Nordic Journal of Computing

CLOCK DIFFERENCE DIAGRAMS

Kim G. Larsen[1] Justin Pearson[2] Carsten Weise[1]
Wang Yi[2]
[1] *BRICS*,* *Aalborg University, Denmark*
[2] *Department of Computer Systems, Uppsala University, Sweden*

Abstract. In this paper, we present Clock Difference Diagrams (CDD), a new BDD-like data-structure for effective representation and manipulation of certain non-convex subsets of the Euclidean space, notably those encountered in verification of timed automata. It is shown that all set-theoretic operations including inclusion checking may be carried out efficiently on Clock Difference Diagrams. Other clock operations needed for fully symbolic analysis of timed automata e.g. future- and reset-operations, can be obtained based on a tight normalform for CDD. A version of the real-time verification tool UPPAAL using Clock Difference Diagrams as the main data-structure has been implemented. Experimental results demonstrate significant space-savings: for nine industrial examples the savings are in average 42% with moderate increase in runtime.

CR Classification: D.2.1, D.2.2, D.2.4, I.2.2, I.6.4, F.3.1.

Key words: automatic verification, real-time systems, timed automata, symbolic model-checking, clock decision diagrams.

1. Introduction

Model-checking has established itself as a powerful technique for checking whether a given formally described system satisfies a desired property. For parallel systems, model-checking suffers from the inherent problem of state explosion, i.e. the exponential growth in the size of the global state-space in the number of component systems. The *symbolic approach* to model-checking attempts to conquer this problem by using implicit representations of sets of states. In particular, in the case of finite-state systems, Binary Decision Diagrams [Burch *et al.* 1990], BDD's, has proven to be an extremely compact and efficient data-structure in many practical applications.

In the last few years model-checking techniques and tools have been successfully extended to the setting of real-time systems (e.g. [Wang *et al.* 1994],[Henzinger *et al.* 1995],[Daws and Yovine 1995],[Bengtsson *et al.* 1996]).

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

Received December 1998, Revised June 1999

The verification engines of most tools in this category are based on timed automata following the pioneering work of [Alur and Dill 1990]. Whereas the initial decidability results use a symbolic representation in terms of a partitioning of the infinite state-space of a timed automaton into finitely many equivalence classes (so-called *regions*), current tools such as KRONOS and UPPAAL are based on more efficient data structures and algorithms for symbolic representation and manipulation of convex subsets of the Eucledian space using simple constraints, so-called Clock Constraints, over clock variables.

Clock Constraints only offer a symbolic representation of the continuous part of the state-space of timed automata. A fully symbolic representation should ideally integrate a similar symbolic representation of the discrete part. Unfortunately, Clock Constraints are not closed under the union operation[1], which has made this a notoriously difficult task.

In this paper we present Clock Difference Diagrams, CDD's, a new BDD-like data-structure for representing and manipulating certain non-convex subsets of the Eucledian space. In particular CDD's are a generalization of Clock Constraints which are closed under arbitrary finite unions.

In section 2 we give the preliminary definitions for timed automata and Clock Constraints. Section 3 introduces our new data-structure, CDD's, and sections 4 and 5 show how set-theoretic operations as well as other operations required for the fully symbolic analysis of timed automata may be carried out on this representation. Section 6 presents a relative normal form for CDD's, which is relative to a notion of granularity. Section 7 report on encouraging experimental results obtained from a version of the real-time verification tool UPPAAL based on CDD's. Section 8 concludes the paper.

Related Work

The work in [Balarin 1996] and [Wong-Toi and Dill 1995] represent early attempts of applying BDD-technology to the verification of continuous real-time systems. In [Balarin 1996], Clock Constrains themselves are coded as BDD's. However, unions of Clock Constraints are avoided and replaced by convex hulls leading to an approximation algorithm. In [Wong-Toi and Dill 1995], BDD's are applied to a symbolic representation of the discrete control part, whereas the continuous part is dealt with using Clock Constraints.

The Numerical Decision Diagrams (NDD's) of [Asarain *et al.* 1997],[Bozga *et al.* 1997] offer a canonical representation of unions of zones, essentially via a BDD-encoding of the collection of regions covered by the union. The paper [Campos and Clarke 1995] offers a similar BDD-encoding in the simple case of one-clock automata. In both cases, the encodings are extremely sensitive

[1] as the union of two convex sets is not necessarily convex.

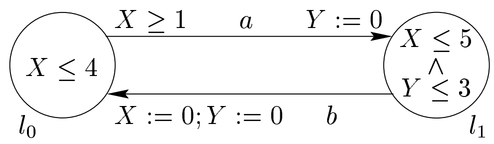

Fig. 1: A Timed Automaton.

to the size of the constants used in clock constraints. 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 (IDD's) of [Strehl and Thiele 1998]. In [Strehl 1998], IDD's are used for analysis in a discrete setting. Whereas IDD's nodes are associated with independent real-valued variables the nodes of CDD's are associated with differences of clock values, which makes the nodes highly interdependent (and of course more expressive). Thus, the subset and emptiness checking algorithms for CDD's are substantially different from IDD's. Also, the canonical form requires additional attention, as bounds on different arcs along a path may interact.

Another approach in [R.L.Spelberg *et al.* 1998] applies partitioned refinement to obtain an efficient real-time model-checking algorithm. The CDD-datastructure was first introduced in [Larsen *et al.* 1998]. A similar datastructure has recently been introduced in [Møller *et al.* 1999],[Møller *et al.* 1999].

2. Preliminaries

The theory of timed automata was first introduced in [Alur and Dill 1990] to provide a formal model for real–time systems based on finite-state automata. In recent years, several verification tools for real–time systems in the framework of timed automata have been developed. A key to the success of these tools is the application of the well-known data-structure *Difference Bounded Matrices*, DBM for representing *clock constraints*.

2.1 Timed Automata

A timed automaton is a standard finite-state automaton extended with a finte collection of real-valued clocks. In a timed automaton, the nodes (also known as control nodes) are labelled with an *invariant* (a condition on clocks), and transitions are labelled with a *guard* (a condition on clocks), a *synchronisation action*, and a *clock reset* (a subset of clocks to be reset). 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 action (if it is an instantaneous switch from the current node to another) or a positive real number i.e. a time delay (if the automaton stays within a node letting time pass).

Consider the timed automaton of Fig. 1. It has two control nodes l_0 and l_1 and two real–valued clocks X and Y. A *state* of the automaton is of the form (l, s, t), where l is a control node, and s and t are non–negative reals giving the value of the two clocks X and Y. A control node is labelled with a condition (the invariant) on the clock values that must be satisfied for states involving this node. Assuming that the automaton starts to operate in the state $(l_0, 0, 0)$, it may stay in node l_0 as long as the invariant $X \leq 4$ of l_0 is satisfied. During this time the values of the clocks increase synchronously. Thus from the initial state, all states of the form (l_0, t, t), where $t \leq 4$, are reachable. The edges of a timed automaton may be decorated with a condition (guard) on the clock values that must be satisfied in order to be enabled. Thus, only at the states (l_0, t, t), where $1 \leq t \leq 4$, the edge from l_0 to l_1 is enabled. Additionally, edges may be labelled with synchronization actions and simple valuations reseting clocks. For instance, when following the edge from l_0 to l_1 the action a is performed to synchronize with the environment and the clock Y is reset to 0 leading to states of the form $(l_1, t, 0)$, where $1 \leq t \leq 4$.

For the formal definition, we assume a finite set of actions, A (the alphabet of the automata) for synchronisation and a finite set of real-valued variables C for clocks. We use $a, b, \ldots$ to range over A and $X_1, X_2, \ldots$ to range over C. We use $\mathcal{B}(C)$ ranged over by g and later by D to denote the set of conjunctive formulas of atomic constraints of the forms $X_i \sim m$ or $X_i - X_j \sim n$, where $X_i, X_j \in C$ are clocks, $\sim \in \{\leq, <, \geq, >\}$, and m, n are integer constants. The elements of $\mathcal{B}(C)$ are called *clock constraints*.

