $\mathcal{A}$ be a timed automaton. Define the constants $M(x)$, $L(x)$ and $U(x)$ for each clock x as described before. The abstraction $\mathfrak{a}_{\prec LU}$ is sound, complete, and coarser or equal to $\mathfrak{a}_{\equiv_M}$.*

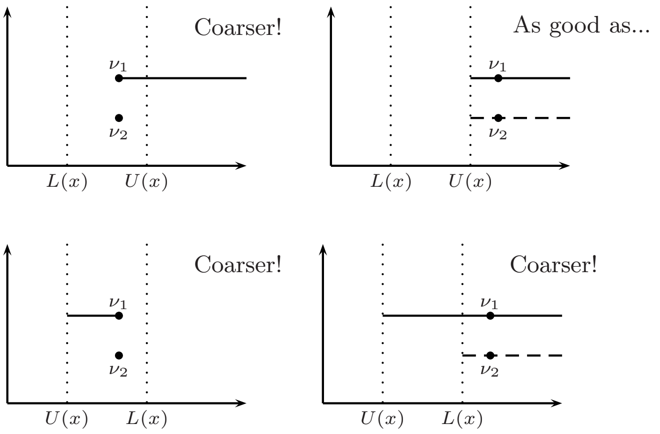


Fig. 2. Quality of $\mathfrak{a}_{\preccurlyeq_{LU}}$ compared with $\mathfrak{a}_{\equiv_M}$ for $M = \max(L, U)$.

Proof: Completeness is obvious, and soundness comes from lemma 3. Definitions of $\mathfrak{a}_{\prec LU}$ and $\mathfrak{a}_{\equiv_M}$ give the last result because for each clock, x, we have $M(x) = \max(L(x), U(x))$. ■

This result could suggest to use $\mathfrak{a}_{\preccurlyeq_{LU}}$ in real time model-checkers. However, we do not yet have an efficient method for computing the transition relation '$\Longrightarrow_{\mathfrak{a}_{\preccurlyeq_{LU}}}$'. Indeed, even if W is a zone, it might be the case that $\mathfrak{a}_{\preccurlyeq_{LU}}(W)$ is not even convex (we urge the reader to construct such an example for herself). For effectiveness and efficiency reasons we prefer abstractions which transform zones into zones because we can then use the DBM data structure. In the next section we present DBM-based extrapolation operators that will give abstractions which are sound, complete, finite and also effective.

4 Extrapolation Using Zones

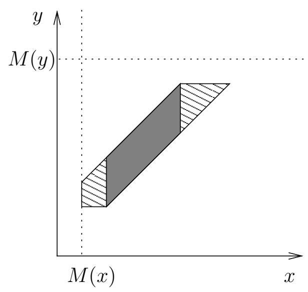
The (sound and complete) symbolic transition relations induced by abstractions considered so far unfortunately do not preserve convexity of sets of valuations. In order to allow for sets of valuations to be represented *efficiently* as zones, we consider slightly finer abstractions $\mathfrak{a}_{Extra}$ such that for every zone Z, $Z \subseteq \mathfrak{a}_{Extra}(Z) \subseteq \mathfrak{a}_{\preccurlyeq_{LU}}(Z)$ (resp. $Z \subseteq \mathfrak{a}_{Extra}(Z) \subseteq \mathfrak{a}_{\equiv_M}(Z)$) (this ensures correctness) and $\mathfrak{a}_{Extra}(Z)$ is a zone (this gives an effective representation). These abstractions are defined in terms of *extrapolation* operators on DBMs. If *Extra* is an extrapolation operator, it defines an abstraction, $\mathfrak{a}_{Extra}$, on zones such that for every zone Z, $\mathfrak{a}_{Extra}(Z) = [\![Extra(D_Z)]\!]$, where D_Z is the DBM in normal form which represents the zone Z.

In the remainder, we consider a timed automaton $\mathcal{A}$ over a set of clocks $X = \{x_1, .., x_n\}$ and we suppose we are given another clock x_0 which is always zero. For all these clocks, we define the constants $M(x_i)$, $L(x_i)$, $U(x_i)$ for $i = 1, ..., n$. For x_0, we set $M(x_0) = U(x_0) = L(x_0) = 0$ (x_0 is always equal to zero, so we assume we are able to check whether x_0 is really zero). In our framework, a zone will be represented by DBMs of the form $\langle c_{i,j}, \prec_{i,j} \rangle_{i,j=0,\ldots,n}$.

We now present several extrapolations starting from the classical one and improving it step by step. Each extrapolation will be illustrated by a small picture of a zone, shown in black, over two clocks, x and y, and its corresponding extrapolation, shown as a dashed area. In succession, the pitures illustrate the increasing coarseness of each extrapolation.

Classical extrapolation based on maximal bounds $M(x)$. If D be a DBM $\langle c_{i,j}, \prec_{i,j} \rangle_{i,j=0\ldots n}$, $Extra_M(D)$ is given by the DBM $\langle c'_{i,j}, \prec'_{i,j} \rangle_{i,j=0\ldots n}$ defined and illustrated below:

$$(c'_{i,j}, \prec'_{i,j}) = \begin{cases} \infty & \text{if } c_{i,j} > M(x_i) \\ (-M(x_j), <) & \text{if } -c_{i,j} > M(x_j) \\ (c_{i,j}, \prec_{i,j}) & \text{otherwise} \end{cases}$$

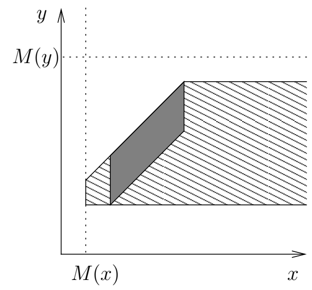


This is the extrapolation operator used in the real-time model-checkers UPPAAL and KRONOS. This extrapolation removes bounds that are larger than the maximal constants. The correctness follows from $\mathfrak{a}_{Extra_M}(Z) \subseteq \mathfrak{a}_{\equiv_M}(Z)$ and is proved in [Bou03] and for the location-based version in [BBFL03].

In the remainder, we will propose several other extrapolations that will improve the classical one, in the sense that the zones obtained with the new extrapolations will be larger than the zones obtained with the classical extrapolation.

