Data-Structures for the Verification of Timed Automata*

Eugene Asarin[3] Marius Bozga[1] Alain Kerbrat[1]
Oded Maler[1] Amir Pnueli[2] Anne Rasse[1]

[1] VERIMAG, Centre Equation, 2, av. de Vignate, 38610 Gières, France,
Oded.Maler@imag.fr
[2] Dept. of Computer Science, Weizmann Inst. Rehovot 76100, Israel,
amir@wisdom.weizmann.ac.il
[3] Institute for Information Transmission Problems, 19 Bol. Karetnyi per., 101447
Moscow, Russia, asarin@ippi.ac.msk.su

Abstract. In this paper we suggest *numerical decision diagrams*, a BDD-based data-structure for representing certain subsets of the Euclidean space, namely those encountered in verification of timed automata. Unlike other representation schemes, NDD's are *canonical* and provide for all the necessary operations needed in the verification and synthesis of timed automata. We report some preliminary experimental results.

1 Introduction

Consider a transition system $\mathcal{A} = (Q, \delta)$, where Q is the set of *states* and $\delta : Q \mapsto 2^Q$ is a *transition function*, mapping each state $q \in Q$ into the set of *q-successors* $\delta(q) \subseteq Q$.

The problem of calculating or characterizing all the states reachable from a subset $F \subseteq Q$ of the state-space is one of the central problems in verification. The basic algorithm to calculate this set of states is the following:

$$
\begin{array}{l}
F_0 := F \\
\textbf{for } i = 0, 1, \ldots, \textbf{ repeat} \\
\quad F_{i+1} := F_i \cup \delta(F_i) \\
\textbf{until } F_{i+1} = F_i
\end{array}
$$

where $\delta(F_i) = \bigcup_{q \in F_i} \delta(q)$.

Symbolic methods [BCM+93], [McM93] have proved to be a very useful tool in the analysis of large discrete transition systems composed of many interacting components. Instead of transforming the description of the system into an enormous "flat" transition table over[4] $\mathbb{B}^m$, on which reachability analysis is practically impossible, these methods represent the transition relation as a formula over the state variables. Given such a formula $\mathcal{T}$ and a formula P describing

---

* This research was supported in part by the European Community projects HYBRID EC-US-043 and INTAS-94-697. VERIMAG is a joint laboratory of CNRS and UJF.

[4] We use $\mathbb{B}$ for $\{0, 1\}$ and $\mathbb{R}$ for the non-negative reals.

the subset F of the state-space one can calculate a new formula P' characterizing the set $\delta(F)$ of immediate successors of F. Iterating the procedure until a fixed-point is reached yields a formula P^* characterizing the set of all states reachable from F. When δ is expressed by a formula $\mathcal{T}(X, X')$, and F by a formula $P(X)$. the above algorithm can be reformulated as:

$$
\begin{array}{l}
P_0(X) := P(X) \\
\textbf{for } i = 0, 1, \ldots, \textbf{ repeat} \\
\qquad P_{i+1}(X) := P_i(X) \vee \exists Y\, (P_i(Y) \wedge \mathcal{T}(Y, X)) \\
\textbf{until } P_i(X) = P_{i+1}(X)
\end{array}
$$

The essence of any symbolic method is a data-structure for representing sets (equivalently, the formulas characterizing them) on which the above operations can be performed, in particular the forward (or backward) projection (line 3), boolean operations and equivalence testing. Binary decision diagrams (BDD's) [Bry86] are such a data-structure for boolean domains. The calculation of the forward projection is relatively-easy on large practical problems and the space requirements for the representations are reasonable. Given an ordering of the variables, BDD's also have the canonicity property: all equivalent formulas lead to the same BDD and equivalence testing is thus trivial.

The verification of timed automata introduces an additional ingredient, that is, a set of continuous variables (clocks) ranging over non-countable domains. The dynamics of the passage of time cannot be captured by a "next-state" transition relation, and symbolic methods are unavoidable as states and trajectories cannot be enumerated. The sets encountered in reachability analysis of timed automata are thus certain subsets of $\mathbb{B}^m \times \mathbb{R}^d$. While the discrete part is standard, the subsets of $\mathbb{R}^d$ that need to be represented and manipulated are what we call *k-polyhedral sets*, namely sets definable by a boolean combination of basic inequalities of the form $x_i < c$, $x_i \leq c$, $x_i - x_j < c$ and $x_i - x_j \leq c$, for $i, j \in \{1, \ldots, d\}$ and $c \in \{0, \ldots, k\}$. Such polyhedral sets have been called *regions* in [AD94].