DEFINITION 1. *A timed automaton over actions A and clocks C is a tuple $\langle N, l_0, E, I \rangle$ where*

- *N is a finite set of nodes,*
- *$l_0 \in N$ is the initial node,*
- *$E \subseteq N \times \mathcal{B}(C) \times A \times 2^C \times N$ is the set of edges, and finally,*
- *$I : N \to \mathcal{B}(C)$ assigns invariants to nodes.*

When $\langle l, g, a, r, l' \rangle \in E$, we write $l \xrightarrow{g,a,r} l'$.

Formally, we represent the values of clocks as functions (called clock valuations) from C to the non–negative reals $\mathbb{R}_{\geq 0}$. We denote by $\mathcal{V}$ the set of clock valuations for C. A semantical *state* of an automaton is now a pair (l, u), where l is a node of the automaton and u is a clock valuation and the semantics of the automaton is given by a transition system with the following two types of transitions (corresponding to delay-transitions and action-transitions):

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

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

2.2 Symbolic Reachability Analysis

In general, the semantics of a timed automaton is an infinite and uncountable transition system, and is thus not an appropriate basis for decision algorithms. Efficient algorithms may be obtained using a *symbolic* semantics based on *symbolic states* of the form (l, D), where $D \in \mathcal{B}(C)$ [Henzinger *et al.* 1994],[Wang *et al.* 1994]. The symbolic counterpart to the standard semantics is given by the following two types of symbolic transitions:

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

where time progress $D^{\uparrow} = \{u + d \,|\, u \in D \wedge d \in \mathbb{R}_{\geq 0}\}$ and clock reset $r(D) = \{[r \mapsto 0]u \,|\, u \in D\}$. It may be shown that the set of clock constraints $\mathcal{B}(C)$ is closed under these two operations ensuring the well-definedness of the semantics [Larsen *et al.* 1995]. Moreover, the symbolic semantics fully characterizes the above concrete semantics in the following sense:

- whenever $(l_0, \{u_0\}) \rightsquigarrow^* (l, D)$ then $(l_0, u_0) \longrightarrow^* (l, u)$ for all $u \in D$
- whenever $(l_0, u_0) \longrightarrow^* (l, u)$, then $(l_0, \{u_0\}) \rightsquigarrow^* (l, D)$ for some D with $u \in D$

where u_0 may be any clock valuation, $\{u_0\} \in \mathcal{B}(C)$ denotes the clock constraint which is satisfied only by u_0 and $\longrightarrow^*$ denotes the transitive closure of the relation $\longrightarrow$.

Based on the symbolic semantics, a number of verification tools have been developed for real-time systems (e.g. Kronos [Daws and Yovine 1995] and

```
PASSED:= {}
WAIT:= {(l_0, D_0)}
repeat
    begin
        get (l, D) from WAIT
        if (l, D) |= φ then return "YES"
        else if D ⊈ D' for all (l, D') ∈ PASSED then
            begin
                add (l, D) to PASSED        (*)
                NEXT:={(l_s, D_s) : (l, D) ⇝ (l_s, D_s) ∧ D_s ≠ ∅}
                for all (l_s', D_s') in NEXT do
                    put (l_s', D_s') to WAIT
            end
    end
until WAIT={}
return "NO"
```

Fig. 2: An algorithm for symbolic reachability analysis.

UPPAAL [Bengtsson *et al.* 1996]). The abstract reachability algorithm implemented in these tools is shown in Fig. 2. 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 control–node and D is a clock constraint.

2.3 Difference Bounded Matrices

In the abstract reachability algorithm, we observe that data structures for representing clock constraints are crucial for efficient implementation. One such well–known data structure is that of difference bounded matrices (DBM, see [Bellman 1958] and [D.Dill 1989]), which offers a canonical representation for clock constraints.

To introduce the notion of DBM, we assume that the set of clocks C is organized as $\{X_1 \dots X_n\}$ and further assume an additional zero–clock X_0 which always has the value 0. Now a clock constraint in the form $X_i{\sim}m$ can be rewritten as $X_i - X_0{\sim}m$. In fact, all clock constraints in $\mathcal{B}(C)$ may be transformed to the conjunctions of constraints in the form $X_i - X_j \leq m$ or $X_i - X_j < n$ where m,n are integers. For instance, $X_i - X_j > 4$ is equivalent to $X_j - X_i < -4$. In the following, we assume that all clock constraints in $\mathcal{B}(C)$ include the implicit constraints on clocks: $X_0 - X_i \leq 0$ and $X_i - X_0 < \infty$.

Now, a clock constraint may be viewed as a set of upper bounds on the differences between pairs of clock variables, which may be represented as a matrix. Such a matrix is called a DBM. A DBM representation of a clock constraint D may be interpreted as a weighted, directed graph, where the vertices correspond to the clocks of $C = \{X_0, X_1 \dots X_n\}$. The graph has an edge from X_i to X_j with weight m if $X_j - X_i \leq m$ is a constraint of D. The case of strict ordering $<$ can be represented with an extra label on the weight values representing the fact that the difference is strict ($<$).

The advantage with the graph interpretation is that it provides the key to a canonical representation for clock constraints, which again enables efficient algorithms for performing constraint operations and inclusion checking. In general, the same set of clock valuations may correspond to several clock constraints and thus graphs. The canonical representation for a clock constraint is obtained by simply deriving the shortest–path closure for its graph and can thus be computed in time $\mathcal{O}(n^3)$, where n is the number of clocks of D. We call such closures *closed constraints*. For closed constraints D, the operations $D^{\uparrow}$ and $r(D)$ may be performed in time $\mathcal{O}(n)$. In addition, if D is closed, $D \subseteq D'$ (for any D') if and only if whenever $(X_i - X_j \leq m) \in D$ then $(X_i - X_j \leq m') \in D'$ such that $m \leq m'$, which may be checked in time $\mathcal{O}(n^2)$.

Finally we notice that the sets of clock valuations described by DBM's are convex sets (polyhedra in the n-dimensional real space $(\mathbb{R}_{\geq 0})^n$). Following the literature, we shall call such convex sets *zones*.

DBM's obviously consume space of order $\mathcal{O}(n^2)$. 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* [Larsen *et al.* 1997] is preferable 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.

2.4 Union of Zones

We know that the set $\mathcal{B}(C)$ of clock constraints is closed under conjunction, but it is not closed under disjunction, as the union of two convex sets is not necessarily a convex set. This leads to a problem in model checking algorithms for timed automata: termination of reachability analysis is detected by a testing for inclusion of a newly computed zone in the existing explored zones, thus making a test of inclusion of a zone in a union of zones desirable. We shall introduce a new data structure which is able to deal efficiently with finite unions of zones. We call such finite unions *federations*. Ad-hoc approaches to the problem of inclusion in a federation will explode in the number of zones belonging to the federation. There are some other approaches which are related to our problem, but will not help in solving it. In [Rokicki 1993] an $O(|V|^4)$ algorithm was presented which tested if the union of two DBM's could be represented by another DBM. This algorithm

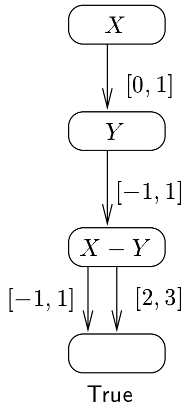

Fig. 3: Example CDD

however cannot be used for the inclusion test in general, as there exists convex sets S_1, S_2 and S_3 such that $S_1 \cup S_2 \cup S_3$ is a convex set, but the union of no two of the sets is convex. The approach in [Dechter *et al.* 1991] uses constraint systems where the individual constraints are of the form

$$X_i - X_j \in I_1 \cup I_2 \cup \cdots I_k$$

with $I_1, \ldots, I_k$ being intervals, thus allowing for disjunction in the individual constraints. These systems however can not represent all finite unions of convex sets. If each constraint has exactly two intervals, then a system with n constraints represents the union of the 2^n convex regions obtained by selecting one interval for each constraint.