Diagonal extrapolation based on maximal constants $M(x)$. The first improvement consists in noticing that if the whole zone is above the maximal bound of some clock, then we can remove some of the diagonal constraints of the zones, even if they are not themselves above the maximal bound. More formally, if $D = \langle c_{i,j}, \prec_{i,j} \rangle_{i,j=0,\ldots,n}$ is a DBM, $Extra^+_M(D)$ is given by $\langle c'_{i,j}, \prec'_{i,j} \rangle_{i,j=0,\ldots,n}$ defined as:

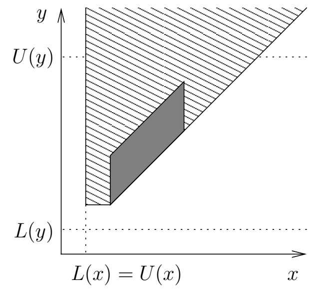
$$(c'_{i,j}, \prec'_{i,j}) = \begin{cases} \infty & \text{if } c_{i,j} > M(x_i) \\ \infty & \text{if } -c_{0,i} > M(x_i) \\ \infty & \text{if } -c_{0,j} > M(x_j), i \neq 0 \\ (-M(x_j), <) & \text{if } -c_{i,j} > M(x_j), i = 0 \\ (c_{i,j}, \prec_{i,j}) & \text{otherwise} \end{cases}$$



For every zone Z it then holds that $Z \subseteq \mathfrak{a}_{Extra_M}(Z) \subseteq \mathfrak{a}_{Extra^+_M}(Z)$.

Extrapolation based on LU-bounds $L(x)$ and $U(x)$. The second improvement uses the two bounds $L(x)$ and $U(x)$. If $D = \langle c_{i,j}, \prec_{i,j} \rangle_{i,j=0,\ldots,n}$ is a DBM, $Extra_{LU}(D)$ is given by $\langle c'_{i,j}, \prec'_{i,j} \rangle_{i,j=0,\ldots,n}$ defined as:

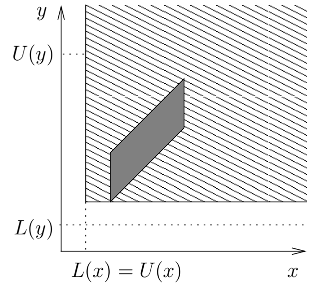
$$(c'_{i,j}, \prec'_{i,j}) = \begin{cases} \infty & \text{if } c_{i,j} > L(x_i) \\ (-U(x_j), <) & \text{if } -c_{i,j} > U(x_j) \\ (c_{i,j}, \prec_{i,j}) & \text{otherwise} \end{cases}$$



This extrapolation benefits from the properties of the two different maximal bounds. It does generalise the operator $\mathfrak{a}_{Extra_M}$. For every zone Z, it holds that $Z \subseteq \mathfrak{a}_{Extra_M}(Z) \subseteq \mathfrak{a}_{Extra_{LU}}(Z)$.

Diagonal extrapolation based on LU-bounds $L(x)$ and $U(x)$. This last extrapolation is a combination of the extrapolation based on LU-bounds and the improved extrapolation based on maximal constants. It is the most general one. If $D = \langle c_{i,j}, \prec_{i,j} \rangle_{i,j=0,\ldots,n}$ is a DBM, then $Extra^+_{LU}(D)$ is given by the DBM $\langle c'_{i,j}, \prec'_{i,j} \rangle_{i,j=0,\ldots,n}$ defined as:

$$(c'_{i,j}, \prec'_{i,j}) = \begin{cases} \infty & \text{if } c_{i,j} > L(x_i) \\ \infty & \text{if } -c_{0,i} > L(x_i) \\ \infty & \text{if } -c_{0,j} > U(x_j), i \neq 0 \\ (-U(x_j), <) & \text{if } -c_{0,j} > U(x_j), i = 0 \\ (c_{i,j}, \prec_{i,j}) & \text{otherwise} \end{cases}$$



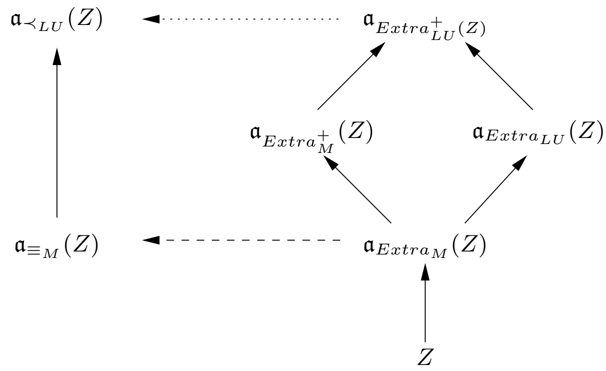


Fig. 3. For any zone Z, we have the inclusions indicated by the arrows. The sets $\mathfrak{a}_{Extra^+_M}(Z)$ and $\mathfrak{a}_{Extra_{LU}}(Z)$ are incomparable. The $\mathfrak{a}_{Extra}$ operators are DBM based abstractions whereas the other two are semantic abstractions. The dashed arrow was proved in [BBFL03] whereas the dotted arrow is the main result of this paper.

Correctness of these Abstractions. We know that all the above extrapolations are complete abstractions as they transform a zone into a clearly larger one. Finiteness also comes immediately, because we can do all the computations with DBMs and the coefficients after extrapolation can only take a finite number of values. Effectiveness of the abstraction is obvious as extrapolation operators are directly defined on the DBM data structure. The only difficult point is to prove that the extrapolations we have presented are correct. To prove the correctness of all these abstractions, due to the inclusions shown in Fig. 3, it is sufficient to prove the correctness of the largest abstraction, *viz* $\mathfrak{a}_{Extra^+_{LU}}$.

Proposition 2. *Let Z be a zone. Then $\mathfrak{a}_{Extra^+_{LU}}(Z) \subseteq \mathfrak{a}_{\preccurlyeq_{LU}}(Z)$.*

The proof of this proposition is quite technical. However, as it is a key result we provide a proof in the Appendix. Using all what precedes we are able to claim the following theorem which states that $\mathfrak{a}_{Extra^+_{LU}}$ is an abstraction which can be used in the implementation of timed automata.

Theorem 1. *$\mathfrak{a}_{Extra^+_{LU}}$ is sound, complete, finite and effectively computable.*

5 Acceleration of Successor Computation

In the preceding section it was shown that the abstraction based on the new extrapolation operator is coarser than the one currently used in timed automata model-checkers. This can result in a smaller symbolic representation of the state space of a timed automaton. As we will see in this section, besides reducing the memory requirements, identifying lower and upper bounded clocks can be used to speed up the successor computation.

