Model Checking Timed Automata with Priorities using DBM Subtraction

Alexandre David[1], John Håkansson[2], Kim G. Larsen[1], and Paul Pettersson[2]

[1] Department of Computer Science, Aalborg University, Denmark
{adavid,kgl}@cs.aau.dk.
[2] Department of Information Technology, Uppsala University, Sweden
{johnh,paupet}@it.uu.se.

Abstract. In this paper we describe an extension of timed automata with priorities, and efficient algorithms to compute subtraction on DBMs (difference bounded matrices), needed in symbolic model-checking of timed automata with priorities. The subtraction is one of the few operations on DBMs that result in a non-convex set needing *sets* of DBMs for representation. Our subtraction algorithms are efficient in the sense that the number of generated DBMs is significantly reduced compared to a naive algorithm. The overhead in time is compensated by the gain from reducing the number of resulting DBMs since this number affects the performance of symbolic model-checking. The uses of the DBM subtraction operation extend beyond timed automata with priorities. It is also useful for allowing guards on transitions with urgent actions, deadlock checking, and timed games.

1   Introduction

Since the introduction of timed automata [2] in 1990, the theory has proven its capability of specifying and analysing timed systems in many case studies, e.g., [4, 23]. To support such studies, tools as Kronos [7], UPPAAL [18], and RED [24] have been developed to offer means for modelling, simulation, model-checking, and also testing, of real-time systems specified as timed automata.

In the implementation of real-time systems, the concept of *priorities* is often used as a way to structure and control the usage of shared resources. Priorities are often associated with processes (or tasks) to control their usage of shared resources such as CPU or shared memory areas. As a consequence, programming languages such as Ada [3, 12], and scheduling policies used in real-time operating system, such as *rate-monotonic scheduling* [9], are often based on a notion of priorities on tasks. In lower levels, closer to the hardware, priorities are often associated with interrupts to hardware devices and access to e.g., shared communication buses.

Priorities have been studied in process algebras, e.g., [11, 8], and can be modelled using timed automata [12, 14]. However, it can be cumbersome and error-prone to do so. Consider the simple example shown in Figure 1 and assume that the location l can be reached with any time assignment satisfying the constraint
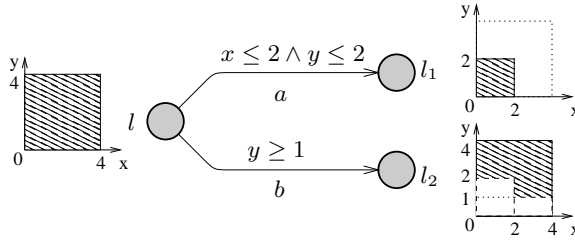
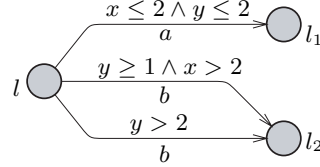**Fig. 1.** A timed automaton with priorities on actions.

Fig. 2. Encoding of the priorities in Fig. 1.

$x \leq 4 \wedge y \leq 4$. Further assume that the edge labelled with a has priority over the edge labelled b. We see that l_1 can be reached with any time assignment satisfying the constraint $x \leq 2 \wedge y \leq 2$. The location l_2 is reachable under the constraint $(y \geq 1 \wedge x \leq 4 \wedge y \leq 4) \wedge \neg(x \leq 2 \wedge y \leq 2)$, which is a non-convex set of clock valuations and thus not representable as a conjunction of simple constraints. This fact will make (symbolic) state-space exploration potentially costly, since the set of clock valuations reachable in one step over a low priority transition, such as transitions derived from the b-edge, generally will have to be represented by *a set* of convex constraint systems. In Figure 2 we show a timed automaton in which the priorities of the automaton in Figure 1 have been encoded. Note that the b-edge has been split to two edges to encode the disjunctive constraints on the clock valuations reaching l_2.

Model-checking tools for timed automata typically uses DBMs (difference bound matrices) [13, 22] to represent convex constraints on clock variables. However, as illustrated above, analysis of timed automata with priorities will require the model-checking engine to efficiently handle disjunctive constraints. As a second contribution of this paper, we present a variety of techniques for performing *subtractions* on DBMs. That is, how to compute $D - D'$ defined as $D \wedge \neg D'$, for two DBMs D and D'. Guided by the goal to minimise the set of DBMs resulting from subtraction, and to keep them disjoint, we give a heuristic algorithm with good performance. To back up this statement, we present experimental evidence from applying a version of the UPPAAL tool extended with priorities, on a set of examples. We note that DBM subtraction is already needed for backward model-checking of full TCTL or scheduler synthesis [23], controller synthesis [10], and to support urgent guards.

The rest of this paper is organised as follow: Timed automata with priorities are described in Section 3, and the required DBM subtraction operation in Section 4. In Section 5 we present subtraction algorithms that reduce the set

of resulting DBMs. We show with experiments in Section 6 that our algorithm improves DBM subtractions significantly.

Related work: Priorities in process algebras are described in [11], where priorities on actions are defined in two levels. A process algebra of communicating shared processes is described in [8], where priorities are described as real numbers on events and timed actions. Preservation of congruence is a major concern in these papers.