The solution in this paper is to represent the constraints as a directed acyclic graph (DAG), a Clock Difference Diagram (CDD). The DAG given in Fig. 3 represents the constraint system

$$0 \leq X \leq 1 \ \wedge \ -1 \leq Y \leq 1 \wedge \big((-1 \leq X - Y 1) \ \vee \ (2 \leq X - Y \leq 3)\big)$$

CDD's are inspired by Binary Decision Diagrams (BDD's, see [Bryant 1986]) and Interval Difference Diagrams (IDD's, see [Strehl and Thiele 1998]). A BDD is an acyclic graph with two terminal nodes True and False, where each node represents a boolean variable. Inner nodes have two branches, one to be followed if the variable is true and the other if the variable is false. Given a specified ordering of the variables and insisting on maximal sharing of isomorphic subgraphs, each boolean expression has a unique representation. IDD's extend this idea to independent real valued variables, where the edges are now labelled with intervals of $\mathbb{R}$ instead of true and false. This approach is however not sufficient for the symbolic model checking of timed automata. CDD's further extend this idea where the nodes can represent differences of

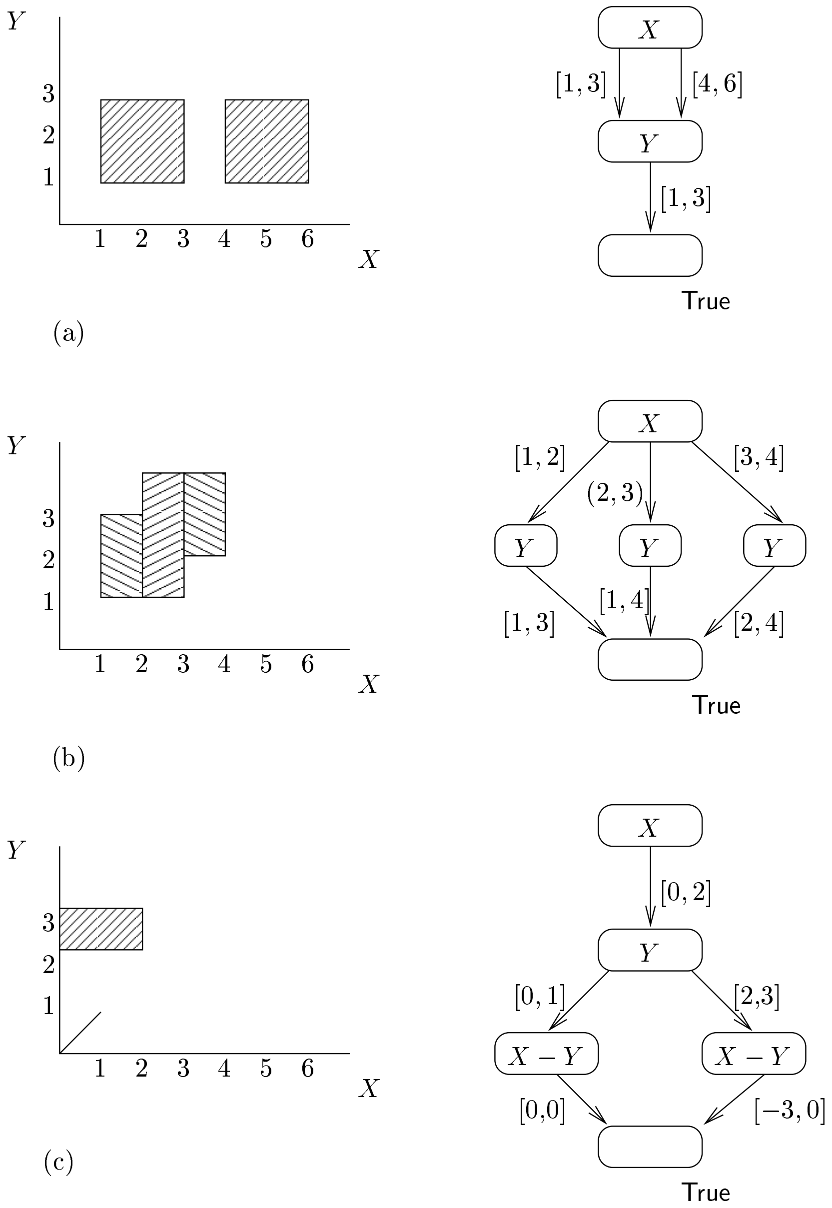

Fig. 4: Three example CDD's. Intervals not shown lead implicitly to False; e.g. in (a) there are arcs from the X-node to False for the three intervals $]-\infty,1[$, $]3,4[$, and $]6,\infty[$.

variables – hence dependencies between nodes – and it turns out that CDD's exactly represent unions of DBM's, offering an improved data structure for the symbolic model checking of timed automata.

3. Clock Difference Diagrams

This section defines *Clock Difference Diagram* (CDD). 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. Fig. 4 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 $1 \leq 0 < j \leq n$. 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 $\bot \in \mathcal{T}$, 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, so these are all open, closed and half-open intervals of the real line, which we will typically write as $(a, b), [a, b], (a, b]$ and $[a, b)$. 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:

- 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.
- 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).

As an example, if the type is $(2, 1)$ and $I = [3, 5)$, then $I(2, 1)$ would be the constraint $3 \leq X_2 - X_1 < 5$. For v where $v(X_2) = 9$ and $v(X_1) = 5.2$ we would find that $I(2, 1)(v)$ is true, while for v' with $v'(X_2) = 3$ and $v'(X_1) = 4$ we would have $I(2, 1)(v')$ is false.

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* $\mathsf{type} : V \to \mathcal{T}$ *and* $\mathsf{succ} : V \to 2^{\mathcal{I} \times V}$ *such that*

- *V has exactly two terminal nodes called* True *and* False, *where* $\mathsf{type}(\mathsf{True}) = \mathsf{type}(\mathsf{False}) = \bot$ *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.*

- *it has maximal sharing: for all $n, m \in V$, whenever $\mathsf{succ}(n) = \mathsf{succ}(m)$ then $n = m$,*
- *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. Fig. 4 gives some examples of CDD's. We omit all arcs going to False to improve readability[2]. These missing arcs can easily be deduced from the figures: assume a node has outgoing arcs labeled with intervals $I_1, \ldots, I_k$, then let

$$S_1 := \bigcup_{i \in \{1, \ldots, k\}} I_i, \qquad S_2 := \mathbb{R} \setminus S_1$$

and let $J_1, \ldots, J_\ell$ be intervals of the real line such that they are disjoint, their union is S_2 and they are maximal, i.e. the union of any pair of them is not an interval. Then to complete the CDD, arcs labeled by the J_j all going to False need to be added.

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*

[2] We should point out that all the algorithms in this paper require that all arcs including those leading to False to be specified.

