Efficient Timed Reachability Analysis Using Clock Difference Diagrams

G. Behrmann[1], K.G. Larsen[1], J. Pearson[2], C. Weise[1], and W. Yi[2]

[1] BRICS[***], Aalborg University, Denmark, {behrmann|kgl|cweise}@cs.auc.dk
[2] Dept. of Computer Systems, Uppsala University, Sweden, {justin|yi}@docs.uu.se

Abstract. One of the major problems in applying automatic verification tools to industrial-size systems is the excessive amount of memory required during the state-space exploration of a model. In the setting of real-time, this problem of state-explosion requires extra attention as information must be kept not only on the discrete control structure but also on the values of continuous clock variables.
In this paper, we exploit Clock Difference Diagrams, CDD's, a BDD-like data-structure for representing and effectively manipulating certain non-convex subsets of the Euclidean space, notably those encountered during verification of timed automata.
A version of the real-time verification tool UPPAAL using CDD's as a compact data-structure for storing explored symbolic states has been implemented. Our experimental results demonstrate significant space-savings: for eight industrial examples, the savings are in average 42% with moderate increase in runtime.
We further report on how the symbolic state-space exploration itself may be carried out using CDD's.

1 Motivation

In the last few years a number of verification tools have been developed for real-time systems (e.g. [HHW95,DY95,BLLPW96]). The verification engines of most tools in this category are based on reachability analysis of timed automata following the pioneering work of Alur and Dill [AD94]. A timed automaton is an extension of a finite automaton with a finite set of real-valued clock-variables. Whereas the initial decidability results are based on a partitioning of the infinite state-space of a timed automaton into finitely many equivalence classes (so-called *regions*), tools such as KRONOS and UPPAAL are based on more efficient data structures and algorithms for representing and manipulating timing constraints over clock variables. The abstract reachability algorithm applied in these tools is shown in Figure 1. The algorithm checks whether a timed automaton may reach a state satisfying a given state formula ϕ. It explores the state space of the automaton in terms of *symbolic states* of the form (l, D), where l is a

[***] BRICS: Basic Research in Computer Science, Centre of the Danish National Research Foundation

```
PASSED:= {}
WAIT:= {$(l_0, D_0)$}
repeat
        begin
        get $(l, D)$ from WAIT
        if $(l, D) \models \phi$ then return "YES"
        else if $D \not\subseteq D'$ for all $(l, D') \in$ PASSED then
                begin
                add $(l, D)$ to PASSED          (*)
                NEXT:=$\{(l_s, D_s) : (l, D) \leadsto (l_s, D_s) \wedge D_s \neq \emptyset\}$
                for all $(l_{s'}, D_{s'})$ in NEXT do
                        put $(l_{s'}, D_{s'})$ to WAIT
                end
        end
until WAIT={}
return "NO"
```

Fig. 1. An algorithm for symbolic reachability analysis.

control–node and D is a constraint system over clock variables $\{X_1, \dots, X_n\}$. More precisely, D consists of a conjunction of simple clock constraints of the form $X_i \, op \, c$, $-X_i \, op \, c$ and $X_i - X_j \, op \, c$, where c is an integer constant and $op \in \{<, \leq\}$. The subsets of $\mathbb{R}^n$ which may be described by clock constraint systems are called *zones*. Zones are among those convex polyhedra, where all edge-points are integer valued, and where border lines may or may not belong to the set (depending on a constraint being strict or not).

We observe that several operations of the algorithm are critical for efficient implementation. In particular the algorithm depends heavily on operations for checking set inclusion and emptiness. In the computation of the set NEXT, operations for intersection, forward time projection (future) and projection in one dimension (clock reset) are required. A well-known data-structure for representing clock constraint systems is that of *Difference Bounded Matrices*, DBM, [Dill87], giving for each pair of clocks[1] the upper bound on their difference. All operations required in the reachability analysis in Figure 1 can be easily implemented on DBM's with satisfactory efficiency. In particular, the various operations may benefit from a *canonical* DBM representation with tightest bounds on all clock differences computed by solving a shortest path problem. However, computation of this canonical form should be postponed as much as possible, as it is the most costly operation on DBM's with time-complexity $O(n^3)$ (n being the number of clocks).

DBM's obviously consume space of order $O(n^2)$. Alternatively, one may represent a clock constraint system by choosing a minimal subset from the constraints of the DBM in canonical form. This *minimal form* [LPW95] is preferable

[1] For uniformity, we assume a special clock X_0 which is always zero. Thus $X_i \, op \, c$ and $-X_i \, op \, c$ can be rewritten as the differences $X_i - X_0 \, op \, c$ and $X_0 - X_i \, op \, c$.

when adding a symbolic state to the main global data-structure PASSED, as in practice the space-requirement is only linear in the number of clocks.