In [6] priorities are introduced for *live* systems (time-lock and deadlock free with no indefinite waiting), with the purpose to preserve liveness in the composition of such systems. In our work we have focused on introducing priorities for existing timed automata models, and developing efficient algorithms for DBM subtraction.

In [16] a notion of priorities for timed automata based on total orderings and an algorithm for computing DBM subtractions have been proposed. In this paper, we introduce a more general notion of priorities for timed automata where the priority ordering is allowed to be partial, and allowing the priority ordering to be defined both on the level of synchronisation actions as well as the individual automata. We believe that both these two suggestions will be useful when modelling real-time systems with priorities. The subtraction of [16] is claimed to be optimal, i.e., it generates the fewest possible number of DBMs as a result. However, the ordering of constraining operations needed for subtraction is not mentioned. We argue in this paper that ordering is important and optimality of subtraction w.r.t. reachability is more difficult than just having the minimal number of DBMs from subtractions.

2 Preliminaries

2.1 Clock Constraints

Model-checking of timed automata involves exploring a state-space of symbolic states, where each symbolic state represents a set of clock valuations. For a set C of n clocks, a clock valuation is a map $v : C \mapsto \mathbb{R}_{\geq 0}$. We denote by $\mathcal{B}(C)$ the set of conjunctions of atomic constraints in the form $x_i \sim m$ or $x_i - x_j \sim m$, where m is a natural number, x_i and x_j are clock valuations of clocks i and j, and $\sim \in \{<, \leq, =, \geq, >\}$. Although it is possible to represent sets of clock valuations as regions [2], using zones is much more efficient in practice [5]. A *zone* corresponds to the set of clock valuations that satisfies a conjunction of constraints in $\mathcal{B}(C)$. A zone is convex by definition, and we represent it as a difference bound matrix (DBM).

2.2 Difference Bound Matrices

A DBM is a conjunction $D = \bigwedge_{1 \leq i,j \leq n}(x_i - x_j \sim b_{ij})$ for $\sim \in \{<, \leq\}$, written as $D = \bigwedge d_{ij}$. We use d_{ij} (or e_{ij}) to denote the constraints of a DBM D (or E).

The bound of a constraint d_{ij} is denoted $|d_{ij}|$. We define a *complement* operation over $\sim$ so that $\bar{\leq} = <$ and $\bar{<} = \leq$.

A DBM is *canonical* if it is closed under entailment, e.g. by Floyd's shortest path algorithm [15]. We consider all DBMs to be canonical.

Definition 1 (Operations on constraints) For constraints d_{ij} and e_{ij}:

- $d_{ij} \leq e_{ij} \Leftrightarrow d_{ij} \Rightarrow e_{ij}$
- $d_{ij} < e_{ij} \Leftrightarrow d_{ij} \neq e_{ij} \wedge d_{ij} \leq e_{ij}$
- $\neg d_{ij} = \neg(x_i - x_j \sim b_{ij}) = x_j - x_i \bar{\sim} - b_{ij}$. Note that $\neg d_{ij}$ is a new d'_{ji} comparable with other constraints e_{ji}.
- $d_{ik} + d_{kj} = (x_i - x_k \sim b_{ik}) + (x_k - x_j \sim' b_{kj}) = x_i - x_j \sim'' b_{ik} + b_{kj}$ where $\sim'' = <$ if $\sim = <$ or $\sim' = <$, otherwise $\sim'' = \leq$.
- $d_{ik} - d_{jk} = d_{ik} + \neg d_{jk}$

We write $v \models g$ to denote that a constraint $g \in \mathcal{B}(C)$ is satisfied by a clock valuation v. The notation $v \oplus d$ represents a valuation where all clocks have advanced by the real valued delay d from their value in v. For a set of clocks r we denote by $[r \mapsto 0]v$ the valuation that maps clocks in r to zero, and agrees with v for all other clocks. We write $D = \{v \mid v \models x_i - x_j \sim b_{ij}\}$ for the set of clock valuations that satisfy the constraints of D.

Definition 2 (Operations on zones) For a zone D and a clock constraint g:

- *conjunction*: $D \wedge g = \{v \mid v \in D, v \models g\}$,
- *delay*: $D^{\uparrow} = \{v \oplus t \mid v \in D, t \in \mathbb{R}_{\geq 0}\}$,
- *reset*: $r(D) = \{[r \mapsto 0]v \mid v \in D\}$,
- *free*: $free(D, r) = \{[r \mapsto t]v \mid v \in D, t \in \mathbb{R}_{\geq 0}\}$, and
- *negation*: $\neg D = \{v \mid v \notin D\}$.