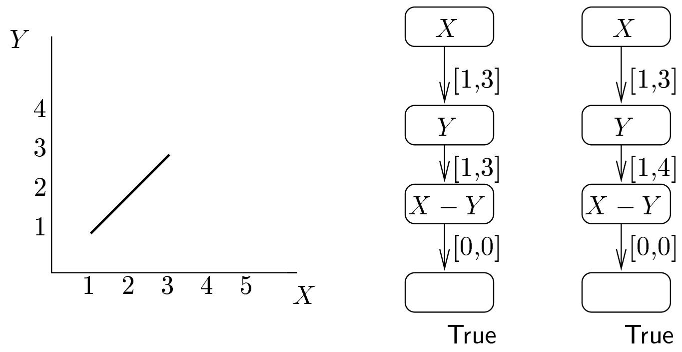

Fig. 5: Two CDD's for the same zone

- *for terminal nodes,* $[\![\mathsf{False}]\!] := \emptyset$ *and* $[\![\mathsf{True}]\!] := \mathcal{V}$
- *for inner nodes,* $[\![n]\!] := \{v \in \mathcal{V} \mid n \xrightarrow{I} m, I(\mathsf{type}(n))(v) = \mathsf{true}, v \in [\![m]\!]\}$

Any path $n_0 \xrightarrow{I_1} \ldots \xrightarrow{I_k} n_k$, with $n_k = \mathsf{True}$, in a CDD leading from some node n_0 to the true node can be seen as representing all the valuations which fulfill the constraints induced by this path. Thus similar to the previous definition we can assign a semantics to such paths by:

DEFINITION 4. *For paths in a* CDD *leading to true we can define their semantics recursively by:*

- $[\![\mathsf{True}]\!]_P := \mathcal{V}$,
- $[\![n_0 \xrightarrow{I_1} \ldots \xrightarrow{I_k} n_k = \mathsf{True}]\!]_P := \{v \in \mathcal{V} \mid I_1(\mathsf{type}(n_0))(v) = \mathsf{true},$
  $v \in [\![n_1 \xrightarrow{I_2} \ldots \xrightarrow{I_k} n_k]\!]_P\}$

Note that each path in the form $n_0 \xrightarrow{I_1} \ldots \xrightarrow{I_k} n_k$ represents just the conjunction of all the constraints $v(X_i) - v(X_j) \in I_\ell$ with $\mathsf{type}(n_\ell) = (i, j)$. Hence, a DBM can be constructed from the lower and upper bounds in the constraints, describing the same set of valuations as $[\![n_0 \xrightarrow{I_1} \ldots \xrightarrow{I_k} n_k]\!]_P$. The DBM constructed from a path will be called *the path's* DBM.

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 CDD's in Fig. 5 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

Algorithm 1 $\mathsf{op}\,(n_1, n_2,\, \mathsf{baseOp}\,)$- Applies the operation baseOp to the CDD's n_1 and n_2

if n_1 and n_2 are terminal nodes **then**
  return node equal to $n_1\, \mathsf{baseOp}\, n_2$.
end if
if $\mathsf{type}(n_1) \sqsubseteq \mathsf{type}(n_2) \wedge \mathsf{type}(n_1) \neq \mathsf{type}(n_2)$ **then**
  return the node $(\mathsf{type}(n_1), \{(I_1,\, \mathsf{op}\,(n_1', n_2,\, \mathsf{baseOp}\,)) \mid n_1 \xrightarrow{I_1} n_1'\}$
end if
if $\mathsf{type}(n_1) \sqsubseteq \mathsf{type}(n_2) \wedge \mathsf{type}(n_1) \neq \mathsf{type}(n_2)$ **then**
  return the node $(\mathsf{type}(n_2), \{(I_2,\, \mathsf{op}\,(n_1, n_2',\, \mathsf{baseOp}\,)) \mid n_2 \xrightarrow{I_2} n_2'\}$
end if
if $\mathsf{type}(n_1) = \mathsf{type}(n_2)$ **then**
  return the node
    $(\mathsf{type}(n_1), \{(I_1 \cap I_2,\, \mathsf{op}\,(n_1', n_2',\, \mathsf{baseOp}\,)) \mid n_1 \xrightarrow{I_1} n_1', n_2 \xrightarrow{I_2} n_2', I_1 \cap I_2 \neq \emptyset\}$
end if

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 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 a later section.

As a set of valuations can be seen as a subset in $\mathbb{R}^n$ (where n is the number of clocks), an important and natural question is: What subsets of $\mathbb{R}^n$ can be represented by a CDD? Informally a valuation is satisfied by a CDD if there exists a path from the root to True. In fact it is easy to show the following:

PROPOSITION 1. *Given a* CDD T *and a node* n *then*

$$[\![n]\!] = \bigcup \{[\![n = n_0 \xrightarrow{I} \cdots \xrightarrow{I_k} n_k = \mathsf{True}]\!]_P \mid n_0 \xrightarrow{I} \cdots \xrightarrow{I_k} n_k \textit{ is a path in } T\}$$

CDD's exactly represent finite unions of zones. The above proposition shows that any CDD can be seen as the union of a finite number of zones. Further down we will show that CDD's are closed under (finitary) union, thus any finite union of zones can be represented as a CDD.

4. Set-Theoretic Operations

The set of valuations that satisfy a CDD can be seen as a subset in $\mathbb{R}^n$ (where n is the number of clocks). All the standard set-theoretic operations like union, intersection and complement can be performed on CDD's. As an example the set-theoretic complement of a CDD is achieved by simply swapping the True and False nodes. Note that no such simple algorithm exists for DBM's which are not closed under complement.

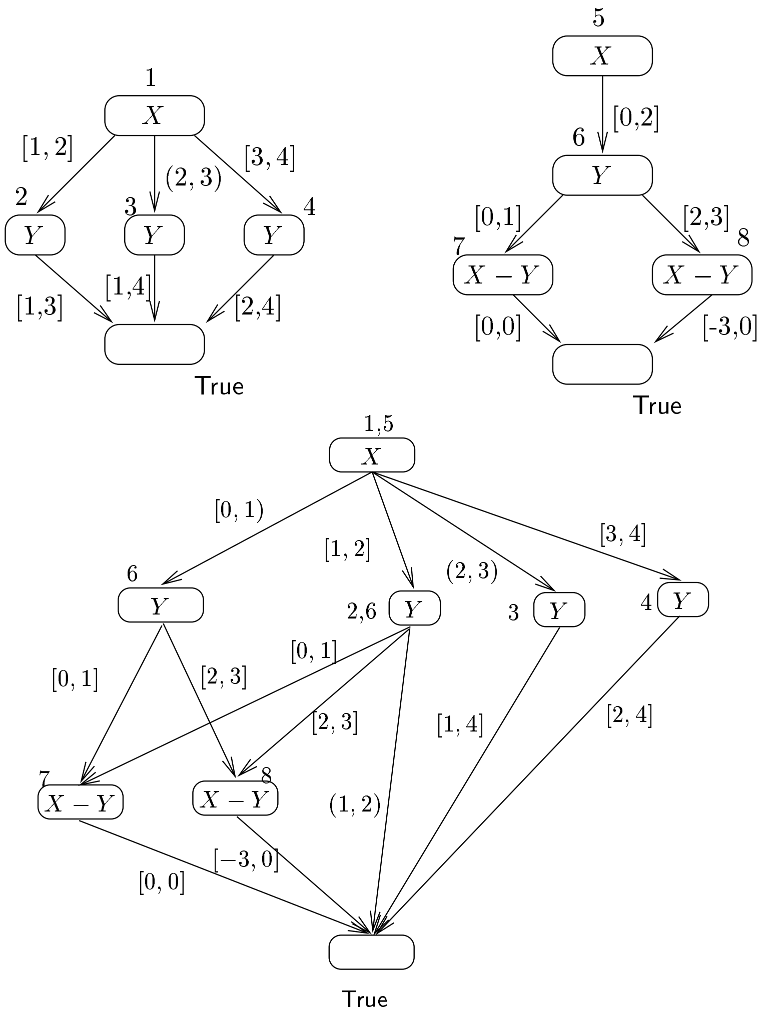

Fig. 6: Two CDD's and their union. Each node is given the same number in the original CDD's and their union.The label 1,5 is the union of node 1 and 5 likewise the label 2,6 is the union of node 2 and 6.

Binary set-theoretic operations can easily be defined recursively by first defining the base case for the terminal nodes suitably. Alg. 1 assumes the base cases – i.e. both nodes are either True or False – given by an operation baseOp. Remember for this base case operation, that the node True corresponds to the full set of valuations, while the node False represents the empty set. Under this assumption, the given algorithm defines the operation op for arbitrary nodes of the CDD. Fig. 6 show an example of the union operation applied to two nodes of the CDD.