Considering once again the reachability algorithm in Figure 1, we see that a symbolic state (l, D) from the waiting-list WAIT is freed from being explored (the inner box) provided some symbolic state (l, D') already in PASSED 'covers' it (i.e. $D \subseteq D'$). Though clearly a sound rule and provably sufficient for termination of the algorithm, exploration of (l, D) may be avoided under less strict conditions. In particular, it suffices for (l, D) to be 'covered' collectively by the symbolic states in PASSED with location l, i.e.:

$$D \subseteq \bigcup \{D' \mid (l, D') \in \textsc{Passed}\} \qquad (1)$$

However, this requires handling of unions of zones, which complicates things considerably. Using DBM's, finite unions of zones – which we will call *federations* in the following – may be represented by a list of all the DBM's of the union. However, the more "non-convex" the zone becomes, the more DBM's will be needed. In particular, this representation makes the inclusion-check of (1) computational expensive.

In this paper, we introduce a more efficient BDD-like data-structure for federations, *Clock Difference Diagrams*, CDD's. A CDD is a directed acyclic graph, where inner nodes are associated with a given pair of clocks and outgoing arcs state bounds on their difference. This data-structure contains DBM's as a special case and offers simple boolean set-operations and easy inclusion- and emptiness-checking. Using CDD's, the PASSED-list may be implemented as a collection of symbolic states of the form (l, F), where F is a CDD representing the union of all zones for which the location l has been explored[2]. Thus, the more liberal termination condition of (1) may be applied, potentially leading to faster termination of the reachability algorithm. As any BDD-like data-structure, CDD's eliminate redundancies via sharing of substructures. Thus, the CDD representation of F is likely to be much smaller than the explicit DBM-list representation. Furthermore, sharing of identical substructures between CDD's from *different* symbolic states may be obtained for free, opening for even more efficient storage-usage.

Having implemented a CDD-package and used it in modifying UPPAAL, we report on some very encouraging experimental results. For eight industrial examples found in the literature, significant space-savings are obtained: the savings are in average 42% with moderate increase in run-time (in average an increase of 7%).

To make the reachability algorithm of Figure 1 fully symbolic, it remains to show how to compute the successor set NEXT based on CDD's. In particular, algorithms are needed for computing forward projection in time and clock-reset for this data-structure. Similar to the canonical form for DBM's these operation are obtained via a *canonical* CDD form, where bounds on all arcs are as tight as possible.

[2] Thus D is simply unioned with F, when a new symbolic state (l, D) is added to the PASSED-list (cf. Fig. 1, line $(*)$).

Related Work. The work in [Bal96] and [WTD95] represent early attempts of applying BDD-technology to the verification of continuous real-time systems. In [Bal96], DBM's themselves are coded as BDD's. However, unions of DBM's are avoided and replaced by convex hulls leading to an approximation algorithm. In [WTD95], BDD's are applied to a symbolic representation of the discrete control part, whereas the continuous part is dealt with using DBM's.

The Numerical Decision Diagrams of [ABKMPR97,BMPY97] offer a canonical representation of unions of zones, essentially via a BDD-encoding of the collection of regions covered by the union. [CC95] offers a similar BDD-encoding in the simple case of one-clock automata. In both cases, the encodings are extremely sensitive to the size of the in-going constants. As we will indicate, NDD's may be seen as degenerate CDD's requiring very fine granularity.

CDD's are in the spirit of Interval Decision Diagrams of [ST98]. In [Strehl'98], IDD's are used for analysis in a discrete, one-clock setting. Whereas IDD's nodes are associated with independent real-valued variables, CDD-nodes – being associated with differences – are highly dependent. Thus, the subset- and emptiness checking algorithms for CDD's are substantially different. Also, the canonical form requires additional attention, as bounds on different arcs along a path may interact.

The CDD datastructure was first introduced in [LPWW98], where a thorough study of various possible normalforms is given. A similar datastructure has recently been introduced in [MLAH99a,MLAH99b].

2 Timed Automata

Timed automata were first introduced in [AD94] and have since then established themselves as a standard model for real–time systems. We assume familiarity with this model and only give a brief review in order to fix the terminology and notation used in this paper.

A timed automaton is a standard finite-state automaton extended with a finite collection of real-valued clocks. The nodes (often called *(control) nodes*) are labelled with an *invariant*. Transitions are labelled with a *guard*, a *clock reset* and a *synchronisation*. Guards and invariants are clock constraints. Intuitively, a timed automaton starts execution with all clocks set to zero. Clocks increase uniformly with time while the automaton is within a node. The automaton can only stay within a node while the clocks fulfill the node's invariant. A transition can be taken if the clocks fulfill the guard. By taking the transition, all clocks in the clock reset will be set to zero, while the remaining keep their values. Thus transitions occur instantaneously. Semantically, a state of an automaton is a pair of a control node and a *clock valuation*, i.e. the current setting of the clocks. Transitions in the semantic interpretation are either labelled with a synchronisation (if it is an instantaneous switch from the current node to another) or with a positive time delay (if the automaton stays within a node letting time pass).