Subtraction Given two DBMs D and E, we want to subtract E from D. The resulting set S is defined as the set satisfying the constraints of D and $\neg E$. The set is not necessarily a zone. The result $S = D \wedge \neg E$, denoted $D - E$, is written as:

$$S = D \wedge \neg(\bigwedge_{1 \leq i,j \leq n} e_{ij}) = \bigvee_{1 \leq i,j \leq n} (D \wedge \neg e_{ij}), \quad \text{(De Morgan law)},$$

which is a union of D constrained by each of the negated constraints of E. This gives us a straight-forward *basic algorithm* to compute subtractions. Figure 3 illustrates this basic algorithm with two clocks. The result $S = D - E$ is represented by the union of the six smaller zones on the right in the figure. The number of zones in S is bounded by n^2, and creating each of these zones is $O(n^2)$, so the complexity of the operation is $O(n^4)$ (in both time and space), with n being the number of clocks (we assume this every time we discuss complexity).
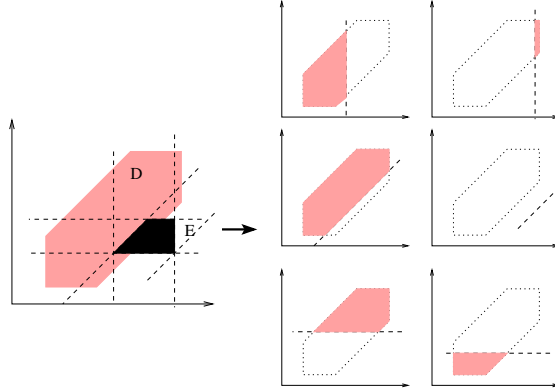
Fig. 3. Basic subtraction algorithm. The result of $D - E$ is the union of all the six zones on the right.

3 Timed Automata with Priorities

We denote by *Act* a set of actions, including the *internal action* τ and *synchronising* actions a. A synchronising action a has a complement $\bar{a} \in Act$ such that $\bar{\bar{a}} = a$. A *timed automaton* $A^i = \langle N^i, l_0^i, E^i, I^i \rangle$ is a finite state automaton with a set N^i of locations l^i, a set E^i of edges, and an initial location l_0^i. The function $I^i : N^i \mapsto \mathcal{B}(C)$ maps to each location an invariant condition. An edge of automata A^i from location l^i to $l^{i'}$ is denoted $l^i \xrightarrow{g,a,r} l^{i'}$. The edges are labelled with clock guards $g \in \mathcal{B}(C)$, actions $a \in Act$, and a set of clocks $r \subseteq C$.

We define a *network of timed automata* as the parallel composition of timed automata $A^1|\cdots|A^n$ communicating on a set of actions *Act*, and extend these with priority orders on actions or automata. A priority order on actions is a partial order $\prec_a$. We write $a \prec_a a'$ to denote that a' has higher priority than a. Similarly for automata we write $A^i \prec_A A^j$ to denote that an automaton A^j has higher priority than A^i.

3.1 Semantics

A *state* of a network of timed automata is a pair $\langle l, v \rangle$, where l is a vector of locations l^i for each automaton, and v is a clock valuation. The initial state $\langle l_0, v_0 \rangle$ puts all automata in their initial locations l_0^i, and maps all clocks to zero.

The invariant $I(l)$ is defined as the conjunction of terms $I^i(l^i)$ for each automaton A^i. An update of the location for automata A^i is denoted by $l[l^{i'}/l^i]$ as the location vector originating from l where l^i is replaced by $l^{i'}$. For a transition t we denote by g^t the conjunction of the guards on the edges participating in that transition. Similarly we denote by r^t the union of clock sets r on the edges, and by l^t the location vector of the state generated by t.

Using a priority order $\prec$ on transitions, a transition can *block* another if it has a higher priority. From a state $\langle l, v \rangle$ a transition t is blocked according to the predicate $block(t) = \exists t'.t \prec t' \wedge v \models g^{t'} \wedge [r^{t'} \mapsto 0]v \models I(l^{t'})$.

The transitions between states can be either delay transitions, internal transitions, or synchronising transitions. The following rules define all possible transitions t:

- *Delay transition:* $\langle l, v \rangle \xrightarrow{d} \langle l, v \oplus d \rangle$ if $v \oplus d' \models I(l)$ for $0 \le d' \le d$.
- *Internal transition:* $\langle l, v \rangle \xrightarrow{a} \langle l[l^{i'}/l^i], v' \rangle$ if there is an edge $l^i \xrightarrow{g,a,r} l^{i'}$ with a local action a such that $v' = [r \mapsto 0]v$, $v \models g$, $v' \models I^i(l^{i'})$, and $\neg block(t)$.
- *Synchronising transition:* $\langle l, v \rangle \xrightarrow{a} \langle l[l^{i'}/l^i, l^{j'}/l^j], v' \rangle$ if there are two edges $l^i \xrightarrow{g^i, a, r^i} l^{i'}$ and $l^j \xrightarrow{g^j, \bar{a}, r^j} l^{j'}$ such that $i \ne j$, $v' = [r^i \mapsto 0, r^j \mapsto 0]v$, $v \models g^i$, $v \models g^j$, $v' \models I^i(l^{i'})$, $v' \models I^j(l^{j'})$, and $\neg block(t)$.

With these semantics, a delay transitions can never be blocked, and no transition can be blocked by a delay transition. In Section 3.3 we show how a priority order $\prec$ over internal and synchronising transitions can be derived from the orders $\prec_a$ or $\prec_A$.