As long as these polyhedra are convex (i.e., definable by conjunctions of basic inequalities and their negations), there exists a canonical representation, the difference bounds matrix (DBM, see for example [Dil89]). This is a $(d+1) \times (d+1)$ matrix with entries taken from $\{0, \ldots, k\}$ denoting the constants in a non-redundant set of inequalities whose intersection forms the region. For this representation, the intersection is done very easily via min and max operations. The forward and backward projections via elimination of the time quantifier are also done very efficiently on DBM's. Things however get complicated when we have arbitrary unions of convex polyhedra. In this case there is no unique representation and most tools represent such sets as a list of DBM's. The more "non-convex" the set becomes, more matrices are required in order to represent it and this makes equivalence testing and redundancy elimination difficult. Moreover, it is not clear how this representation is to be combined with a symbolic representation of the discrete part.

In this paper, we suggest an alternative BDD-based data-structure, *Numerical Decision Diagram* (NDD) that has a caonicity property: *given an ordering of the* *clock variables, every k-polyhedral set has a unique minimal representation.* For this data-structure we have boolean set-theoretic operations and equivalence testing for free.[5] We present an algorithm to calculate forward and backward projection in time for this data-structure and thus have all the ingredients needed in order to do reachability analysis for timed automata. Since this representation is BDD-based it can be combined naturally with symbolic methods for the discrete part of the system.

The rest of the paper is organized as follows. In section 2 we present timed automata and define the components of their reachability analysis algorithms. In section 3 we define NDD's and their forward projection algorithm for the *discrete-time* interpretation of timed automata. In section 4 we show how a discretization scheme, first reported in [GPV94], can be used to extend the scope of NDD's to the *dense-time* interpretation. Finally we present some experimental results.

2 Timed Automata

First, some notations. We use bold-face letters to denote points in $\mathbb{R}^d$. Thus, $\mathbf{v}$ stands for $(v_1, \ldots, v_d)$, where $v_i \in \mathbb{R}$, for every $i = 1, \ldots, d$. For points $\mathbf{u}, \mathbf{v} \in \mathbb{R}^d$, we write $\mathbf{u} \leq \mathbf{v}$ to denote that $u_i \leq v_i$, for every $i = 1, \ldots, d$. A subset $S \subseteq \mathbb{R}^d$ is called *monotonic* if $\mathbf{v} \in S$ implies $\mathbf{u} \in S$, for every $\mathbf{u} \in \mathbb{R}^d$ satisfying $\mathbf{u} \leq \mathbf{v}$.

For the sake of (the few) readers not familiar with timed automata we start with an informal illustration of the behavior of these creatures. Consider the timed automaton of figure 1. It has two states and two clocks z_1 and z_2. Suppose it starts operating in the configuration $(q_1, 0, 0)$ (the two last coordinates denote the values of the clocks). Then it can stay at q_1 as long as the staying condition for q_1 is true, namely $z_1 \leq 2$. Meanwhile the values of the clocks grow and the set of all configurations reachable from $(q_1, 0, 0)$ without leaving q_1 is $\{(q_1, t, t) : 0 \leq t \leq 2\}$. However, after one second, the condition $z_1 \geq 1$ (the guard of the transition from q_1 to q_2) is satisfied and the automaton can move to q_2 while setting z_2 to 0. Hence the additional reachable configurations are $\{(q_2, t, 0) : 1 \leq t \leq 2\}$. Having entered q_2 in one of these configurations, the automaton can either stay there as long as $z_1 \leq 5 \wedge z_2 \leq 3$ or can unconditionally move to $(q_1, 0, 0)$, etc.

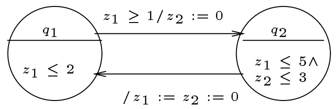

Fig. 1. A timed automaton.

[5] That is, for the same price as for BDD's in general.

Since the state-space of timed automata contains real values, we have an infinite-state automaton and an enumerative approach, where all states and transitions are enumerated, is impossible. We will use notation such as $G_{qq'}$ to denote the set of values in the clock space that satisfy the condition ("guard") for the transition from q to $q' \neq q$. Similarly, G_{qq} denotes the set of clock values allowing the automaton to stay in q ("staying conditions"). In timed automata such sets are restricted to be k-polyhedral subsets of $\mathbb{R}^d$, that is, the class of sets obtainable by applying set-theoretic operations to half-spaces of the form $\{\mathbf{v} : v_i \leq c\}$, $\{\mathbf{v} : v_i < c\}$, $\{\mathbf{v} : v_i - v_j \leq c\}$ or $\{\mathbf{v} : v_i - v_j < c\}$ for some integer $c \in \{0, \ldots, k\}$, where k is some positive integer.[6] These sets constitute the finite *region graph* [AD94] whose properties underlie all analysis methods for timed automata.