For the formal definition, we denote the clocks by $C = \{X_1, \ldots, X_n\}$, and use $\mathcal{B}(C)$ ranged over by g and D to denote the set of clock constraint systems over C.

Definition 1. *A timed automaton A over clocks C is a tuple $\langle N, l_0, E, I\rangle$ where N is a finite set of nodes (control-nodes), l_0 is the initial node, $E \subseteq N \times \mathcal{B}(C) \times 2^C \times C$ corresponds to the set of edges, and finally, $I : N \to \mathcal{B}(C)$ assigns invariants to nodes. In the case, $\langle l, g, r, l'\rangle \in E$, we write $l \xrightarrow{g,r} l'$.*

Formally, we represent the values of clocks as functions (called clock assignments) from C to the non–negative reals $\mathbb{R}_{\geq}$. We denote by $\mathcal{V}$ the set of clock assignments for C. A semantical *state* of an automaton A is now a pair (l, u), where l is a node of A and u is a clock assignment for C, and the semantics of A is given by a transition system with the following two types of transitions (corresponding to delay–transitions and edge–transitions):

- $(l, u) \longrightarrow (l, u + d)$ if $I(l)(u)$ and $I(l)(u + d)$
- $(l, u) \longrightarrow (l', u')$ if there exist g, r such that $l \xrightarrow{g,r} l'$, $u \in g$, $u' = [r \mapsto 0]u$, $I(l)(u)$ and $I(l')(u')$

where for $d \in \mathbb{R}_{\geq}$, $u + d$ denotes the time assignment which maps each clock X in C to the value $u(X) + d$, and for $r \subseteq C$, $[r \mapsto 0]u$ denotes the assignment for C which maps each clock in r to the value 0 and agrees with u over $C \backslash r$. By $u \in g$ we denote that the clock assignment u satisfies the constraint g (in the obvious manner).

Clearly, the semantics of a timed automaton yields an infinite transition system, and is thus not an appropriate basis for decision algorithms. However, efficient algorithms may be obtained using a finite–state *symbolic* semantics based on *symbolic states* of the form (l, D), where $D \in \mathcal{B}(C)$ [HNSY94,YPD94]. The symbolic counterpart to the standard semantics is given by the following two (fairly obvious) types of symbolic transitions:

$\bullet\ (l, D) \rightsquigarrow (l, (D \wedge I(l))^{\uparrow} \wedge I(l))$ $\bullet\ (l, D) \rightsquigarrow (l', r(g \wedge D \wedge I(l)) \wedge I(l'))$ if $l \xrightarrow{g,r} l'$

where time progress $D^{\uparrow} = \{u + d \,|\, u \in D \wedge d \in \mathbb{R}_{\geq}\}$ and clock reset $r(D) = \{[r \mapsto 0]u \,|\, u \in D\}$. It may be shown that $\mathcal{B}(C)$ (the set of constraint systems) is closed under these two operations ensuring the well–definedness of the semantics. Moreover, the symbolic semantics corresponds closely to the standard semantics in the sense that, whenever $u \in D$ and $(l, D) \rightsquigarrow (l', D')$ then $(l, u) \longrightarrow (l', u')$ for some $u' \in D'$.

3 Clock Difference Diagrams

While in principle DBM's are an efficient implementation for clock constraint systems, especially when using canonical form only when necessary and minimal form when suitable, they are not very good at handling unions of zones. In this section we will introduce a more efficient data structure for federations: *clock difference diagrams* or short CDD's. A CDD is a directed acyclic graph with two kinds of nodes: inner nodes and terminal nodes. Terminal nodes represent the constants true and false, while inner nodes are associated with a *type* (i.e. a clock pair) and arcs labeled with intervals giving bounds on the clock pair's difference. Figure 2 shows examples of CDD's.

A CDD is a compact representation of a decision tree for federations: take a valuation, and follow the unique path along which the constraints given by type and interval are fulfilled by the valuation. If this process ends at a true node, the valuation belongs to the federation represented by this CDD, otherwise not. A CDD itself is not a tree, but a DAG due to sharing of isomorphic subtrees.

A *type* is a pair (i, j) where $1 \leq i < j \leq n$. The set of all types is written $\mathcal{T}$, with typical element t. We assume that $\mathcal{T}$ is equipped with a linear ordering $\sqsubseteq$ and a special bottom element $(0, 0) \in \mathcal{T}$, in the same way as BDD's assume a given ordering on the boolean variables. By $\mathcal{I}$ we denote the set of all non-empty, convex, integer-bounded subsets of the real line. Note that the integer bound may or may not be within the interval. A typical element of $\mathcal{I}$ is denoted I. We write $\mathcal{I}_\emptyset$ for the set $\mathcal{I} \cup \{\emptyset\}$.