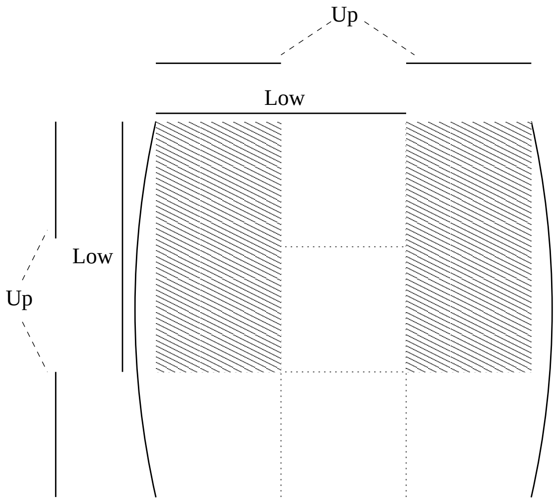


Fig. 4. DBM in LU-form. All DBM entries in the white areas are $+\infty$. DBMs in LU-form can be represented as an asymmetric DBM, in which only rows for lower bounded clocks and columns for upper bounded clocks are included.

In certain models some clocks only have lower bounds or only have upper bounds. We say that a clock x is *lower-bounded* (resp. *upper-bounded*) if $L(x) > -\infty$ (resp. $U(x) > -\infty$). Let D be a DBM and $D' = \mathit{Extra}^+_{LU}(D)$. It follows directly from the definition of the extrapolation operator that for all x_i, $U(x_i) = -\infty$ implies $c'_{j,i} = +\infty$ and $L(x_i) = -\infty$ implies $c'_{i,j} = +\infty$. We say that a DBM D is in *LU-form* whenever all coefficients $c_{i,j} = \infty$, except when x_i is lower bounded *and* x_j is upper bounded. Thus, D' is in LU-form. If we let $\mathit{Low} = \{i \mid x_i \text{ is lower bounded}\}$, $\mathit{Up} = \{i \mid x_i \text{ is upper bounded}\}$ and $\mathit{Clocks} = \mathit{Low} \cup \mathit{Up}$, then it follows that a DBM in LU-form can be represented by a $|\mathit{Low}| \times |\mathit{Up}|$ matrix rather the normal $|\mathit{Clocks}| \times |\mathit{Clocks}|$, since all remaining entries in the DBM will be $+\infty$, see Fig. 4.

The projection of a DBM onto the LU-form is given by the function LU-PROJECTION$(D) = D'$, where

$$D'_{ij} = \begin{cases} D_{ij} & i \in \mathit{Low}, j \in \mathit{Up} \\ \infty & \text{otherwise} \end{cases}$$

Lemma 5. *Let D be a DBM in normal form. Then* LU-PROJECTION(D) *is*

- *in normal form*
- *in LU-form*
- *reachability equivalent to D*

Proof: Canonicity is easy to show. The LU-form follows directly from the definition of the LU-form and the definition of the projection function. Reachability equivalence is a consequence of $[\![D]\!] \subseteq [\![\text{LU-PROJECTION}(D)]\!] \subseteq [\![\mathit{Extra}^+_{LU}(D)]\!] \subseteq \mathfrak{a}_{\preccurlyeq_{LU}}([\![D]\!])$. ∎

In the remainder of this section, we will alter the successor computation such that we retain during the entire successor computation. We first summarise how the DBM based successor computation is currently performed. Let D be a DBM **in normal form**. We want to compute the successor of D w.r.t. an edge $\ell \xrightarrow{g,Y} \ell'$. In UPPAAL, this is broken down into a number of elementary DBM operations, quite similar to the symbolic semantics of timed automata, see Table 1. After applying the guard and the target invariant, the result must be checked for consistency and after applying the extrapolation operator, the DBM must be brought back into normal form. Checking the consistency of a DBM is done by computing the normal form and checking the diagonal for negative entries. In general, the normal form can be computed using the $\mathcal{O}(n^3)$-time Floyd-Warshall all-pairs-shortest-path algorithm, but when applying a guard or invariant, resetting clocks, or computing the delay successors, the normal form can be recomputed much more efficiently [Rok93], see the left column of Fig. 5.[3] Table 1 shows the operations involved and their complexity (all DBMs except D_5 are in normal form). The last step is clearly the most expensive. As mentioned, with the new extrapolation operator D_5 is in LU-form. Using LU-PROJECTION, the successor computation can be changed such that all intermediate DBMs are in LU-form:

$$\begin{aligned}
D_1 &= \text{LU-PROJECTION}(\text{INTERSECTION}(g, D))\\
D_2 &= \text{LU-PROJECTION}(\text{RESET}_Y(D_1))\\
D_3 &= \text{LU-PROJECTION}(\text{ELAPSE}(D_2))\\
D_4 &= \text{LU-PROJECTION}(\text{INTERSECTION}(I(\ell), D_3))\\
D_5 &= \text{EXTRAPOLATION}(D_4)\\
D_6 &= \text{LU-PROJECTION}(\text{CANONIZE}(D_5))
\end{aligned}$$

The correctness of this change follows from Lemma 5. The final step in creating a more efficient successor computation is to replace the operations on the left of Fig. 5 with those on the right:

$$\begin{aligned}
D_1 &= \text{LU-INTERSECTION}(g, D)\\
D_2 &= \text{LU-RESET}_Y(D_1)\\
D_3 &= \text{LU-ELAPSE}(D_2)\\
D_4 &= \text{LU-INTERSECTION}(I(\ell), D_3)\\
D_5 &= \text{LU-EXTRAPOLATION}(D_4)\\
D_6 &= \text{LU-CANONIZE}(D_5)
\end{aligned}$$

The correctness of this change is ensured by the following lemmas.

Lemma 6. *Let D be a DBM in normal form and LU-form. Then we have the following syntactic equalities:*

$$\begin{aligned}
\text{LU-PROJECTION}(\text{INTERS}(g, D)) &= \text{LU-INTERS}(g, D)\\
\text{LU-PROJECTION}(\text{RESET}_Y(D)) &= \text{LU-RESET}_Y(D)\\
\text{ELAPSE}(D) &= \text{LU-ELAPSE}(D)
\end{aligned}$$

Proof: For all LU operations, we must argue, that the constraints in the LU-projection of the DBM are preserved.