Algorithm 2 Test for emptiness of $[\![n]\!]$

let P be the set of paths starting in n leading to True
while $P \neq \emptyset$ **do**
  extract and remove a path p from P
  test p for satisfiability
  if there is a valuation that satisfies p **then**
    return false
  end if
end while
return true

The same principle as for the binary case can be applied to any n-ary set-theoretic function. Note that our algorithm do not care about keeping reducedness. However, this can easily be achieved using standard BDD techniques like hashing on each node n and an operation-cache for memorizing the result of operations already performed on the same arguments. Extending the algorithm with these techniques leads to very efficient implementations [3].

Test for inclusion of one CDD in another can be realized by exploiting the well known set-theoretic equivalence

$$A \subseteq B \iff A \cap \neg B = \emptyset$$

As we do not yet have a normal or canonical form for CDD's, there is no straightforward syntactic way to test for emptiness of a CDD. However, a rather simple procedure follows immediately from Prop. 1: all that needs to be done is to test satisfiability of all paths leading to True. Satisfiability of a single path may itself be decided straightforwardly using the existing procedure for DBM's. Alg. 2 offers the details of this approach to set-inclusion.

Alg. 2 may be specialised when testing if a zone Z defined by a DBM is included in a CDD. This specialized version for set–inclusion has proved particular usefull in obtaining an efficient CDD-based version of UPPAAL's reachability algorithm. We report on this in a later section and refer for more information to [Behrmann *et al.* 1999].

Note that when testing for emptiness of a DBM as required in the first if-statement of Alg. 3, we need to compute its canonical form. Once this canonical form of D has been obtained, we may improve the efficiency of algorithm by passing $D \wedge I(\mathsf{type}(n))$ [4] also in canonical form. As the conjunction of $I(\mathsf{type}(n))$ adds no more than two constraints to that of D, computation of the canonical form for $D \wedge I(\mathsf{type}(n))$ can be done faster than in the general case (in time $\mathcal{O}(n^2)$ rather than $\mathcal{O}(n^3)$). We illustrate

[3] More precisely, the operations become linear in the product of the sizes of the argument CDD's.

[4] $\wedge$ refers to the operation of taking the DBM for D and conjoin the constraints of $I(\mathsf{type}(n))$.

Algorithm 3 Deciding set inclusion for a zone and a CDD

$\mathsf{subset}(D, n)$
if $D = \mathsf{False}$ or $n = \mathsf{True}$ **then**
  return True
else if $n = \mathsf{False}$ **then**
  return False
else
  return $\bigwedge_{n \xrightarrow{I} m} \mathsf{subset}(D \wedge I(\mathsf{type}(n)), m)$
end if

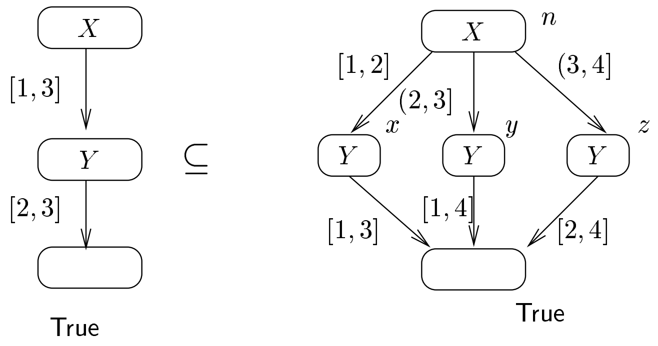

Fig. 7: Example zone for inclusion

the action of Alg. 3 on the example Fig. 7, where the nodes are named n, x, y and z for reference. First we call $\mathsf{subset}(X \in [1,3] \wedge Y \in [2,3], n)$ which is expanded as follows (we have added the implicit branches to the False node not shown in the picture):

$$
\begin{aligned}
&\mathsf{subset}(X \in [1,3] \wedge Y \in [2,3], n) = \\
&\quad \mathsf{subset}(X \in [1,3] \wedge Y \in [2,3] \wedge X \in (-\infty, 1), \mathsf{False}) \wedge \\
&\quad \mathsf{subset}\ (X \in [1,3] \wedge Y \in [2,3] \wedge X \in [1,2], x) \wedge \\
&\quad \mathsf{subset}(X \in [1,3] \wedge Y \in [2,3] \wedge X \in (2,3], y) \wedge \\
&\quad \mathsf{subset}(X \in [1,3] \wedge Y \in [2,3] \wedge X \in (3,4], z) \wedge \\
&\quad \mathsf{subset}(X \in [1,3] \wedge Y \in [2,3] \wedge X \in (4, \infty), \mathsf{False}) \\
&\qquad = \textit{(simplification of first arguments)} \\
&\quad \mathsf{subset}(\mathsf{False}, \mathsf{False}) \wedge \\
&\quad \mathsf{subset}\ (X \in [1,2] \wedge Y \in [2,3], x) \wedge \\
&\quad \mathsf{subset}(X \in (2,,3] \wedge Y \in [2,3], y) \wedge \\
&\quad \mathsf{subset}(\mathsf{False}, z) \wedge \\
&\quad \mathsf{subset}(\mathsf{False}, \mathsf{False})
\end{aligned}
$$

Thus we only need to pursue the calculation of two conjuncts, as the others are readily satisfied. In both cases one additional expansion of subset followed by constraint simplification yields the final result 'true'. In this particular application, the algorithm never reaches a stage, where a call of the form $\mathsf{subset}(D, \mathsf{False})$ is made with D a non-empty zone.

The application also demonstrates another important point: when using either Alg. 2 combined with an intersection and complement operation or Alg. 3, several inclusion checks may be terminated early without having to go through the whole CDD. For example, if instead in the above example the zone D was given by the clock constraint $X \in [6,7] \wedge Y \in [1,3]$, the algorithm would terminate at the very first step as there would be a call $\mathsf{subset}(D, \mathsf{False})$, where D is a non-empty constraint system.

5. Clock Operations for Fully Symbolic Analysis

Two operations play an important role in the analysis of timed automata: future (letting time progress) and reset (set a clock to an integer value). The two operations can easily be defined semantically on sets of valuations. Given a valuation v, a clock X and a positive real d, let $v+d$ be the valuation where each clock is increased by d. Further $[X \mapsto 0]v$ denotes the valuation where clock X has values 0, while all other clocks retain their value from v. Now applying pointwise extension we obtain the following future and reset operations on sets of valuations:

$$\begin{array}{lcl} \mathsf{future(S)} & := & \{v + d \mid v \in S, d \in \mathbb{R}_{\geq 0}\} \\ [X \mapsto 0]S & := & \{[X \mapsto 0]v \mid v \in S\} \end{array}$$

These operations can be carried out for zones using DBM's which are in canonical form. For a canonical DBM $D = \{d_{ij}\}$, the future is computed by

- removing the upper bound on all individual clocks, i.e. set d_{i0} to $+\infty$ for all i

Assuming that j is the index of clock X, the reset of D is computed by

- setting clock X to 0, i.e. $d_{j0} = d_{0j} = 0$, and
- remove all upper bounds on the differences between X and other clocks

In order to extend these operations to CDD's, we need to bring a CDD in a suitable canonical form. The basic idea is to generate an equivalent CDD where all paths to True are tightened, i.e. the corresponding DBM's are in canonical form:

DEFINITION 5. *A* CDD *is called tightened iff for all paths leading to* True, *the path's* DBM *is in canonical form.*

Algorithm 4 Turn a CDD starting at n into a tightened CDD

$new := n$
repeat
  $old := new$
  let $new :=$ False
  for all paths P leading from old to True **do**
    compute P's DBM and it's canonical DBM D
    if D does not represent the empty set **then**
      let d be the node corresponding to D
      let $new := new$ union d
    end if
  end for
until $old = new$
return new

Alg 4 shows a way to bring any CDD into such a form. Observe that in the tightening and unioning step, intervals from nodes can only become smaller. As there are smallest intervals – i.e. they cannot be further divided by the algorithm – namely those of the form $[c, c]$ and $(c, c+1)$ (where c is an integer), termination of the algorithm is guaranteed. The fact that all paths in the resulting CDD are DBM's in canonical form follows from the construction: only in this case can the result be stable.