In order to relate intervals and types to constraint, we introduce the following notation: (i) given a type (i, j) and an interval I of the reals, by $I(i, j)$ we denote the clock constraint having type (i, j) which restricts the value of $X_i - X_j$ to the interval I, (ii) given a clock constraint D and a valuation v, by $D(v)$ we denote the application of D to v, i.e. the boolean value derived from replacing the clocks in D by the values given in v.

Note that typically we will use the notation jointly, i.e. $I(i, j)(v)$ expresses the fact that v fulfills the constraint given by the interval I and the type (i, j).

This allows us to give the definition of a CDD:

Definition 2 (Clock Difference Diagram). *A Clock Difference Diagram (CDD) is a directed acyclic graph consisting of a set of nodes V and two functions* $\mathsf{type} : V \longrightarrow \mathcal{T}$ *and* $\mathsf{succ} : V \longrightarrow 2^{\mathcal{I} \times V}$ *such that*

- *V has exactly two terminal nodes called* True *and* False*, where* $\mathsf{type}(\mathsf{True}) = \mathsf{type}(\mathsf{False}) = (0, 0)$ *and* $\mathsf{succ}(\mathsf{True}) = \mathsf{succ}(\mathsf{False}) = \emptyset$.
- *all other nodes $n \in V$ are inner nodes, which have attributed a type* $\mathsf{type}(n) \in \mathcal{T}$ *and a finite set of successors* $\mathsf{succ}(n) = \{(I_1, n_1), \ldots, (I_k, n_K)\}$*, where* $(I_i, n_i) \in \mathcal{I} \times V$.

We shall write $n \xrightarrow{I} m$ to indicate that $(I, m) \in \mathsf{succ}(n)$*. For each inner node n, the following must hold:*

- *the successors are disjoint: for* $(I, m), (I', m') \in \mathsf{succ}(n)$ *either* $(I, m) = (I', m')$ *or* $I \cap I' = \emptyset$,
- *the successor set is an $\mathbb{R}$-cover:* $\bigcup\{I \mid \exists m. n \xrightarrow{I} m\} = \mathbb{R}$,
- *the CDD is ordered: for all m, whenever $n \xrightarrow{I} m$ then* $\mathsf{type}(m) \sqsubseteq \mathsf{type}(n)$

Further, the CDD is assumed to be reduced, i.e.

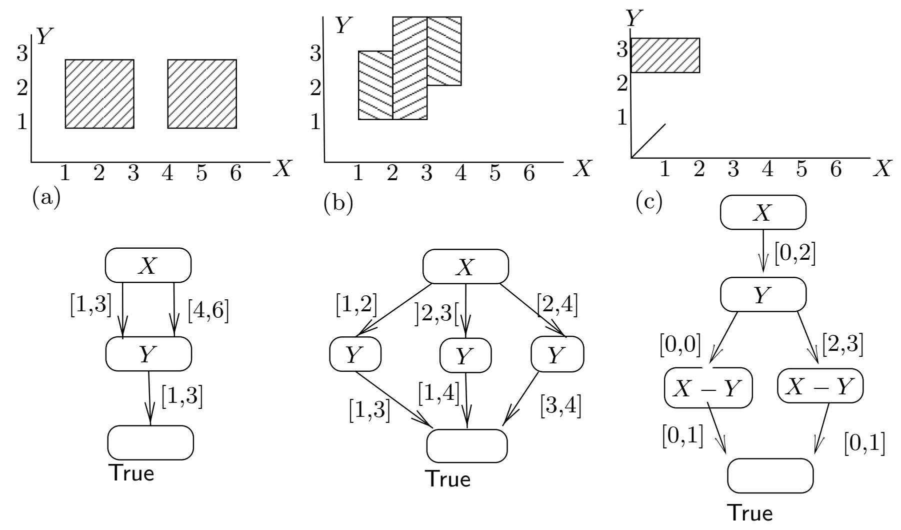

Fig. 2. Three example CDD's. Intervals not shown lead implicitly to False

– *it has maximal sharing: for all* $n, m \in V$, $\mathsf{succ}(n) = \mathsf{succ}(m)$ *implies* $n = m$,
– *it has no trivial edges: whenever* $n \xrightarrow{I} m$ *then* $I \neq \mathbb{R}$,
– *all intervals are maximal: whenever* $n \xrightarrow{I_1} m, n \xrightarrow{I_2} m$ *then* $I_1 = I_2$ *or* $I_1 \cup I_2 \notin \mathcal{I}$

Note that we do not require a special root node. Instead each node can be chosen as the root node, and the sub-DAG underneath this node is interpreted as describing a (possibly non-convex) set of clock valuations. This allows for sharing not only within a representation of one set of valuations, but between all representations. Figure 2 gives some examples of CDD's. The following definition makes precise how to interpret such a DAG:

Definition 3. *Given a CDD* $(V, \mathsf{type}, \mathsf{succ})$, *each node* $n \in V$ *is assigned a semantics* $[\![n]\!] \subseteq \mathcal{V}$, *recursively defined by*

– $[\![\mathsf{False}]\!] := \emptyset$, $[\![\mathsf{True}]\!] := \mathcal{V}$,
– $[\![n]\!] := \{v \in \mathcal{V} \mid n \xrightarrow{I} m, I(\mathsf{type}(n))(v) = \mathsf{true}, v \in [\![m]\!]\}$ *n an inner node*

For BDD's and IDD's, testing for equality can be achieved easily due to their canonicity: the test is reduced to a pure syntactical comparison. However, in the case of CDD's canonicity is not achieved in the same straightforward manner.

To see this, we give an example of two reduced CDD's in Figure 3(a) describing the same set. The two CDD's are however not isomorphic. The problem with CDD's – in contrast to IDD's – is that the different types of constraints in the nodes are not independent, but influence each other. In the above example obviously $1 \leq X \leq 3$ and $X = Y$ already imply $1 \leq Y \leq 3$. The constraint on Y

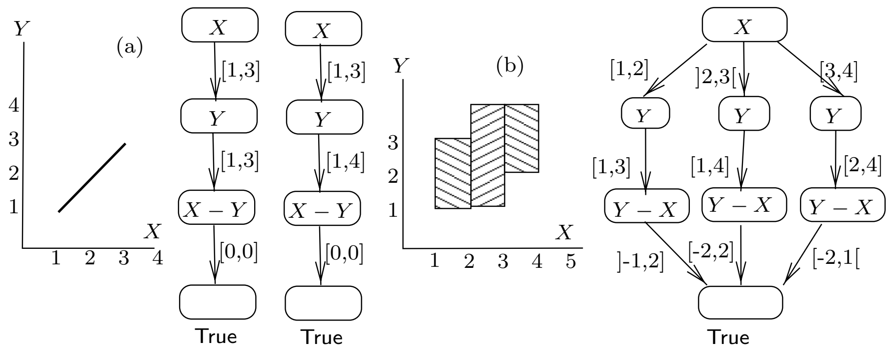

Fig. 3. (a) Two reduced CDD's for the same zone, (b) A tightened CDD

in the CDD on the right hand side is simply too loose. Therefore a step towards an improved normal form is to require that on all paths, the constraints should be the tightest possible. We turn back to this issue in the final section.

4 Operation on CDD's

Simple Operations. Three important operations on CDD's, namely union, intersection and complement, can be defined analogously to IDD's. All use a function makenode which for a given type t and a successor set $S = \{(I_1, n_1), \dots, (I_k, n_K)\}$ will either return the unique node in the given CDD $C = (V, \mathsf{type}, \mathsf{succ})$ having these attributes or, in case no such exists, add a new node to the CDD with the given attributes. This operation – shown in Figure 4 – is important in order to keep reducedness of the CDD. Note that using a hashtable to identify nodes already in V, makenode can be implemented to run in constant time. Note further that makenode itself uses an operation reduce – not given in this paper – which ensures that S itself is reduced, i.e. it has maximal sharing, no trivial edges and all intervals are maximal. Additionally, S is required to be well-formed, i.e. all intervals must be disjoint and form an $\mathbb{R}$-cover.

Then union can be defined as in Figure 4. Intersection is computed by replacing "union" by "intersect" everywhere in the definition of the union operation, and additionally adjusting the base cases. The complement is computed by essentially swapping True and False nodes.[3]

From constraint systems to CDD's. The reachability algorithm of Uppaal currently works with constraint systems (represented either as canonical DBM's or in the minimal form). The desired reachability algorithm will need to combine