3.2 Symbolic Semantics

We use zones to define a symbolic, finite semantics for networks of timed automata with priorities. A *symbolic state* is a pair $\langle l, D \rangle$ with a location vector l and a zone D. A *symbolic transition* is denoted $\langle l, D \rangle \Longrightarrow \langle l', \hat{D}' \rangle$, where $\hat{D}'$ is a disjunction of a set of zones and $\langle l', \hat{D}' \rangle$ are all symbolic states $\langle l', D' \rangle$ such that $D' \in \hat{D}'$. The set of zones that block a transition t are describe by the predicate $Block(t) = \bigvee_{t \prec t'} free(I(l^{t'}), r^{t'}) \wedge g^{t'}$. The rules for symbolic transitions t are:

- *Symbolic delay transition:* $\langle l, D \rangle \xRightarrow{\delta} \langle l, D^{\uparrow} \wedge I(l) \rangle$.
- *Symbolic internal transition:* $\langle l, D \rangle \xRightarrow{a} \langle l[l^{i'}/l^i], \hat{D}' \rangle$ if there is an edge $l^i \xrightarrow{g,a,r} l^{i'}$ with a local action a, and $\hat{D}' = r(D \wedge g - Block(t)) \wedge I^i(l^{i'})$.
- *Symbolic synchronising transition:* $\langle l, D \rangle \xRightarrow{a} \langle l[l^{i'}/l^i, l^{j'}/l^j], \hat{D}' \rangle$ if there are two edges $l^i \xrightarrow{g^i, a, r^i} l^{i'}$ and $l^j \xrightarrow{g^j, \bar{a}, r^j} l^{j'}$ such that $i \ne j$, and: $\hat{D}' = (r^i \cup r^j)(D \wedge g^i \wedge g^j - Block(t)) \wedge I^i(l^{i'}) \wedge I^j(l^{j'})$.

Theorem 1 (Correctness of Symbolic Semantics) Assume location vectors l_0, l_f, clock assignments u_0, u_f, and a set of zones $\hat{D}_f$. Let $\{u_0\}$ denote the clock constraint with a single solution u_0.

- (*Soundness*) whenever $\langle l_0, \{u_0\} \rangle \Longrightarrow^* \langle l_f, \hat{D}_f \rangle$ then $\langle l_0, u_0 \rangle \longrightarrow^* \langle l_f, u_f \rangle$ for all $u_f \in \hat{D}^f$.
- (*Completeness*) whenever $\langle l_0, u_0 \rangle \longrightarrow^* \langle l_f, u_f \rangle$ then $\langle l_0, \{u_0\} \rangle \Longrightarrow^* \langle l_f, \hat{D}_f \rangle$ for some $\hat{D}_f$ such that $u_f \in \hat{D}_f$.

Proof: By induction on the length of transition sequences. Using the zone operations of Definition 2 it can be shown that $block(t)$ and $Block(t)$ characterizes the same sets of clock valuations. $\square$

3.3 Priorities in UPPAAL

The priority order $\prec$ over transitions can be derived from the priority orders $\prec_a$ on actions and $\prec_A$ on automata. We describe here the order used in the UPPAAL tool [18]. For transitions t and t' with actions a and a' we derive a priority order from $\prec_a$ by defining $t \prec t'$ as $a \prec_a a'$.

Deriving a priority order on transitions from $\prec_A$ is less straightforward, as two automata with different priorities may be involved in a synchronising transition. For two transitions t and t', where t is a synchronisation between A^i and A^j such that $\neg(A^j \prec_A A^i)$, and t' is a synchronisation between $A^{i'}$ and $A^{j'}$ such that $\neg(A^{j'} \prec_A A^{i'})$, we define $t \prec t'$ to hold when:

$$(A^j \prec_A A^{j'}) \vee ((A^i \prec_A A^{i'}) \wedge \neg(A^j \prec_A A^{j'}) \wedge \neg(A^j \succ_A A^{j'}))$$

Intuitively the (weakly) higher priority processes A^j and $A^{j'}$ are compared first. If they are related they define the relation between t and t', otherwise the relation is defined by the relation between A^i and $A^{i'}$.

In a model with priorities on both actions and automata, priorities are resolved by comparing priorities on actions first. Only if they are the same, the priority order on automata is used.

4 DBM Subtraction

4.1 Improved Subtraction

The first observation from the basic algorithm given in Section 2.2 is that some splits may be avoided by taking into account only the constraints that are not redundant in the DBM. It is possible to compute the set of minimal constraints of a DBM [19] in $O(n^3)$ (the set being unique w.r.t. a given clock ordering). As this minimal set is semantically equivalent to the original set E, we use this set E_m instead: $D - E = D - E_m$. In the experiments this algorithm is the base for comparing with our other improvements since it obviously reduces splitting. Figure 4 shows the reduced subtraction by using the minimal set of constraints. We show it is worth spending this extra time because it is more important to reduce the number of DBMs in the result. The global complexity is still $O(n^4)$.