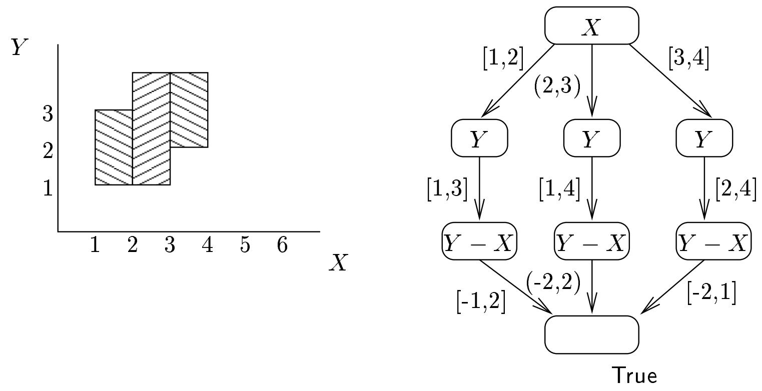

Fig. 8: A tightened CDD

Note that in a tightened CDD, a path leading to True cannot have contradicting constraints, i.e. it cannot represent the empty set. This is because the path's DBM would have a negative entry along, while this is ruled out by the definition of the path's DBM. Fig. 8 shows the tightened CDD-representation for Example (b) from Fig. 4.

Starting from a tightened CDD, future and reset can now be defined by extending the definition of the operations on DBM's. For a given interval

$I \neq \emptyset$, let I^∞ be this interval with its upper bound being extended to $+\infty$, i.e. $I^\infty := \{x \in \mathbb{R} \mid \exists x_0 \in I . x_0 \leq x\}$. Then to compute the future of a tightened CDD, it is sufficient to

- replace each interval I on an edge leaving a node of type $(i, 0)$ by I^∞.

This is clearly the analogue of removing the upper bound on all constraints on the individual clocks in a DBM. The only problem is that the result is not a CDD in the sense of Def. 2, as the intervals within a node are not disjoint any more. Further down we explain how to deal with this situation.

To compute the reset of X_j to 0, assuming again a tightened CDD, we need to

- replace each interval I on an edge leaving a node of type $(j, 0)$ by $[0, 0]$, and
- replace each interval I on an edge leaving a node of type (j, i) respectively (j, i) where $i > 0$ by $(-\infty, +\infty)$.

This is again the extension of the reset operation to canonical DBM's. Note that the correctness of both operations therefore follows from the canonicity of the CDD, the correctness of the operations on DBM's, and the fact that both operations distribute over union of DBM's together with Prop. 1.

The reset as the future operation leave us however with DBM's violating the disjointness condition. As the semantics of DBM's is defined as in Def. 3, this is in principle not a problem: a valuation belongs to the set represented by a CDD if there is some path representing it. The fact that there might be several paths including it yields no problem. It should however be noted that all our algorithms require disjoint intervals out of nodes to be correct. Luckily it is rather easy to obtain an equivalent CDD with the disjointness property. Alg. 5 shows how to do this for one node, and must thus be applied to all nodes of the CDD from bottom to top. Note that the direction is important, as the algorithm uses the union operation which is only well-defined on nodes with disjoint intervals.

Fig. 9 shows how to compute the future of example (b) from Fig. 4, leaving us with a CDD violating disjointness. Fig. 10 then shows the CDD after making the first node disjoint again.

The existence of all the operations ensures that it is possible to do a fully symbolic model checking of timed automata by treating the discrete part (locations) by conventional (multi-terminal) BDD's and the continuous part (clocks) by CDD's.

6. A Relative Normal Form

A normal form offers the advantage of reducing the test for semantic equality to a syntactic test, if the normal form is unique. In the case of BDD's and

Algorithm 5 Forcing a node n to contain disjoint intervals only

let $new := n$
while there are overlapping intervals in new **do**
  $old := new$
  let $(I_1, n_1), (I_2, n_2) \in \mathsf{succ}(old)$ such that $I_1 \cap I_2 \neq \emptyset$
  $M := \{(I, n') \mid (I, n') \in \mathsf{succ}(old), I \neq I_1, I \neq I_2\}$
  $M := M \cup \{(I_1 \cap I_2, n_1 \text{ union } n_2)\}$
  if $I_1 \setminus I_2 \neq \emptyset$ **then**
    $M := M \cup \{(I_1 \setminus I_2, n_1)\}$
  end if
  if $I_2 \setminus I_1 \neq \emptyset$ **then**
    $M := M \cup \{(I_2 \setminus I_1, n_2)\}$
  end if
  $new := (\mathsf{type}(old), M)$
end while
return new

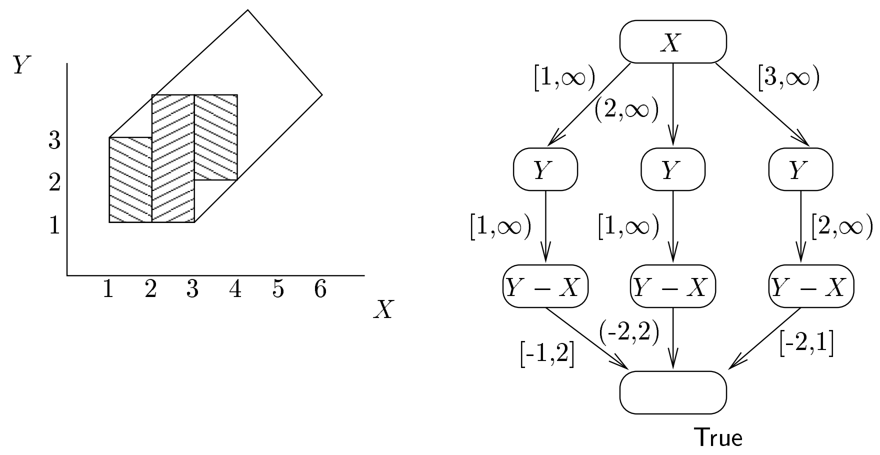

Fig. 9: First step in computing the future of example Fig. 4 (b)

IDD's, normal forms come along with reducedness of the data structure. For DBM's, the canonical form is a normal form. Already in Sec. 3 we have seen that reduced CDD's are not unique. But even a tightened CDD as defined in the previous section is not unique: Fig. 11 gives two CDD's which are tightened and semantically equivalent, but not graph-isomorphic.

The main problem here is granularity – the left hand CDD chooses a different granularity for X than the right hand CDD. Note that by requiring that all intervals are as small as possible – i.e. either of the form $[c, c]$ or $(c, c+1)$ for an integer c – together with reducedness and canonicity one would arrive at a unique normal form. In fact under this assumption CDD's are equivalent

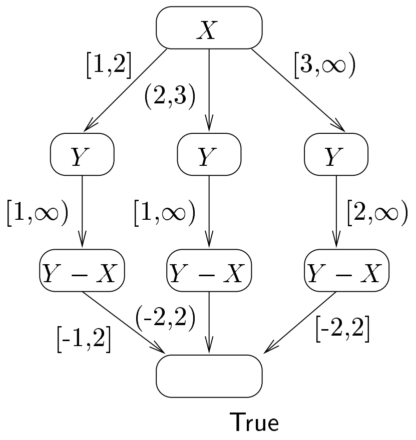

Fig. 10: The obtained CDD in computing the future of example Fig. 4 (b)

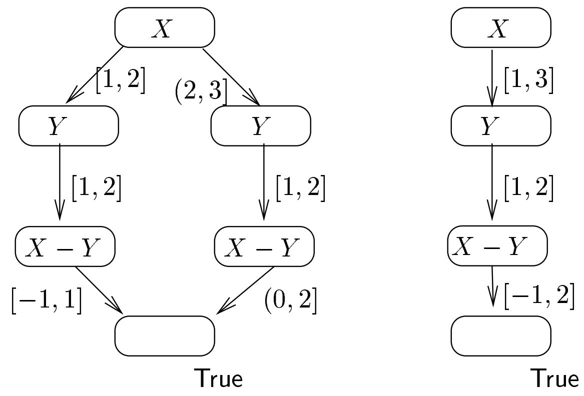

Fig. 11: Two equivalent tightened CDD's

to NDD's ([Asarain *et al.* 1997]), which has a unique normal form. However for reasons of time- and space-efficiency, the granularity of a CDD should be as coarse as possible. Of special interest are data structures which are invariant to rescaling of the timed automata (see e.g. [Weise and Lenzkes 1997]).