[3] As for the BDD apply-operator, using a hashed operation-cache is needed to avoid recomputation of the same operation for the same arguments.

```
makenode($t, S$):  reduce(S)
                   if ($\exists n \in V$.type($n$) $= t \wedge$ succ($n$) $= S$) return $n$
                   else $V := V \cup \{n\}$ // where $n$ is a fresh node
                        type := type $\cup \{n \mapsto t\}$; succ := succ $\cup \{n \mapsto S\}$
                        return $n$
                   endif
union($n_1, n_2$):if $n_1 =$ True or $n_2 =$ True then return True
                elseif $n_1 =$ False then return $n_2$
                elseif $n_2 =$ False then return $n_1$
                else if type($n_1$) = type($n_2$) then
                        return makenode$\big($type($n_1$), $\{(I_1 \cap I_2$,union$(n_1', n_2')) \mid$
                                $n_1 \xrightarrow{I_1} n_1', n_2 \xrightarrow{I_2} n_2', I_1 \cap I_2 \neq \emptyset\}\big)$
                    elseif type($n_1$) $\sqsubseteq$ type($n_2$) then
                        return makenode$\big($type($n_1$), $\{(I_1,$ union$(n_1', n_2)) \mid n_1 \xrightarrow{I_1} n_1'\}\big)$
                    elseif type($n_2$) $\sqsubseteq$ type($n_1$) then
                        return makenode$\big($type($n_2$), $\{(I_2,$ union$(n_1, n_2')) \mid n_2 \xrightarrow{I_2} n_2'\}\big)$
                    endif
                endif
makeCDD($D$):  $n :=$ True
               for $t \in \mathcal{T} \setminus \{(0,0)\}$ do // use ordering $\sqsubseteq$
                   $I := I_{D(t)}$
                   if $I \neq \mathbb{R}$ then
                      if $lo(I) = \emptyset$ then $n :=$ makenode($t, \{(I, n), (hi(I),$ False$)\}$)
                      elseif $hi(I) = \emptyset$ then $n :=$ makenode($t, \{(I, n), (lo(I),$ False$)\}$)
                      else $n :=$ makenode($t, \{(I, n), (hi(I),$ False$), (lo(I),$ False$)\}$)
                      endif
                   endif
               endfor
               return $n$
subset($D, n$):  if $D =$ false or $n =$ True then return true
                 elseif $n =$ False then return false
                 else return $\bigwedge_{n \xrightarrow{I} m}$ subset($D \wedge I$(type($n$)), $m$)
                 endif
```

Fig. 4. Algorithms

and compare DBM's obtained from exploration of the timed automaton with CDD's used as a compact representation of the PASSED-list.

For the following we assume that a constraint system D holds at most one simple constraint for each pair of clocks X_i, X_j (which is obviously true for DBM's and the minimal form). Let $D(i, j)$ be the set of all simple constraints of type (i, j), i.e. those for $X_i - X_j$ and $X_j - X_i$. The constraint system $D(i, j)$ gives an upper and/or a lower bound for $X_i - X_j$. If not present, choose $-\infty$ as lower and $+\infty$ as upper bound. Denote the interval defined thus by $I_{D(i,j)}$.

Further, given an interval $I \in \mathcal{I}$, let $lo(I) := \{r \in \mathbb{R} \mid \forall r' \in I. r < r'\}$ be the set of lower bounds and $hi(I) := \{r \in \mathbb{R} \mid \forall r' \in I. r > r'\}$ the set of upper bounds. Note that always $lo(I), hi(I) \in \mathcal{I}_\emptyset$. Using this notation, a simple algorithm makeCDD for constructing a CDD from a constraint system can be given as in Figure 4. Using this, we can easily union zones to a CDD as required in the modified reachability algorithm of Uppaal (cf. footnote 2). Note that for this asymmetric union it is advisable to use the minimal form representation for the zone, as this will lead to a smaller CDD, and subsequently to a faster and less space-consuming union-operation.

Crucial Operations. Testing for equality and set-inclusion of CDD's is not easy without utilizing a normal form. Looking at the test given in (1) it is however evident that all we need is to test for inclusion between a zone and a CDD. Such an asymmetric test for a zone Z and a CDD n can be implemented as shown in Figure 4 without need for canonicity.

Note that when testing for emptiness of a DBM as in the first if-statement, we need to compute its canonical form. If we know that the DBM is already in canonical form, the algorithm can be improved by passing $D \wedge I(\mathsf{type}(n))$ in canonical form. As $D \wedge I(\mathsf{type}(n))$ adds no more than two constraints to the zone, computation of the canonical form can be done faster than in the general case, which would be necessary in the test $D = \mathsf{true}$.

The above algorithm can also be used to test for emptiness of a CDD using $\mathsf{empty}(n) := \mathsf{subset}(\mathsf{true}, \mathsf{complement}(n))$, where true is the empty set of constraints, fulfilled by every valuation.

As testing for set inclusion $C_1 \subseteq C_2$ of two CDD's C_1, C_2 is equivalent to testing for emptiness of $C_1 \cap \overline{C_2}$, also this check can be done without needing canonicity.

5 Implementation and Experimental Results

This section presents the results of an experiment where both the current[4] and an experimental CDD-based version of Uppaal were used to verify eight industrial examples found in the literature – including a gearbox controller [LPY98], various communication protocols used in Philips audio equipment (see [BPV94], [DKRT97], [BGK+96]), and in B&O audio/video equipment [HSLL97,HLS98], and the start-up algorithm of the DACAPO protocol [LPY97] – as well as Fischer's protocol for mutual exclusion.

In Table 1 we present the space requirements and runtime of the examples on a Sun UltraSPARC 2 equipped with 512 MB of primary memory and two 170 MHz processors. Each example was verified using the current purely DBM-based algorithm of Uppaal (Current), and three different CDD-based algorithms. The first (CDD) uses CDD's to represent the continuous part of the Passed-list, the second (Reduced) is identical to CDD except that all inconsistent paths – i.e.