4.2 Disjoint Subtraction

The improved algorithm gives $D - E$ as a union of DBMs that overlap each other, which means there are redundant points. These points will duplicate later operations needlessly, so a second improvement is to ensure that the result is a union of disjoint DBMs. The downside of it is that inclusion checking may become worse for later generated DBMs. The problem exists even without subtraction and it is not obvious to conclude if we are improving or not on this point. The ordering of the splits affects the result but it is still guaranteed to be disjoint. The complexity is still $O(n^4)$.
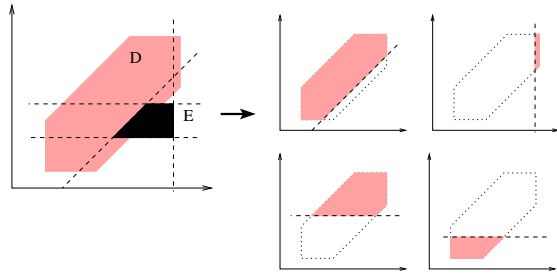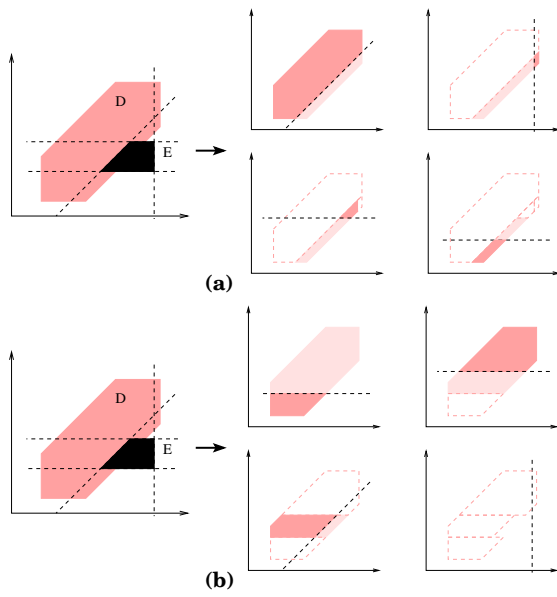
Fig. 4. Subtraction using the minimal set of constraints.

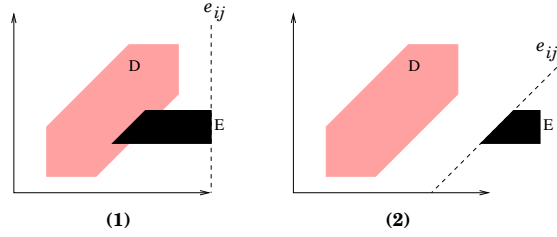**Fig. 5.** Subtraction with disjoint result with two orderings (a) and (b) for splitting.

Fig. 6. Particular cases to consider to simplify subtractions: (1) ignore e_{ij} and (2) $D - E = D$.

Disjoint Subtraction Algorithm. We compute the subtraction $D - E$ with the minimal set of constraints $E_m \subseteq E$ as follows:

1. Compute E_m.
2. $S = false$, $R = D$
3. $\forall e_{ij} \in E_m$, $i \neq j$:
4. $S = S \vee (R \wedge \neg e_{ij})$
5. $R = R \wedge e_{ij}$.
6. Return S.

R is the remainder of the subtraction and serves to compute consecutive splits. The ordering of the splits has an impact on the resulting number of DBMs as shown in Figure 5. The resulting DBMs are disjoint and the result is correct in both cases but in (a) we have 4 splits and a remainder that we will discard and in (b) we have 3 splits and no remainder (the last case discards the remainder trivially and is not a real split).

Lemma 1 (Soundness and completeness of disjoint subtraction) The algorithm still computes the same subtraction $D - E$ and S is a union of disjoint DBMs: $\forall s_1, s_2 \in S. \ s_1 \neq s_2 \Rightarrow s_1 \cap s_2 = \emptyset$.

4.3 Simple Improvements

There are two obvious cases illustrated in Figure 6 that we can detect before starting to compute $D - E$:

1. The negated constraint $\neg e_{ij}$ reduces D to an empty zone, which corresponds to a disjunct with false: we ignore *the constraint e_{ij}*.
2. The negated constraint $\neg e_{ij}$ has no effect on D, which means that $E \wedge D = false$ because DBMs are convex, therefor we stop and the result is D. We ignore the *whole subtraction*.

5 Reducing DBM Subtractions

Reducing subtractions means to reduce the number of splits of the operation but it is not obvious to define what an optimal split is. For a subtraction $D - E$, there may be different combinations that give the same minimal number of splits but the resulting DBMs will be used in further computations and the different combinations will give varying future splits. The problem of computing the minimal split is interesting but it is not obvious if it is possible to do it without worsening the original complexity $O(n^4)$ of the subtraction. In this section we propose a heuristic that tackles both problems: It tries to choose a good ordering to reduce the number of splits overall.

5.1 Efficient Heuristic

The idea is to use a good ordering of the constraints of E to compute the splits such that the first splits will cut the original DBM D into as large as possible DBMs to cancel the upcoming splits as soon as possible (when there is nothing left to do). We use the values $|e_{ij}| - |d_{ij}|$ to order the constraints e_{ij}, taking the smallest values first. This measures on one dimension how much the corresponding facet of E is "inside" D. The *important* trick of the algorithm is to always take the constraint with the smallest value after *every* split because the DBM changes after every split. Complexity-wise, this is equivalent to sorting in $O(n^4)$ (n^2 constraints) instead of $O(n^2 log n^2)$ but it gives better results and it makes more sense since at every iteration the previous values lose their meaning.