In the rest of this section we will present a *relative normal form*. Given two CDD's, the relative normal form gives us a coarse representation for both of them. Their relative normal forms will be isomorphic if the original CDD's are equivalent. The basic idea is to define the granularity of each of them in the term's of the other CDD's granularity. In order to formalize this, we define the notion of one CDD being *finer* than another:

Algorithm 6 n_1 $\mathsf{refinedBy}$ n_2: Making a node n_1 finer than n_2

let $new := n_1$
if $n_2 = \mathsf{True}$ or $n_2 = \mathsf{False}$ or $n_1 = \mathsf{False}$ **then**
  return new
end if
if $\mathsf{type}(n_1) = \mathsf{type}(n_2)$ **then**
  $new := (\mathsf{type}(n_1), \{(I_1 \cap I_2, n_1' \ \mathsf{refinedBy}\ n_2') \mid (I_1, n_1') \in \mathsf{succ}(n_1), (I_2, n_2') \in \mathsf{succ}(n_2), I_1 \cap I_2 \neq \emptyset\}$
else if $\mathsf{type}(n_1) \sqsubseteq \mathsf{type}(n_2)$ **then**
  $new := (\mathsf{type}(n_2), \{(I, n_1 \ \mathsf{refinedBy}\ n_2') \mid (I, n_2') \in \mathsf{succ}(n_2)\})$
else if $\mathsf{type}(n_2) \sqsubseteq \mathsf{type}(n_1)$ **then**
  $new := (\mathsf{type}(n_1), \{(I, n_1' \ \mathsf{refinedBy}\ n_2) \mid (I, n_1') \in \mathsf{succ}(n_1)\}$
end if
return new

DEFINITION 6. *Given two* CDD*'s starting at nodes n_1 and n_2 resp., we say that n_1 is finer than n_2 iff*

- $n_1 = \mathsf{False}$, *or*
- $n_2 = \mathsf{True}$, *or* $n_2 = \mathsf{False}$, *or*
- *both are inner nodes with* $\mathsf{type}(n_1) = \mathsf{type}(n_2)$, *and for each* $(I, n_1') \in \mathsf{succ}(n_1)$ *there is* $(J, n_2') \in \mathsf{succ}(n_2)$ *such that* $I \subseteq J$ *and* n_1' *is finer than* n_2'.

We say that n_1 and n_2 have the same granularity iff n_1 is finer than n_2 and vice versa.

Given n_1 and n_2, there is an algorithm to find a CDD n which is equivalent to n_1 and finer than n_2, see Alg. 6. Thus for two given CDD's, it is easy to find equivalent CDD's which are of same granularity by applying Alg. 6 to them in both ways. For these CDD's, equality can be decided pure syntactically:

THEOREM 1. *Let n_1 and n_2 be two* CDD*'s which are tightened and of same granularity. Then $[\![n_1]\!] = [\![n_2]\!]$ iff n_1 and n_2 are graph-isomorphic.*

PROOF. The if-part follows immediately from the definition of $[\![.]\!]$. So assume two tightened CDD's of same granularity which describe the same federation. Let v be some valuation of the federation. Then there is a unique path in n_1 which satisfies this valuation, due to disjointness. This path must lead to True. As n_1 is finer than n_2, we can construct a similar path in n_2, which has the same number of nodes, but the intervals might be larger than those along the path in n_1. By symmetry, this holds in the opposite direction, and thus the two paths are the same in both graphs. As the CDD's are tightened, paths to True cannot represent the empty set, so therefore all paths to True are the same in both graphs. The rest of the graph must be the same, as it can be completed from the True-paths the same way as we mentioned for our drawings in Sect. 3. Thus the two CDD's are graph-isomorphic. □

Note that an implicit result of the above theorem is that there exists a unique normal form for CDD's with the minimal granularity. It would be more interesting to find the unique normal form with maximal granularity, which gives the coarsest representation of CDD's. We conjecture that such a normal form exists; unfortunately we are not able to prove this conjecture.

7. Implementation and Experimental Results

We have implemented a CDD-package and used it to obtain a modified, CDD-based reachability algorithm for UPPAAL. The full details of this implementation may be found in [Behrmann *et al.* 1999].

In this section we present the results obtained from the experiment, which applied both the current version of UPPAAL [5] and the CDD-based version of UPPAAL to the verification of nine industrial examples (mostly) found in the literature. The examples include a gearbox controller [Lindahl *et al.* 1998], various communication protocols used in Philips audio equipment [Bosscher *et al.* 1994], [D'Arginio *et al.* 1997],[Bengtsson *et al.* 1996], and in B&O audio/video equipment [Havelund *et al.* 1997],[Havelund *et al.* 1998], the start-up algorithm of the DACAPO protocol [Lönn *et al.* 1997], and an Advanced Field Bus Protocol (AF100). In addition the comparison of performance was made on Fischer's Protocol for mutual exclusion.

In Table I 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 two different CDD-based algorithms. The first (CDD) uses CDD's to represent the continuous part of the PASSED-list, and the second (Reduced) is identical to CDD except that all inconsistent paths are removed from the CDD's. 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 6%). When inconsistent paths are eliminated the average space-saving increases to 55% at the cost of an average increase in run-time of 35%. 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%.

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

TABLE I: 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.

System	P	V	C	Current Time	Current Space	CDD Time	CDD Space	Reduced Time	Reduced Space
PHILIPS	4	4	2	0.2	25	0.2	23	0.2	23
PHILIPS COL	7	13	3	21.8	2,889	23.0	1,506	28.8	1,318
B&O	9	22	3	56.0	5,793	55.9	2,248	63.4	2,240
BRP	6	7	4	22.1	3,509	21.3	465	46.5	448
POWERDOWN1	10	20	2	81.3	4,129	79.2	1,539	82.6	1,467
POWERDOWN2	8	20	1	19.3	4,420	19.8	4,207	19.7	4,207
DACAPO	6	12	5	55.1	4,474	57.1	2,950	64.5	2,053
GEARBOX	5	4	5	10.5	1,849	11.2	888	12.4	862
AF100	16	32	4	283.7	23,063	269.2	8,993	289.7	8,993
FISCHER4	4	1	4	1.1	129	1.4	96	2.5	48
FISCHER5	5	1	5	40.6	1,976	61.5	3,095	154.4	396

8. Conclusion

In this paper Clock Difference Difference Diagrams (CDD's) have been presented. It has been shown that CDD's are able to represent finite unions of zones (that is regions representable by DBM's), and that all set-theoretic operations and additionally all operations necessary for timed reachability analysis of timed automata can be defined and computed on CDD's. These operations do not need a normal form, in contrast to DBM's and IDD's. In particular an algorithm is presented which decides CDD inclusion without converting any intermediate results to a unique normal form. In conclusion, CDD's can be used for fully symbolic model checking of timed systems. An implementation of a CDD-package for the real-time verification tool UPPAAL has shown that CDD's are useful in practice, leading to considerable space-savings.

We have given a relative normal form for CDD's, which is relative to a notion of granularity. An implication of this result is that there exists a unique normal form for CDD's, which allows for semantic equality to be reduced to a test for syntactic identity. In fact, the unique normal form of a CDD is its equivalent CDD with finest granularity, which is an NDD or BDD-like encoding of the region graphs induced by the original CDD. It would be more interesting to find the unique normal form with maximal granularity, which gives the coarsest representation of CDD's if such a normal form exists. However it is obviously too costly to compute such a normal form. We should emphasize that such a normal form with maximal granularity is not necessary for application of CDD's to timed model-checking because all the necessary operations can be carried out without a normal form.