[4] More precisely Uppaal version 2.19.2, which is the most recent version of Uppaal currently used in-house.

Table 1. Performance statistics for a number of systems. **P** is the number of processes, **V** the number of discrete variables, and **C** the number of clocks in the system. All times are in seconds and space usage in kilobytes. Space usage only includes memory required to store the PASSED-list.

				Current		**CDD**		**Reduced**		**CDD+BDD**	
System	**P**	**V**	**C**	**Time**	**Space**	**Time**	**Space**	**Time**	**Space**	**Time**	**Space**
PHILIPS	4	4	2	0.2	25	0.2	23	0.2	23	0.35	94
PHILIPS COL	7	13	3	21.8	2,889	23.0	1,506	28.8	1,318	70.6	5,809
B&O	9	22	3	56.0	5,793	55.9	2,248	63.4	2,240	300.2	4,221
BRP	6	7	4	22.1	3,509	21.3	465	46.5	448	68.9	873
POWERDOWN1	10	20	2	81.3	4,129	79.2	1,539	82.6	1,467	164.7	4,553
POWERDOWN2	8	20	1	19.3	4,420	19.8	4,207	19.7	4,207	79.5	5,574
DACAPO	6	12	5	55.1	4,474	57.1	2,950	64.5	2,053	256.1	6,845
GEARBOX	5	4	5	10.5	1,849	11.2	888	12.35	862	29.9	7,788
FISCHER4	4	1	4	1.14	129	1.36	96	2.52	48	2.3	107
FISCHER5	5	1	5	40.6	1,976	61.5	3,095	154.4	396	107.3	3,130

those representing the empty set – are removed from the CDD's, and the third (CDD+BDD) extends CDD's with a BDD-based representation of the discrete part in order to achieve a fully symbolic representation of the PASSED-list. As can be seen, our CDD-based modification of UPPAAL leads to truly significant space-savings (in average 42%) with only moderate increase in run-time (in average 7%). When inconsistent paths are eliminated the average space-saving increases to 55% at the cost of an average increase in run-time of 54%. If we only consider the industrial examples the average space-savings of CDD are 49% while the average increase in run-time is below 0.5%. Maybe unexpectedly, CDD+BDD when compared with Current leads to a degraded performance in both time and space. Additionally, a closer look at the usage of the WAIT-list reveals that the less strict termination condition of (1) only in a few cases leads to faster termination. This offers a good explanation for the lack in runtime-improvement.

6 Towards a Fully Symbolic Timed Reachability Analysis

The presented CDD-version of UPPAAL uses CDD's to store the PASSED-list, but zones (i.e. DBM's) in the exploration of the timed automata. The next goal is to use CDD's in the exploration as well, thus treating the continuous part fully symbolic. In combination with the suggested BDD-based approach for the discrete part, this would result in a fully symbolic timed reachability analysis, saving even more space and time.

The central operations when exploring a timed automaton are time progress and clock reset. Using *tightened CDD's*, these operations can be defined along the same lines as for DBM's. A tightened CDD is one where along each path to True all constraints are the the tightest possible. In [LPWW98] we have shown how to effectively transform any given CDD into an equivalent tightened one.

Figure 3(b) shows the tightened CDD-representation for example (b) from Figure 2. Given this tightened version, the time progress operation is obtained by simply removing all upper bounds on the individual clocks. In general, this gives a CDD with overlapping intervals, which however can easily be turned into a CDD obeying our definition. More details on these operations can be found in [LPWW98].

CDD's come equipped with an obvious notion of being *equally fine partitioned.* For equally fine partitioned CDD's we have the following normal form theorem [LPWW98]:

Theorem 1. *Let C_1, C_2 be two CDD's which are tightened and equally fine partitioned. Then $[\![C_1]\!] = [\![C_2]\!]$ iff C_1 and C_2 are graph-isomorphic.*

A drastic way of achieving equally fine partitioned CDD's is to allow only atomic integer-bounded intervals, i.e. intervals of the form $[n, n]$ or $(n, n+1)$. This approach has been taken in [ABKMPR97,BMPY97] demonstrating canonicity. However, this approach is extremely sensitive to the size of the constants in the analyzed model. In contrast, for models with large constants our notion of CDD allows for coarser, and hence more space-efficient, representations.

7 Conclusion

In this paper, we have presented Clock Difference Diagrams, CDD's, a BDD-like data-structure for effective representation and manipulation of finite unions of zones. A version of the real-time verification tool Uppaal using CDD's to store explored symbolic states has been implemented. Our experimental results on eight industrial examples found in the literature demonstrate significant space-savings (in average 42%) with a moderate increase in run-time (in average 7%). Currently, we are pursuing realization of the fully symbolic state-space exploration of the last section and [LPWW98], extending Uppaal from pure reachability checking to checking for general real-time properties.