[3] The pseudo code is in a procedural style rather than a functional style. The latter would require a copy of the input DBM.

```
proc TIGHTEN(D, x, y, v)
  D_xy := min(D_xy, v)
  Close1(D, x)
  Close1(D, y)
where
proc Close1(D, k)
  for i : Clocks do
      for j : Clocks do
          D_ij := min(D_ij, D_ik + D_kj)
      od od
end
```

```
proc LU-TIGTHEN(D, x, y, v)
  if x ∈ Low ∩ Up ∧ y ∈ Low ∩ Up
    then D_xy := min(D_xy, v)
         LU-Close1(D, x)
         LU-Close1(D, y) fi
  if x ∈ Low \ Up                    implies y = 0
    then for i : Low do
             D_i0 := min(D_i0, D_ix + v)
         od fi
  if y ∈ Up \ Low                    implies x = 0
    then for j : Up do
             D_0j := min(D_0j, v + D_yj)
         od fi
where
proc LU-Close1(D, k)
  for i : Low do
      for j : Up do
          D_ij := min(D_ij, D_ik + D_kj)
      od od
end
```

```
proc RESET(D, x, v)
  D_0x := −v
  D_x0 := v
  for y : Clocks^+ do
      D_xy := v + D_0y
      D_yx := D_y0 − v
  od
end
```

```
proc LU-RESET(D, x, v)
  if x ∈ Up then D_0x := −v fi
  if x ∈ Low then D_x0 := v fi
  if x ∈ Low
    then for y : Up^+ do
             D_xy := v + D_0y
         od fi
  if x ∈ Up
    then for y ∈ Low^+ do
             D_yx := D_y0 − v
         od fi
end
```

```
proc ELAPSE(D)
  for x : Clocks^+ do
      D_x0 := ∞
  od
end
```

```
proc LU-ELAPSE(D)
  for x : Low^+ do
      D_x0 := ∞
  od
end
```

```
proc CANONIZE(D)
  for k : Clocks do
      for i : Clocks do
          for j : Clocks do
              D_ij = min(D_ij, D_ik + D_kj)
          od od od
end
```

```
proc LU-CANONIZE(D)
  for k : Low ∩ Up do
      for i : Low do
          for j : Up do
              D_ij = min(D_ij, D_ik + D_kj)
          od od od
end
```

Fig. 5. On the left: Operations on symmetric DBMs for computing successors. On the right: The corresponding operations on asymmetric DBMs. All operations maintain the canonical form. Notice that our model does not allow guards over clock differences, hence one of the clocks given to TIGHTEN will be zero.

Operation	Symmetric	Asymmetric
$D_1 = \textsc{Intersection}(g, D)$	$\mathcal{O}(n^2 \cdot \lvert g \rvert)$	$\mathcal{O}(\lvert Low \rvert \cdot \lvert Up \rvert \cdot \lvert g \rvert)$
$D_2 = \textsc{Reset}_Y(D_1)$	$\mathcal{O}(n \cdot \lvert Y \rvert)$	$\mathcal{O}(\lvert Low \rvert \cdot \lvert Y \rvert + \lvert Up \rvert \cdot \lvert Y \rvert)$
$D_3 = \textsc{Elapse}(D_2)$	$\mathcal{O}(n)$	$\mathcal{O}(\lvert Low \rvert \cdot \lvert Y \rvert)$
$D_4 = \textsc{Intersection}(I(\ell), D_3)$	$\mathcal{O}(n^2 \cdot \lvert I(l) \rvert)$	$\mathcal{O}(\lvert Low \rvert \cdot \lvert Up \rvert \cdot \lvert I(l) \rvert)$
$D_5 = \textsc{Extrapolation}(D_4)$	$\mathcal{O}(n^2)$	$\mathcal{O}(\lvert Low \rvert \cdot \lvert Up \rvert)$
$D_6 = \textsc{Canonize}(D_5)$	$\mathcal{O}(n^3)$	$\mathcal{O}(\lvert Low \rvert \cdot \lvert Up \rvert \cdot \lvert Low \cap Up \rvert)$

Table 1. Steps in computing the zone of a symbolic successor. The Intersection operation computes the intersection of a DBM and a guard by repeatedly running the Tighten algorithm given in Fig. 5. The complexity of performing each step with symmetric and asymmetric DBMs is shown in the second and third column, respectively.

For the first equality we observe that LU-Tighten is always called for a guard $k \bowtie v$, where either x or y is k and the other is zero, since guards on clock differences are not allowed in our model. If k is both a lower bounded and upper bounded clock, then LU-Tighten behaves like Tighten except that LU-Close1 only iterates over lower and upper bounded clocks (for all other clocks, the sum in the loop-body is infinity). If k is lower bounded, but not upper bounded, then the D_{xy} is not in the DBM: In this case, the loop over the lower bounded clocks propagates the effect that tightening D_{xy} would have had had Canonize been called (the correctness of this can be observed by reordering how Canonize visits the clocks – then all iterations but the last do not modify the DBM). The case for k being upper bounded, but not lower bounded, is similar.

The second equality follows a similar observation for reset: For the constraints in the projection, LU-Reset behaves exactly like Reset. Changes to constraints not in the projection can be ignored, since they do not propagate to constraints in the projection.

The final equality follows from the observation that Elapse preserves the LU-form. ■

Lemma 7. *Let D be a DBM in LU-form. Then we have the following syntactic equality:*

$$\textsc{Canonize}(D) = \textsc{LU-Canonize}(D)$$

Proof: Follows from the observation that Canonize preserves the LU-form. ■

Table 1 gives the complexity of the new algorithms. As can be seen, the complexity of the new algorithms is bound by the number of lower and upper bounded clocks rather than the total number of clocks. For some models, and in particular in combination with the location-based abstraction of [BBFL03], we expect this change to provide a significant speedup. One such example is presented in the following section.

6 Job Shop Scheduling

Let $\mathfrak{M}$ be a finite set of machines. A *job* is a finite sequence of pairs $(m_1, d_1)...(m_k, d_k)$, where each m_i is a machine of $\mathfrak{M}$ and d_i is an integer representing a duration. A pair (m_i, d_i) means that the job needs the machine m_i for d_i units of time. The *jobshop scheduling problem* consists in scheduling a finite sets of jobs on the machines of $\mathfrak{M}$ (two jobs can not use the same machine at the same time) such that all jobs are completed in the shortest amount of time. As done in [AM01, AAM04], jobs can be represented by simple timed automata, see Fig. 6 for an example.