Algorithm. The algorithm splits D by choosing the current best e_{ij} as having the smallest $H_{E,R}(i,j) = |e_{ij}| - |r_{ij}|$.

1. If $\exists i, j. \ i \neq j, \ d_{ij} \leq \neg e_{ji}$ then return D.
2. Compute the minimal set of constraints E_m.
3. Initialise $R = D$ and $S = false$.
4. While $R \neq false$ do
5.   Choose $e_{ij} \in E_m$, $i \neq j$ with $min(H_{E,R}(i,j))$.
6.   if $r_{ij} \leq \neg e_{ji}$ return $S \vee R$
7.   else if $e_{ij} \geq r_{ij}$ skip
8.   else $S = S \vee (R \wedge \neg e_{ij})$
9. $R = R \wedge e_{ij}$.
10. Return S.

Step 1 corresponds to case 2 in the preliminaries. It may seem redundant with step 6 but it is to avoid computing the minimal set of constraints in step 2.

Lemma 2 (Soundness of heuristic subtraction) The algorithm computes the subtraction $D - E$ correctly.

Proof: The algorithm is equivalent to the disjoint subtraction in Section 4.2 except for the ordering of the constraints and two improvements to detect the trivial cases mentioned in Section 4.3. □

5.2 Expensive Heuristic

The idea is to ignore (in addition to the previous heuristic) *facets* of E that do not intersect D. A facet of E corresponding to a constraint e_{ij} (of the form $x_i - x_j \sim b_{ij}$) is the hyper-plane $x_i - x_j = b_{ij}$ bounded by the other constraints of E. The intuition is to use the convexity of our DBMs and the fact $D - E = D - (D \cap E)$: If $D \cap E \neq \emptyset$ and a facet of E is not in $D \cap E$, i.e., it does not intersect D, then we can ignore it.

In practice, there are different cases to consider: If the constraints are strict or not and different configurations of the intersections on the corner. In addition, the exact detection of the intersection is $O(n^3)$ and it is not obvious for us if this idea is compatible with the minimal set of constraints, which is, the simple idea poses problems in practice.

To simplify, we define a new heuristic function $H_{E,R}$ that returns ∞ if $\bar{E} \wedge (x_i - x_j = b_{ij}) \cap D = \emptyset$ or the previous value $|e_{ij} - r_{ij}|$ otherwise. The condition means that we make the constraints of E non strict ($\bar{E}$) and we constrain it to be a facet that we use for testing intersection. The intersection detection is partial and is based on case 2 of Section 4.3. The new heuristic function is:

function $H'_{E,R}(i,j)$:
1. $\forall k.\ k \neq i, k \neq j$:
2.   if $|e_{ij} - e_{kj} - r_{ik}| \geq 0$ or $|e_{ij} - e_{ik} - r_{kj}| \geq 0$
3.   then return $+\infty$
4.   return $|e_{ij}| - |r_{ij}|$
end

We use two tricks: First we tighten e_{ji} with $\neg e_{ij}$ to compute the facet as a DBM. A specialisation of Floyd's shortest path [15] can do this in $O(n^2)$ [22]. The second trick is that we do not need all constraints but only the e_{ki} and e_{jk}, which is what the expression in the condition is doing. If the function returns ∞ then there is not intersection with the facet corresponding to the constraint e_{ij}, otherwise we do not know and we use the former heuristic value.

The function has complexity $O(n)$ which is worse than before, but it may reduce splitting further, which we investigate. The global complexity is unfortunately $O(n^5)$. We can get the detection out of the loop and get back to $O(n^4)$ but this is more complex than it seems in practice and the point is to see if it is worth the effort.

6 Experiments

We experiment[1] first with the impact of priorities in our model-checker. In practice, moderate splitting occurs so we focus on subtractions separately to answer the question of what happens on applications that cause much more splitting.

---

[1] All experiments are carried on a dual-Xeon 2.8GHz with 4GB of RAM running Linux 2.6.9.
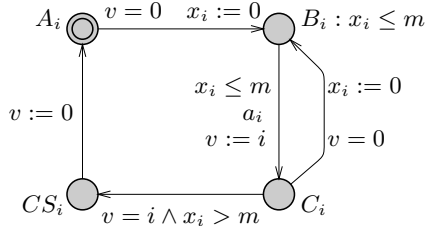
Fig. 7. A process P_i in Fischer's mutual exclusion protocol.

	Original		Priority		Encoded	
N	(s)	(Mb)	(s)	(Mb)	(s)	(Mb)
5	0.85	6.9	0.21	6.6	0.21	6.6
6	17.90	10.2	3.02	6.8	3.46	7.0
7	780.60	40.6	131.57	9.1	144.90	9.8

Table 1. Measurements when model-checking N processes of the Fischer protocol.