References

[ABKMPR97] Asarain, Bozga, Kerbrat, Maler, Pnueli, Rasse. Data-Structures for the Verification of Timed Automata. In Proc. HART'97, LNCS 1201, pp. 346–360.

[AD94] Alur, Dill. Automata for Modelling Real-Time Systems. In *Proc. of ICALP'90*, LNCS 443, 1990.

[Bal96] Felice Balarin. *Approximate Reachability Analysis of Timed Automata.* Proc. Real-Time Systems Symposium, Washington, DC, December 1996, pp. 52–61.

[BGK+96] Bengtsson, Griffioen, Kristoffersen, Larsen, Larsson, Pettersson, Yi. Verification of an audio protocol with bus collision using uppaal. CAV'96, LNCS 1102, 1996.

[BLLPW96] Bengtsson, Larsen, Larsson, Pettersson, Yi. Uppaal in 1995. In *Proc. TACAS'96*, LNCS 1055, pp. 431–434. Springer March 1996.

[BMPY97] Bozga, Maler, Pnueli, Yovine. Some progress in the symbolic verification of timed automata. *Proc. CAV'97*, LNCS 1254,pp. 179–190, 1997.

[BPV94] Bosscher, Polak, Vaandrager. Verification of an Audio-control Protocol. In *Proc. FTRTFT*, LNCS 863, 1994.

[CC95] Campos, Clarke. Real-time symbolic model checking for discrete time models. In C. Rattray T. Rus, Eds., *AMAST Series in Computing: Theories and Experiences for Real-Time System Development*, 1995.

[Dill87] Dill. *Timing Assumptions and Verification of Finite-State Concurrent Systems.* in: LNCS 407, Springer Berlin 1989, pp. 197-212.

[DKRT97] D'Arginio, Katoen, Ruys, Tretmans. Bounded retransmission protocol must be on time ! In *Proc. TACAS'97*, LNCS 1217, 1997.

[DY95] Daws, Yovine. Two examples of verification of multirate timed automata with Kronos. In *Proc. 16*th *IEEE Real-Time Systems Symposium*, pp 66–75, Dec 95.

[HNSY94] Henzinger, Nicollin, Sifakis, Yovine. Symbolic Model Checking for Real-Time Systems. *Information and Computation*, 111(2):193–244, 1994.

[HHW95] Henzinger, Ho, Wong-Toi. A Users Guide to HyTech. Technical report, Department of Computer Science, Cornell University, 1995.

[HLS98] Havelund, Larsen, Skou. Formal Verification of an Audio/Video Power Controller using the Real-Time Model Checker UPPAAL. Technical report made for Bang& Olufsen, 1998.

[HSLL97] Havelund, Skou, Larsen, Lund. Formal Modelling and Analysis of an Audio/Video Protocol: An Industrial Case Study using UPPAAL. In *In Proc. 18th IEEE Real-Time System Symposium*, 1997.

[LPW95] Larsen, Pettersson, Yi. *Compositional and Symbolic Model-Checking of Real-Time Systems.* In Proc. 16th IEEE Real-Time Systems Symposium, December 1995.

[LPWW98] Larsen, Weise, Yi, Pearson. *Clock Difference Diagrams.* DoCS Technical Report No.98/99, Uppsala University, Sweden, presented at the Nordic Workshop on Programming Theory, Turku, Finland, November 1998.

[LPY97] Lönn, Pettersson, Yi. Formal Verification of a TDMA Protocol Start-Up Mechanism. In *Proc. 1997 IEEE Pacific Rim International Symposium on Fault-Tolerant Systems*, pp. 235–242, 1997.

[LPY98] Lindahl, Pettersson, Yi. Formal design and analysis of a gear controller. In Bernhard Steffen (Ed.), *Proc. TACAS'98*, LNCS 1384, pp. 281–297. Springer 1998.

[MLAH99a] Møller, Lichtenberg, Andersen, Hulgaard. *Difference decision diagrams.* Technical report IT-TR-1999-023, Technical University of Denmark, February 1999.

[MLAH99b] Møller, Lichtenberg, Andersen, Hulgaard. *On the symbolic verification of timed systems.* Technical report IT-TR-1999-024, Technical University of Denmark, February 1999.

[ST98] Strehl, Thiele. Symbolic Model Checking of Process Networks Using Interval Diagram Techniques. ICCAD-98, San Jose, California, pp. 686–692, 1998.

[Strehl'98] Strehl. Using Interval Diagram Techniques for the Symbolic Verification of Timed Automata. Technical Report TIK-53, ETH Zürich, July 1998.

[WTD95] Wong-Toi, Dill. *Verification of real-time systems by successive over and under approximation.* CAV'95, July 1995.

[YPD94] Yi, Pettersson, Daniels. Automatic Verification of Real-Time Communicating Systems By Constraint-Solving. In *Proc. 7*th *International Conference on Formal Description Techniques*, 1994.