A function from $\mathbb{R}^d$ to itself is a *reset function* if it sets some of its arguments to 0 and leaves the others intact. We will use $R_{qq'}$ to denote the reset function associated with every pair of states (we take R_{qq} to be the identity function).

We will make the following simplifying assumptions concerning the timed automata that we consider: 1) There is only one transition associated with every pair of states. 2) The values of the clocks are bounded by k. Hence the clock space is $[0, k)^d$. 3) $G_{qq'}$ is convex for every $q, q' \in Q$, and 4) G_{qq} is monotonic for every $q \in Q$. The readers can convince themselves that it costs few states to convert any timed automaton into one satisfying these properties.

We let K denote the interval $[0, k)$ in the dense-time interpretation or the set $\{0, \ldots, k-1\}$ in the discrete-time interpretation. For every $\mathbf{z} \in \mathbb{R}^d$ we use $\mathbf{z} + t$ to denote $\mathbf{z} + t \cdot \mathbf{1}$ where $\mathbf{1} = (1, 1, \ldots, 1)$ is a d-dimensional unit vector.

Definition 1 (Timed Automaton). *A timed automaton is $\mathcal{A} = (Q, Z, \delta)$ such that, Q is a discrete set, $Z = K^d$ is the clock space ($Q \times Z$ is the configuration space) and $\delta : Q \times Z \mapsto 2^{Q \times Z}$ is the transition relation. It is required that δ admits the following decomposition: For every $q, q' \in Q$, let $G_{qq'} \subseteq Z$ be a k-polyhedral monotonic set and let $R_{qq'} : Z \mapsto Z$ be a reset function. Then, for every $(q, z) \in Q \times Z$*

$$\delta(q, z) \quad = \quad \left\{ (q', z') \ : \ \exists t \in K \begin{pmatrix} z + t \in G_{qq} \cap G_{qq'} \ \wedge \\ z' = R_{qq'}(z + t) \end{pmatrix} \right\} \qquad (1)$$

The meaning of $\delta(q, \mathbf{z})$ is the set of $Q \times Z$ configurations the automaton can reach starting at $(q, \mathbf{z})$ by waiting t time (possibly zero), and then taking *at most* one transition.[7]

Every subset of $Q \times Z$ encountered in the analysis of timed automata can be decomposed into a finite union of sets of the form $\{q\} \times P$ where P is k-polyhedral. We will write such sets as (q, P). We will extend functions on ele-

[6] In fact, we can use $c \in \{0, r, 2r \ldots, kr\}$ for some positive rational r.

[7] In the treatment here, we assume that all sets of the form G_{qq} are definable by a positive boolean combination of inequalities of the form $x_i \leq c_i$ and $x_i - x_j \leq c_{ij}$. All the techniques presented here can be generalized to apply to the more general case that some of the inequalities defining G_{qq} are strict.

ments to functions on sets in the natural way, e.g. $\delta(q,P) = \bigcup_{\mathbf{z}\in P} \delta(q,\mathbf{z})$ and $R_{q,q'}(P) = \bigcup_{\mathbf{z}\in P} R_{qq'}(\mathbf{z})$.

Next, we define a function $\Phi : 2^Z \mapsto 2^Z$ (time forward projection) as:

$$\Phi(P) = \{\mathbf{z} + t : \mathbf{z} \in P, t \in K\} \cap Z.$$

It is not hard to see that the immediate successors of a set of configuration (q,P) can be written as

$$\delta((q,P)) = (q,\widehat{P}) \cup \bigcup_{q'\neq q} (q', P_{q'})$$

where $\widehat{P} = \Phi(P) \cap G_{qq}$ and for every q', $P_{q'} = R_{qq'}(\widehat{P} \cap G_{qq'})$. This concludes the motivation for the paper as we see that the additional machinery needed to analyze timed automata consists of calculations of boolean operations, $R(P)$, and $\Phi(P)$ on k-polyhedral sets.