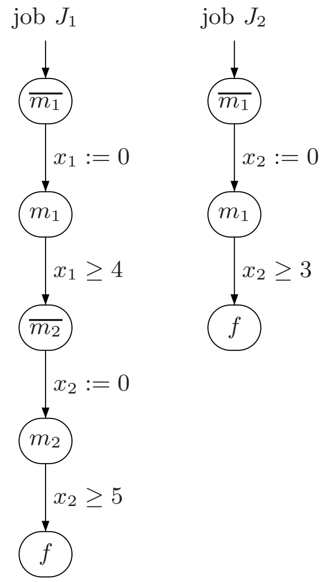
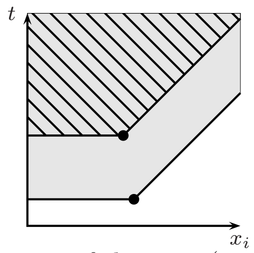


Fig. 6. Timed automata encoding of job shop scheduling problem with two jobs $J_1 = (m_1, 4)(m_2, 5)$ and $J_2 = (m_1, 3)$ and set of machines $\mathfrak{M} = \{m_1, m_2\}$. A state $\overline{m_i}$ means that the job is waiting for machine m_i to be free for doing some work. A state m_i means that machine m_i is working on the given job. By a cartesian product of these automata which forbids two jobs being together in a state m_i, we get all possible schedules for these jobs. The problem thus reduces to computing the minimal time for reaching state f of all jobs.

We first notice that all clocks x_i are only lower bounded, that is, $U(x_i) = -\infty$. We add a clock t which will represent universal time. This clock is never reset and never checked. However we want to be able to check that the time when reaching the configuration where all jobs are in state f is less than some given value (because we look for an optimal scheduling). Thus, we can say that clock

t is only upper bounded, thus that $L(t) = -\infty$. Assume now that we apply the forward computation to this system. We notice that no constraint $x_i \leq c$ for some constant c can be generated (see Table 1, there is no bounding invariant in the states and in the guards). Using what precedes, at each step of the forward computation, if M is the current DBM (after the 6 steps described in Table 1), only the following coefficients are different from $+\infty$: $m_{x_i,t}$ for each i and $m_{0,t}$. Notice that we thus only need to store $(n+1)$ coefficients (where n is the number of jobs) at each step of the computation, instead of $(n+1)^2$ as is usually the case.

What we will see now is that the inclusion checking with these DBMs in LU-form is more general than the domination test presented in [AM01, AAM04]. The idea of the dominating point is the following: if (t, v) and (t', v') are two valuations, then we say that (t, v) is better than (t', v') if $t \leq t'$ and $v \geq v'$. This means that all possible runs from (t', v') can also be done from (t, v). The zones (after LU-extrapolation) we compute can be represented as follows (when projecting on the plan (t, x_i)):



The domination point of the zone (as defined in [AM01, AAM04]) is represented on the previous picture by a bullet. The valuation v corresponding to this domination point is such that $v(x_i) = m_{x_i,t} - m_{0,t}$ for every i and $v(t) = -m_{0,t}$. However notice that this point is not better than any point in the zone but is better than any point with a minimal time (clock t). Given two such zones Z and Z', the domination test as done in [AM01, AAM04] is then equivalent to checking that:

$$\begin{cases} -m_{0,t} \leq -m'_{0,t} \\ m_{x_i,t} - m_{0,t} \geq m'_{x_i,t} - m'_{0,t} \quad \text{for any } i \end{cases} \tag{dp}$$

In contrast, checking inclusion of Z' into Z is equivalent to checking that:

$$\begin{cases} m'_{0,t} \leq m_{0,t} \\ m'_{x_i,t} \leq m_{x_i,t} \quad \text{for any } i \end{cases} \tag{inc}$$

One can easily check that equations (dp) imply equations (inc). We have thus that the domination test implies the inclusion checking. Checking inclusion with this new extrapolation is thus more general than the domination test of [AM01, AAM04]. Intuitively, if Z' is included in Z then the domination points (t, v) of Z and (t', v') of Z' are such that $(t, v) + t' - t \geq (t', v')$ which also implies that all possible paths from (t', v') are also possible from (t, v) after having waited $t' - t$ units of time (thus (t, v) is somehow also "better" than (t', v')).

7 Implementation & Experiments

We have implemented a prototype of a location based variant of the $Extra^+_{LU}$ operator in Uppaal 3.4.2. Maximum lower and upper bounds for clocks are found for each automaton using a simple fixed point iteration. Given a location vector, the maximum lower and upper bounds are found by taking the maximum of the bounds in each location, similar to the approach taken in [BBFL03]. At the moment we have only implemented the LU-Canonize operation on top of a symmetric DBM representation. For storing visited states, we rely on the *minimal constraint form* representation of a zone described in [LLPY97], which does not store $+\infty$ entries.

As expected, experiments with the model in Fig. 1 show that with LU extrapolation, the computation time for building the complete reachable state space does not depend on the value of the constants, whereas the computation time grows with the constant when using the classical extrapolation. We have also performed experiments with models of various instances of the CSMA/CD protocol and Fischer's protocol for mutual exclusion. Finally, experiments using a number of industrial case studies were made. For each model, Uppaal was run with four different options: (-n1) classic non-location based extrapolation (without active clock reduction), (-n2) classic location based extrapolation (active clock reduction is a side-effect of this), (-n3) LU location based extrapolation, and (-A) classic location based extrapolation with convex-hull approximation. In all experiments the minimal constraint form for zone representation was used [LLPY97] and the complete state space was generated. All experiments were performed on a 1.8GHz Pentium 4 running Linux 2.4.22, and experiments were limited to 15 minutes of CPU time and 470MB of memory. The results can be seen in Table 2.

Looking at the table, we see that for both Fischer's protocol for mutual exclusion and the CSMA/CD protocol, Uppaal scales considerably better with the LU extrapolation operator. Comparing it with the convex hull approximation (which is an over-approximation), we see that for these models, the LU extrapolation operator comes close to the same speed, although it still generates more states. Also notice that the runs with the LU extrapolation operator use less memory than convex hull approximation, due to the fact that in the latter case DBMs are used to represent the convex hull of the zones involved (in contrast to using the minimal constraint form of [LLPY97]). For the three industrial examples, the speedup is less dramatic: These models have a more complex control structure and thus little can be gained from changing the extrapolation operator. This is supported by the fact that also the convex hull technique fails to give any significant speedup (in the last example it even degrades performance). During the course of our experiments we also encountered examples where the LU extrapolation operator does not make any difference: the