6.1 Experimenting with Priorities

We describe an experiment where we compare and evaluate models using priorities and models where priorities are manually encoded using guards and possibly extra edges. For every edge of the original automaton, the encoding is done by restricting the existing guards by removing all parts overlapping with higher priority transitions.

Experiment 1 We introduce priorities on actions in a model of the Fischer protocol [17] for mutual exclusion (Figure 7). The action a_i of the model is used to introduce priorities so that $a_i \prec a_j$ when $i < j$, and $a_i \prec \tau$ for all processes $P_1 \cdots P_N$. Since we give τ-actions the highest priority all N processes will enter location B_i, in contrast to the original model where *at most N* are in location B_i simultaneously. Also, the same process P_1 will always enter the critical section because it will be the last process to reach location C_i. The *Encoded* model is created by adding guards to edges from B_i to C_i that evaluate to false when a higher priority transition τ or $a_j \succ a_i$ is enabled.

Table 1 shows measurements of model-checking various models of the Fischer protocol, to verify that there are no deadlocks and that mutual exclusion holds. The column N is the number of processes, *Original* are the time and memory requirements for model-checking the original model without priorities, *Priority* are the corresponding numbers for the model with priorities, and *Encoded* are the numbers for a model encoding the same behaviour as the model with priorities. The results for the *Priority* and *Encoded* models are comparable, and the overhead of the priority extension is small at worst. This is encouraging since tool support for priorities makes modelling easier.

Algorithms	tgame12	tgame12 reduce	jobshop	jobshop strategy	jobshop reduce	jobshop reduce strategy
basic	343314	283334	55514	91849	45970	73018
	124s	10.9s	9.9s	10.8s	3.0s	3.6s
	319M	39.6M	35.2M	38.8M	18.7M	22.6M
reorder	351615	291627	54996	89763	47073	73730
	165s	10.9s	12.3s	13.4s	3.0s	3.5s
	320M	39.7M	31.9M	35.6M	18.7M	22.6M
disjoint	396053	356322	39776	64195	31592	51531
	120s	12.0s	11.7s	12.8s	3.2s	3.7s
	320M	40.6M	37.1M	40.8M	18.6M	22.6M
efficient	323097	275991	24105	47705	20360	37854
	121s	**10.7s**	**5.3s**	**6.8s**	2.2s	**2.5s**
	319M	39.7M	45.1M	48.6M	18.5M	22.0M
expensive	**320668**	**272788**	**23905**	**45428**	**20248**	**37653**
	121s	11.0s	7.5s	8.9s	**2.1s**	2.6s
	319M	39.7M	45.4M	48.8M	18.5M	22.0M

Table 2. Results of the timed game experiments (tgame) with 12 plates with and without *expensiveReduce* (reduce), and the jobshop experiments (jobshop) with or without strategy, with or without *expensiveReduce*.

6.2 Experimenting with DBM Subtractions

Experiment 2 We have implemented a timed game reachability engine whose purpose is to find winning strategies [10] and we made a variant of it to solve jobhsop scheduling problems states as games. The timed game test example ("tgame") is the production cell [20, 21] with 12 plates. The jobshop example ("jobshop") is modelled from [1] where we find a schedule for 4 jobs using 6 resources. We run variants of the prototype where we store (+strategy) and we do not store the strategy. In addition, for both experiments the main loop can reduce (+reduce) or not federations based on an inclusion checking using subtractions. Table 2 shows the results of these experiments. We give the total number of split operations (first number) as in the previous experiments with time (in seconds) and memory consumption (in megabytes).

Comparing "basic" and "reorder" shows that it is not easy to find a good ordering. Results from "disjoint" show as we claimed that reducing the size of the symbolic states may actually interfere with inclusion checking. The "expensive" heuristic gives a marginal gain considering its cost. The "efficient" heuristic is the best choice. The reduction of federations gives significant gains both in time and memory, which means it is possible to contain the splits to some extent. Still this reduction operation needs a good subtraction. Indeed, experiment 2 shows how bad it may go. Concerning memory consumption it is difficult to draw conclusions. The behaviour of the "efficient" implementation may seem an anomaly but it is explained by the fact that the prototype is using sharing of

DBMs between states. The effect here is that DBMs are less shared since we know we have fewer of them.

These experiments confirm that our heuristic has an overhead but it is compensated by the reduction in splits, in particular for our "efficient" heuristic. We speculate that our priority implementation will behave reasonably well with models that generate more splitting thanks to our subtraction algorithm. Furthermore, we have implemented different reduction algorithms to merge DBMs based on subtraction. Our implementation scales well with good merging algorithms thanks to efficient subtractions. However, this is out of scope of this paper.

7 Conclusion

We have shown that our priority extension is useful for modelling and can also be used to reduce the search state-space. Furthermore, its overhead in our model-checker is reasonable. We have also shown that it is worth the extra effort for a DBM subtraction algorithm to produce fewer zones and to avoid redundancy by making the zones disjoint. The priority extension opens the door to more compact models and the support for subtraction allows us to add support for wanted features in UPPAAL such as urgent transitions with clock guards. In addition, we are improving on reduction techniques to make our model-checker more robust against splitting of DBMs.