For future work, we want to investigate more efficient implementation of the CDD-operations aiming at a fully symbolic model-checker for timed automata combining BDD and CDD.

Acknowledgement

The authors thank Gerd Behrmann for the implementation of the CDD package and collaboration on [Behrmann *et al.* 1999] related to this paper.

References

Alur, R. and Dill, D. 1990. Automata for Modelling Real-Time Systems. In *Proceedings of ICALP'90*, Volume 443 of *Lecture Notes in Computer Science*. Springer.

Asarain, E., Bozga, M., Maler, O., Pneuli, A., and Rasse, A. 1997. Data-Structures for the Verification of Timed Automata. In *Proceedings of HART'97*, Volume 1201 of *Lecture Notes in Computer Science*. Springer Verlag, 346–360.

Balarin, Felice. 1996. Approximate Reachability Analysis of Timed Automata. In *Proc. Real-Time Systems Symposium, Washington, DC*, 52–61.

Behrmann, Gerd, Larsen, Kim G, Pearson, Justin, Weise, Carsten, and Wang, Yi. 1999. Efficient Timed Reachability Analysis using Clock Difference Diagrams.

Bellman, Richard. 1958. On a routing problem. *Quarterly of Applied Mathematics 16(1)*, 87–90.

Bengtsson, Griffioen, Kristoffersen, Larsen, Larsson, Pettersson, and Yi. 1996. Verification of an Audio Protocol with Bus collision Using UPPAAL. In *Proceedings of CAV'96*, Volume 1102.

Bengtsson, Johan, Larsen, Kim G. Larsson, Fredrik, Pettersson, Paul, and Yi, Wang. 1996. Uppaal in 1995. In *Proc. of the 2nd Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, Number 1055 in Lecture Notes in Computer Science. Springer–Verlag, 431–434.

Bosscher, D., Polak, I., and Vaandrager, F. 1994. Verification of an Audio-control Protocol. In *Proceedings of Formal Techniques in Real-Time Fault-Tolerant Systems*, Volume 863 of *Lecture Notes in Computer Science*. Springer Verlag.

Bozga, Marius, Maler, Oded, Pnueli, Amir, and Yovine, Sergio. 1997. Some progress in the symbolic verification of timed automata. In *Proceedings of CAV'97*, Volume 1254 of *LNCS*, 179–190.

Bryant, Randel E. 1986. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers C-35*, (August), 677–691.

Burch, J. R., Clarke, E. M., McMillan, K. L., Dill, D. L., and Hwang, L. J. 1990. Symbolic Model Checking: 10^{20} states and beyond. In *Fifth Annual IEEE Symposium on Logic in Computer Science*, Volume Fifth Annual of *Log*. IEEE Computer Society Press, 428–439.

Campos, S.V. and Clarke, E.M. 1995. Real-time symbolic model checking for discrete time models. In *AMAST Series in Computing: Theories and Experiences for Real-Time System Development*.

D'Arginio, Katoen, Ruys, and Tretmans. 1997. Bounded retransmission protocol must be on time ! In *Proceedings of TACAS'97*, Volume 1217 of *Lecture Notes in Computer Science*. Springer Verlag.

Daws, C. and Yovine, S. 1995. Two examples of verification of multirate timed automata with Kronos. In *Proc. of the 16th IEEE Real-Time Systems Symposium*, 66–75.

D.Dill. 1989. Timing assumptions and verification of finite-state systems. *Lecture Notes in Computer Science 407*, 197–212.

DECHTER, RINA, MEIRI, ITAY, AND PEARL, JUDEA. 1991. Temporal Constraint Networks. *Artificial Intelligence 49*, 61–95.

HAVELUND, K., SKOU, A., LARSEN, K.G., AND LUND, K. 1997. Formal Modelling and Analysis of an Audio/Video Protocol: An Industrial Case Study using UPPAAL. In *Proceedings of 18th IEEE Real-Time Systems Symposium*.

HAVELUND, K., SKOU, A., LARSEN, K.G., AND LUND, K. 1998. Formal Verification of an Audio/Videoo Power Control using the Real-Time Model Checker UPPAAl. Tech. report, Bang& Olusfen.

HENZINGER, THOMAS A. HO, PEI-HSIN, AND WONG-TOI, HOWARD. 1995. A Users Guide to HYTECH. Tech. report, Department of Computer Science, Cornell University.

HENZINGER, THOMAS. A, NICOLLIN, XAVIER, SIFAKIS, JOSEPH, AND YOVINE, SERGIO. 1994. Symbolic Model Checking for Real-Time Systems. *Information and Computation 111*, 2, 193–224.

LARSEN, WEISE, YI, AND PEARSON. 1998. Clock Difference Diagrams. Tech. report, DoCS, Uppsala University, Sweden.

LARSEN, KIM G., LARSSON, FREDRICK, PETTERSSON, PAUL, AND WANG, YI. 1997. Efficient Verification of Real-Time Systems: Compact Data Structure and State-Space Reductions. In *Proceedings of the 18th IEEE Real-Time Systems Symposium. San Francisco, California.*.

LARSEN, KIM G., PETTERSON, PAUL, AND WANG, YI. 1995. Compositional and Symbolic Model-Checking of Real-Time Systems. In *Proceedings of the 16th Real-Time Systems Symposium.* IEEE Computer Society Press, 76–87.

LINDAHL, M., PETTERSSON, P., AND YI, W. 1998. Formal Design and Analysis of a Gear Controller. *Lecture Notes in Computer Science 1384*, 281–297.

LÖNN, H., PETTERSSON, P., AND YI, W. 1997. Formal Verification of a TDMA Protocol Start-Up Mechanism. In *Proceedings of 1997 IEEE Pacific Rim International Symposium on Fault-Tolerant Systems*, 235–242.

MØLLER, LICHTENBERG, ANDERSEN, AND HULGAARD. 1999. Difference Decision Diagrams. Tech. Report IT-TR-1999-023, Technical University of Dentmark.

MØLLER, LICHTENBERG, ANDERSEN, AND HULGAARD. 1999. On the symbolic verification of timed systems. Tech. Report IT-TR-1999-024, Technical University of Denmark.

R.L.SPELBERG, H.TOETENEL, AND M.AMMERLAAN. 1998. Partition refinement in Real-Time Model Checking. *Lecture Notes in Computer Science 1486*, 143–157.

ROKICKI, TOMAS GERHARD. 1993. *Representing and Modelling Digital Circuits.* PhD thesis, Department of Computer Science, Stanford University.

STREHL, KARSTEN. 1998. Using Interval Diagram Techniques for the Symbolic Verification of Timed Automata. Tech. Report TIK-53, Institut für Technische Informatik und Kommunikationsnetze (TIK), ETH Zürich.

STREHL, KARSTEN AND THIELE, LOTHAR. 1998. Symbolic Model Checking of Process Networks Using Interval Diagram Techniques. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD-98)*, 686–692.

STREHL, KARSTEN AND THIELE, LOTHAR. 1998. Symbolic Model Checking Using Interval Diagram Techniques. Tech. Report 40, Computer Engineering and Networks Lab (TIK), Swiss Federal Institute of Technolog(ETH), Gloriastrasse 35, 8092 Zuriech, Switzerland.

WANG, YI, PATTERSSON, PAUL, AND DANIELS, MATS. 1994. Automatic Verification of Real-Time Communicating Systems By Constraint-Solving. In *Proceedings of the 7th International Conference on Formal Description Techniques.*

WEISE, CARSTEN AND LENZKES, DIRK. 1997. Efficient Scaling-Invariant Checking of Timed Bisimulation. In *Proceedings of the 14th Annual Symposium on Theoretical Aspects of Computer Science (STACS'97)*, Volume 1200 of *LNCS*, 177–199.

WONG-TOI, HOWARD AND DILL, DAVID L. 1995. Verification of real-time systems by successive over and under approximation. In *Proc. International Conference on Computer-Aided Verification.*