Model	-n1			-n2			-n3			-A		
	Time	States	Mem	Time	States	Mem	Time	States	Mem	Time	States	Mem
f5	4.02	82,685	5	0.24	16,980	3	0.03	2,870	3	0.03	3,650	3
f6	597.04	1,489,230	49	6.67	158,220	7	0.11	11,484	3	0.10	14,658	3
f7				352.67	1,620,542	46	0.47	44,142	3	0.45	56,252	5
f8							2.11	164,528	6	2.08	208,744	12
f9							8.76	598,662	19	9.11	754,974	39
f10							37.26	2,136,980	68	39.13	2,676,150	143
f11							152.44	7,510,382	268			
c5	0.55	27,174	3	0.14	10,569	3	0.02	2,027	3	0.03	1,651	3
c6	19.39	287,109	11	3.63	87,977	5	0.10	6,296	3	0.06	4,986	3
c7				195.35	813,924	29	0.28	18,205	3	0.22	14,101	4
c8							0.98	50,058	5	0.66	38,060	7
c9							2.90	132,623	12	1.89	99,215	17
c10							8.42	341,452	29	5.48	251,758	49
c11							24.13	859,265	76	15.66	625,225	138
c12							68.20	2,122,286	202	43.10	1,525,536	394
bus	102.28	6,727,443	303	66.54	4,620,666	254	62.01	4,317,920	246	45.08	3,826,742	324
philips	0.16	12,823	3	0.09	6,763	3	0.09	6,599	3	0.07	5,992	3
sched	17.01	929,726	76	15.09	700,917	58	12.85	619,351	52	55.41	3,636,576	427

Table 2. Results for Fischer protocol (f), CSMA/CD (c), a model of a buscoupler, the Philips Audio protocol, and a model of a 5 task fixed-priority preemptive scheduler. -n1 is with classical maximum bounds extrapolation, -n2 is with location based maximum bounds extrapolation, -n3 is with location based LU extrapolation, and -A is with convex hull over-approximation. Times are in seconds, states are the number of generated states and memory usage is in MB.

token ring FDDI protocol and the B&O protocols found on the Uppaal website[4] are among these. Finally, we made a few experiments on Fischer's protocol with the LU extrapolation, but without the LU-Canonize operator. This showed that LU-Canonize gives a speedup in the order of 20% compared to Canonize.

As a result of the superior performance of the LU-extrapolation, this abstraction technique will be used as the default option in the forth-coming verions 3.6 of Uppaal.

8 Remarks and Conclusions

In this paper we extend the *status quo* of timed automata abstractions by contributing several new abstractions. In particular, we proposed a new extrapolation operator distinguishing between guards giving an upper bound to a clock and guards giving a lower bound to a clock. The improvement of the usual extrapolation is orthogonal to the location-based one proposed in [BBFL03] in the sense that they can be easily combined. We prove that the new abstraction is sound and complete w.r.t. reachability, and is finite and effectively computable. We implemented the new extrapolation in Uppaal and a new operator for computing the normal form of a DBM. The prototype showed significant improvements in verification speed, memory consumption and scalability for a number of models.

For further work, we aim at implementing and experimentally evaluate the technique of Section 5 for accelerating successor computation by exploiting an asymmetric DBM using an $n \times m$ matrix, where n is the number of lower bounded clocks and m is the number of upper bounded clocks. We expect this to significantly improve the successor computation for some models.

As we have indicated in Section 6, job shop scheduling problems may in particular benefit from the acceleration technique as all clocks of the corresponding automaton are without upper bounds, with the exception of one clock (the clock measuring global time), which lacks lower bounds . Thus, an asymetric DBM representation for such systems will have a size linear in the number of clocks. As we have also demonstrated the inclusion checking done on LU extrapolated zones turns out to be more general than the dominating point checki in [AM01].We need to investigate to what extent a generic timed automaton reachability checker using LU extrapolation can compete with the problem specific implementation in [AM01].

Acknowledgement

The authors would like to thank Emmanuel Fluery for many inspiring descussions on the topics of abstractions

[4] `http://www.uppaal.com`

for timed automata. Also the authors would like to thank the useful comments made by the anonymous referees.

References

[AAM04] Yasmina Abdeddaim, Eugene Asarin, and Oded Maler. Scheduling with timed automata. Submitted to TCS, 2004.

[AD90] Rajeev Alur and David Dill. Automata for modeling real-time systems. In *Proc. 17th International Colloquium on Automata, Languages and Programming (ICALP'90)*, volume 443 of *Lecture Notes in Computer Science*, pages 322–335. Springer, 1990.

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

[AM01] Yasmina Abdeddaim and Oded Maler. Job-shop scheduling using timed automata. In *Proc. 13th International Conference on Computer Aided Verification (CAV'01)*, volume 2102 of *Lecture Notes in Computer Science*, pages 478–492. Springer, 2001.

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

[BBLP04] Gerd Behrmann, Patricia Bouyer, Kim G. Larsen, and Radek Pelanek. Lower and upper bounds in zone based abstractions of timed automata. In *Proc. 10th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'2004)*, volume 2988 of *Lecture Notes in Computer Science*, pages 312–326. Springer, 2004.

[BDM+98] Marius Bozga, Conrado Daws, Oded Maler, Alfredo Olivero, Stavros Tripakis, and Sergio Yovine. KRONOS: a model-checking tool for real-time systems. In *Proc. 10th International Conference on Computer Aided Verification (CAV'98)*, volume 1427 of *Lecture Notes in Computer Science*, pages 546–550. Springer, 1998.

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

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

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

[BY03] Johan Bengtsson and Wang Yi. On clock difference constraints and termination in reachability analysis of timed automata. In J. S. Dong and J. Woodcock, editors, *Proc. of ICFEM'03*, number 2885 in Lecture Notes in Computer Science. Springer–Verlag, 2003.

[CGP99] Edmund Clarke, Orna Grumberg, and Doron Peled. *Model-Checking*. The MIT Press, Cambridge, Massachusetts, 1999.

[Dil89] David Dill. Timing assumptions and verification of finite-state concurrent systems. In *Proc. of the Workshop on Automatic Verification Methods for Finite State Systems*, volume 407 of *Lecture Notes in Computer Science*, pages 197–212. Springer, 1989.

[DT98] Conrado Daws and Stavros Tripakis. Model-checking of real-time reachability properties using abstractions. In *Proc. 4th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'98)*, volume 1384 of *Lecture Notes in Computer Science*, pages 313–329. Springer, 1998.

[LLPY97] Kim G. Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. Efficient verification of real-time systems: Compact data structure and state-space reduction. In *Proc. 18th IEEE Real-Time Systems Symposium (RTSS'97)*, pages 14–24. IEEE Computer Society Press, 1997.

[LPY97] Kim G. Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a nutshell. *Journal of Software Tools for Technology Transfer (STTT)*, 1(1–2):134–152, 1997.