References

1. Yasmina Abdeddaïm, Eugene Asarin, and Oded Maler. Scheduling with timed automata. *Theoretical Computer Science*, 354(2):272–300, march 2006.
2. Rajeev Alur and David L. Dill. Automata for modeling real-time systems. In *Proc. of Int. Colloquium on Algorithms, Languages, and Programming*, volume 443 of *LNCS*, pages 322–335, 1990.
3. J.G.P Barnes. *Programming in Ada, Plus and Overview of Ada 9X*. Addison–Wesley, 1994.
4. Johan Bengtsson, W. O. David Griffioen, Kre J. Kristoffersen, Kim G. Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. Automated analysis of an audio control protocol using UPPAAL. *Journal of Logic and Algebraic Programming*, 52–53:163–181, July-August 2002.
5. Johan Bengtsson and Wang Yi. Timed automata: Semantics, algorithms and tools. In W. Reisig and G. Rozenberg, editors, *Lecture Notes on Concurrency and Petri Nets*, volume 3098 of *Lecture Notes in Computer Science*. Springer-Verlag, 2004.
6. S. Bornot, G. Goessler, and J. Sifakis. On the construction of live timed systems. In S. Graf and M. Schwartzbach, editors, *Proc. of the 6th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 1785 of *LNCS*, pages 109–126. SPRINGER, 2000.
7. Marius Bozga, Conrado Daws, Oded Maler, Alfredo Olivero, Stavros Tripakis, and Sergio Yovine. Kronos: A Model-Checking Tool for Real-Time Systems. In *Proc. of the 10th Int. Conf. on Computer Aided Verification*, number 1427 in Lecture Notes in Computer Science, pages 546–550. Springer–Verlag, 1998.

8. Patrice Brémond-Grégoire and Insup Lee. A process algebra of communicating shared resources with dense time and priorities. *Theoretical Computer Science*, 189(1–2):179–219, 1997.
9. G. C. Buttazzo. *Hard Real-Time Computing Systems. Predictable Scheduling Algorithms and Applications*. Kulwer Academic Publishers, 1997.
10. Franck Cassez, Alexandre David, Emmanuel Fleury, Kim G. Larsen, and Didier Lime. Efficient on-the-fly algorithms for the analysis of timed games. In *To appear in CONCUR'05*, LNCS, 2005.
11. Rance Cleaveland and Matthew Hennessy. Priorities in process algebras. *Inf. Comput.*, 87(1-2):58–77, 1990.
12. J. Corbett. Modeling and analysis of real-time ada tasking programs. In *Proceedings of 15th IEEE Real-Time Systems Symposium, San Juan, P uerto Rico, USA*, pages 132–141. IEEE Computer Society Press, 1994.
13. David L. Dill. Timing assumptions and verification of finite-state concurrent systems. volume 407 of *LNCS*, pages 197–212. Springer, 1989.
14. Elena Fersman, Paul Pettersson, and Wang Yi. Timed automata with asynchronous processes: Schedulability and decidability. In J.-P. Katoen and P. Stevens, editors, *Proc. of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, number 2280 in Lecture Notes in Computer Science, pages 67–82. Springer–Verlag, 2002.
15. Robert W. Floyd. Acm algorithm 97: Shortest path. *Communications of the ACM*, 5(6):345, 1962.
16. Pao-Ann Hsiung and Shang-Wei Lin. Model checking timed systems with priorities. In *RTCSA*, pages 539–544, 2005.
17. Leslie Lamport. A fast mutual exclusion algorithm. *ACM Transactions on Computer Systems*, 5(1):1–11, 1987.
18. Kim G. Larsen, Paul Pettersson, and Wang Yi. Uppaal in a Nutshell. *Int. Journal on Software Tools for Technology Transfer*, 1(1–2):134–152, October 1997.
19. Fredrik Larsson, Kim G. Larsen, Paul Pettersson, and Wang Yi. Efficient verification of real-time systems: Compact data structures and state-space reduction. In *Proc. of the 18th IEEE Real-Time Systems Symposium*, pages 14–24. IEEE Computer Society Press, December 1997.
20. Claus Lewerentz and Thomas Lindner. "production cell": A comparative study in formal specification and verification. In *KORSO – Methods, Languages, and Tools for the Construction of Correct Software*, volume 1009 of *LNCS*, pages 388–416. Springer-Verlag, 1995.
21. Helmut Melcher and Klaus Winkelmann. Controller synthesis for the "production cell" case study. In *Proceedings of the second workshop on Formal methods in software practice*, pages 24–36. ACM Press, 1998.
22. Tomas Gerhard Rokicki. *Representing and Modeling Digital Circuits*. PhD thesis, Stanford University, 1993.
23. Stavros Tripakis and Sergio Yovine. Verification of the Fast Reservation Protocol with Delayed Transmission using the tool Kronos. In *Proc. of the 4th IEEE Real-Time Technology and Applications Symposium*. IEEE Computer Society Press, June 1998.
24. Farn Wang. RED: Model-checker for timed automata with clock-restriction diagram. In Paul Pettersson and Sergio Yovine, editors, *Workshop on Real-Time Tools, Aalborg University Denmark*, number 2001-014 in Technical Report. Uppsala University, 2001.