[Mil71] Robin Milner. An algebraic definition of simulation between programs. In *Proceedings of 2nd International Conference on Artificial Intelligence*. British Com. Soc., 1971.

[Rok93] Tomas G. Rokicki. *Representing and Modeling Digital Circuits*. PhD thesis, Stanford University, Stanford, USA, 1993.

[TARB96] S. Tasiran, R. Alur, R.P.Kurshan, and R.K. Brayton. Verifying abstractions of timed systems. In *Proceedings of CONCUR 1996*, volume 1119 of *Lecture Notes in Computer Science*. Springer Verlag, 1996.

Technical Proofs

This appendix presents the proof of proposition 2. It is quite technical but we will detail it because it is fundamental. Let us recall the statement of the proposition we want to prove.

Proposition 2. *Let Z be a zone. Then $\mathfrak{a}_{Extra^+_{LU}}(Z) \subseteq \mathfrak{a}_{\preccurlyeq_{LU}}(Z)$.*

Proof: Let $D = \langle c_{i,j};\ \prec_{i,j}\rangle_{i,j=0\dots n}$ be a DBM in normal form representing a non-empty zone. We note $D' = \langle c'_{i,j};\ \prec'_{i,j}\rangle_{i,j=0\dots n}$ be the DBM $Extra^+_{LU}(D)$. Let us fix $\nu \in [\![Extra^+_{LU}(D)]\!]$. We want to prove that $\nu \in \mathfrak{a}_{\preccurlyeq_{LU}}([\![D]\!])$. For this, we define the set P_ν as $\{\nu' \in [\![D]\!] \mid \nu' \preccurlyeq_{LU} \nu\}$, and we will prove that P_ν is not empty. This will be sufficient as, by definition of $\mathfrak{a}_{\preccurlyeq_{LU}}$, we have $\nu \in$

$\mathfrak{a}_{\preccurlyeq_{LU}}(\llbracket D \rrbracket) \iff P_\nu \neq \emptyset$. The set P_ν is defined by the constraints:

$$\begin{aligned}&\{x_i - x_j \prec_{i,j} c_{i,j} \mid i,j \in \{0,1,...,n\}\}\\ \cup\ &\{x_i > L(x_i) \mid \nu(x_i) > L(x_i) \text{ and } i \in \{1,...,n\}\}\\ \cup\ &\{x_i \le \nu(x_i) \mid \nu(x_i) \le U(x_i) \text{ and } i \in \{1,...,n\}\}\\ \cup\ &\{x_i \ge \nu(x_i) \mid \nu(x_i) \le L(x_i) \text{ and } i \in \{1,...,n\}\}\end{aligned}$$

The first set of constraints represents the constraints of D. The other lines represent the constraints due to $\cdot \preccurlyeq_{LU} \nu$. Indeed, it is easy to check that for each i, the constraint $(x_i = \nu(x_i)) \vee (L(x_i) < x_i < \nu(x_i)) \vee (U(x_i) < \nu(x_i) < x_i)$ which is the direct definition of $\preccurlyeq_{LU}$ is equivalent to the constraint $(\nu(x_i) > L(x_i) \implies x_i > L(x_i)) \wedge (\nu(x_i) \le U(x_i) \implies x \le \nu(x_i)) \wedge (\nu(x_i) \le L(x_i) \implies x_i \ge \nu(x_i))$. The three last lines above correspond to this set of constraints.

We will simplify the constraints defining P_ν For this, we need the following three lemmas.

Lemma 8. *If $c_{j,0} < +\infty$, then $(c'_{j,0}, \prec'_{j,0}) = \infty$ implies $c_{j,0} > L(x_j)$.*

Lemma 9. *$(c'_{0,i}, \prec'_{0,i}) \neq (c_{0,i}, \prec_{0,i})$ implies $-c_{0,i} > U(x_i)$ (and $(c'_{0,i}, \prec'_{0,i}) = (-U(x_i), <)$).*

Lemma 10. *Let $\nu \in \llbracket Extra^+_{LU}(D) \rrbracket$. Then*

1. *If $\nu(x_i) \le min\{U(x_i), L(x_i)\}$, then $\nu(x_i) \prec_{i,0} c_{i,0}$ and thus $(\nu(x_i), \le) \le (c_{i,0}, \prec_{i,0})$.*
2. *If $\nu(x_i) \le min\{L(x_i), U(x_i)\}$, then $-c_{0,i} \prec_{0,i} \nu(x_i)$ and therefore $(-\nu(x_i), \le) \le (c_{0,i}, \prec_{0,i})$.*

Proof:

1. If $c_{i,0} > L(x_i)$, then we have $\nu(x_i) \prec_{i,0} c_{i,0}$. If $c_{i,0} \le L(x_i)$, then $(c'_{i,0}, \prec'_{i,0}) = (c_{i,0}, \prec_{i,0})$ and we are done for the first inequality.
2. If $-c_{0,i} > U(x_i)$, then it is not possible as $(c'_{0,i}, \prec'_{0,i}) = (-U(x_i), <)$. Otherwise, $(c'_{0,i}, \prec'_{0,i}) = (c_{0,i}, \prec_{0,i})$. Thus we are also done for the second inequality.

■

Applying the previous lemmas, we get that P_ν is represented by the DBM $\langle p_{i,j}, \subset_{i,j} \rangle_{i,j=0...n}$ where $(p_{i,0}, \subset_{i,0}) =$

$$\begin{cases}(\nu(x_i), \le) & \text{if } \nu(x_i) \le L(x_i), U(x_i)\\ \min((\nu(x_i), \le), (c_{i,0}, \prec_{i,0})) & \text{if } L(x_i) < \nu(x_i) \le U(x_i)\\ (c_{i,0}, \prec_{i,0}) & \text{if } \nu(x_i) > U(x_i)\end{cases}$$

and $(p_{0,i}, \subset_{0,i}) =$

$$\begin{cases}(-\nu(x_i), \le) & \text{if } \nu(x_i) \le L(x_i), U(x_i)\\ \min((c_{0,i}, \prec_{0,i}), (-\nu(x_i), \le)) & \text{if } U(x_i) < \nu(x_i) \le L(x_i)\\ \min((c_{0,i}, \prec_{0,i}), (-L(x_i), <)) & \text{if } \nu(x_i) > L(x_i)\end{cases}$$

and $(p_{i,j}, \subset_{i,j}) = (c_{i,j}, \prec_{i,j})$ if $i, j \neq 0$.

We need to prove that P_ν is non empty. If it is not the case, it means that there is a negative cycle in any DBM representing P_ν. As the above DBM only differs from D (which is non-empty and in normal form) by coefficients $(i, 0)$ and $(0, i)$ (for all i's), we get that there exist some i and j (with potentially $i = j$) such that:

$$(p_{i,0}, \subset_{i,0}) + (p_{0,j}, \subset_{0,j}) + (c_{j,i}, \prec_{j,i}) < (0, \le) \qquad (1)$$

We want to prove that this is not possible. We will have to distinguish several cases, depending on the values of $p_{i,0}$ and $p_{0,j}$.

Case $(p_{i,0}, \subset_{i,0}) = (c_{i,0}, \prec_{i,0})$**.**

We can simplify inequality (1) by applying the triangular inequality and we get that

$$(c_{j,0}, \prec_{j,0}) + (p_{0,j}, \subset_{0,j}) < (0, \le)$$

1. *Case* $(p_{0,j}, \subset_{0,j}) = (c_{0,j}, \prec_{0,j})$.
   In this case, we get
   $$(c_{j,0}, \prec_{j,0}) + (c_{0,j}, \prec_{0,j}) < (0, \le)$$
   which implies that D is empty. Contradiction.
2. *Case* $(p_{0,j}, \subset_{0,j}) = (-\nu(x_j), \le)$. (hyp: $\nu(x_j) \le L(x_j)$)
   We then get that
   $$(c_{j,0}, \prec_{j,0}) + (-\nu(x_j), \le) < (0, \le)$$
   which implies that $\nu(x_j) \not\prec_{j,0} c_{j,0}$. In particular we have, $(c'_{j,0}, \prec'_{j,0}) > (c_{j,0}, \prec_{j,0})$, which is possible only if $c_{j,0} > L(x_j)$ (see Lemma 8). However, in this case, we have that $\nu(x_j) \le L(x_j)$, which is a contradiction.
3. *Case* $(p_{0,j}, \subset_{0,j}) = (-L(x_j), <)$. (hyp: $(-L(x_j), <) \le (c_{0,j}, \prec_{0,j})$ and $\nu(x_j) > L(x_j)$)
   We get that
   $$(c_{j,0}, \prec_{j,0}) + (-L(x_j), <) < (0, \le)$$
   and thus that
   $$c_{j,0} \le L(x_j) < \nu(x_j)$$
   As $c_{j,0} \le L(x_j)$, we get that $(c'_{j,0}, \prec'_{j,0}) = (c_{j,0}, \prec_{j,0})$ and thus there is a contradiction (because $\nu(x_j) \prec'_{j,0} c'_{j,0}$).

Case $(p_{i,0}, \subset_{i,0}) = (\nu(x_i), \le)$ **(hyp:** $\nu(x_i) \le U(x_i)$**).**

In this case, we get that

$$(\nu(x_i), \le) + (p_{0,j}, \subset_{0,j}) + (c_{j,i}, \prec_{j,i}) < (0, \le)$$

1. *Case* $(p_{0,j}, \subset_{0,j}) = (c_{0,j}, \prec_{0,j})$. (hyp not used)
   Using the triangular inequality, we can simplify and we get
   $$(\nu(x_i), \le) + (c_{0,i}, \prec_{0,i}) < (0, \le)$$
   If $(c_{0,i}, \prec_{0,i}) = (c'_{0,i}, \prec'_{0,i})$, this is not possible. Otherwise, $-c_{0,i} > U(x_i)$ and $(c'_{0,i}, \prec'_{0,i}) = (-U(x_i), <)$ (see Lemma 9). Thus $\nu(x_i) > U(x_i)$, which is indeed a contradiction.

2. *Case* $(p_{0,j}, \subset_{0,j}) = (-\nu(x_j), \leq)$. (hyp: $\nu(x_j) \leq L(x_j)$)
   We get that

   $$(\nu(x_i) - \nu(x_j), \leq) + (c_{j,i}, \prec_{j,i}) < (0, \leq)$$

   If $(c'_{j,i}, \prec'_{j,i}) = (c_{j,i}, \prec_{j,i})$, then this is not possible. Thus, $\infty = (c'_{j,i}, \prec'_{j,i}) > (c_{j,i}, \prec_{j,i})$ and there are three cases:
   - $c_{j,i} > L(x_j)$, thus $(c_{j,i}, \prec_{j,i}) > (\nu(x_j), \leq)$ which implies that $(\nu(x_i), \leq) < (0, \leq)$, impossible. We thus assume that $c_{j,i} \leq L(x_j)$.
   - $-c_{0,j} > L(x_j)$, thus $-c_{0,j} > \nu(x_j)$. From Lemma 10 (point *2.*), this is not possible (as $\nu(x_j) \leq L(x_j)$) if $\nu(x_j) \leq U(x_j)$. Assume thus that $\nu(x_j) > U(x_j)$. In this case, $-\nu(x_j) \leq c_{0,j}$, *i.e.* $\nu(x_j) \geq -c_{0,j}$, which leads to a contradiction.
   - $-c_{0,i} > U(x_i)$, contradiction with $\nu(x_i) \leq U(x_i)$

3. *Case* $(p_{0,j}, \subset_{0,j}) = (-L(x_j), <)$. (hyp: $\nu(x_j) > L(x_j)$ and $(c_{0,j}, \prec_{0,j}) \geq (-L(x_j), <)$)
   We get that

   $$(\nu(x_i), \leq) + (-L(x_j), <) + (c_{j,i}, \prec_{j,i}) < (0, \leq)$$

   If $(c_{j,i}, \prec_{j,i}) > (L(x_j), <)$, then we get $(\nu(x_i), \leq) < (0, \leq)$ which is not possible. Assume now that $(c_{j,i}, \prec_{j,i}) \leq (L(x_j), <)$. If $(c'_{j,i}, \prec'_{j,i}) = (c_{j,i}, \prec_{j,i})$, then

   $$(\nu(x_i) - \nu(x_j), \leq) + (c'_{j,i}, \prec'_{j,i}) < (0, \leq)$$

   This is not possible. The only possibility is thus to have $\infty = (c'_{j,i}, \prec'_{j,i}) > (c_{j,i}. \prec_{j,i})$. Can we have $-c_{0,j} > L(x_j)$ or $-c_{0,i} > U(x_i)$? By hypothesis, the first case is not possible. If $-c_{0,i} > U(x_i)$, then $(c'_{0,i}, \prec'_{0,i}) = (-U(x_i), <)$ which contradicts the fact that $\nu(x_i) \leq U(x_i)$.

In all cases, there is a contradiction with inequality 1, P_ν is thus non empty. This concludes the proof of proposition 2. ■