3 Numerical Decision Diagrams: Discrete Time

3.1 Representation

The idea of NDD's is elementary. When we consider K^d under the discrete interpretation, we have nothing but subsets of a finite set. Obviously, every element of K can be coded in binary using $b = \lceil \log k \rceil$ bits, where $\lceil \log k \rceil$ is the smallest integer not smaller than $\log k$. Consequently, we can represent every subset of K^d as a boolean function of $d \cdot b$ boolean variables. This function can be represented by a BDD in the usual way. We will use standard positional encoding, i.e., every number $n \in K$ is represented by a set of values $x_0, \ldots x_{b-1}$ such that $n = \sum_{i=0}^{b-1} x_i \cdot 2^i$.

The first question concerning the implementation is the ordering of the bits of every number. Although, especially for sets of the form $x \leq c$, putting the most significant bit first might lead to smaller BDD's, we prefer to put the least significant bit first, because it facilitates the calculation of Φ. Examples of sets and their NDD representation for $d = 1$ and $k = 8$ appear in figure 2. When there are more than one clock variables, there are various ways to order their bits, for example, $x_0, x_1, \ldots, y_0, y_1, \ldots$.

In order to represent decision trees and BDD's textually we will use the expression $bdd(x_i, L, R)$ to denote a tree that tests x_i, branches to the subtree L on zero and to the subtree R on one. For example, the tree obtained from the BDD for $x < 5$ in figure 2 is written as:

$$\begin{array}{llll} bdd(x_0, & bdd(x_1, & 1, & bdd(x_2,1,0)), \\ & bdd(x_1, & bdd(x_2,1,0), & 0 \quad)) \end{array}$$

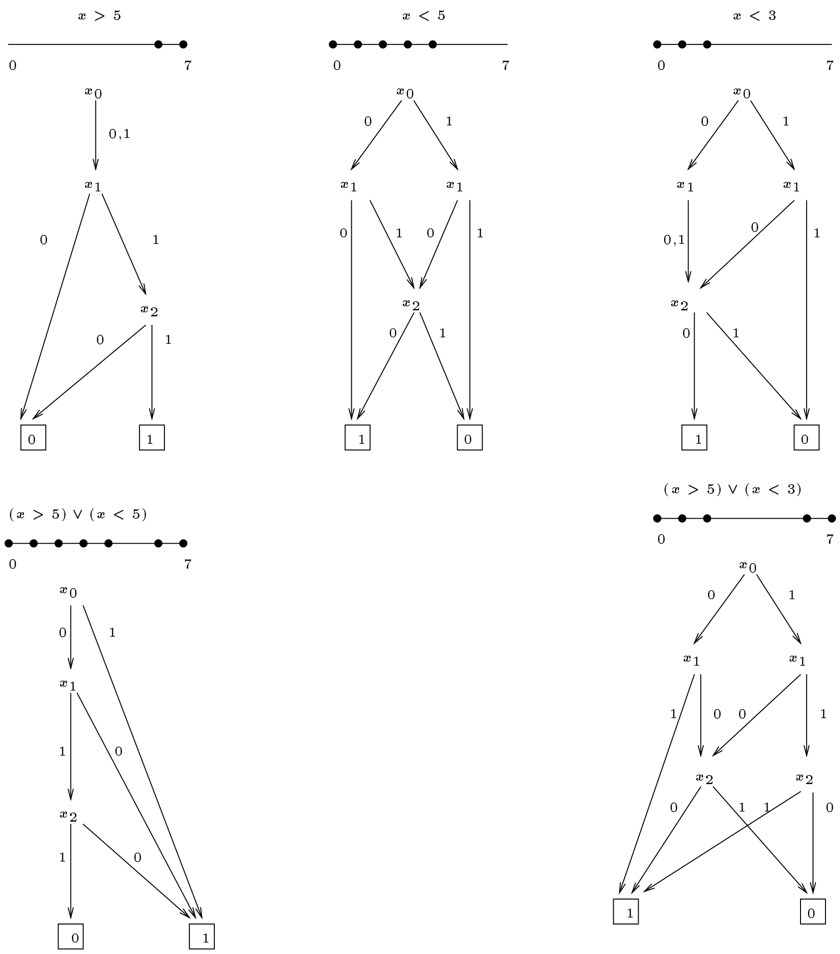

Fig. 2. Some 8-polyhedral sets in one-dimension and their corresponding NDD's.

3.2 Operations

Beside set-theoretic operations that we have for free, the reset operation is also elementary: in order to calculate $R(P)$ for a reset function R that resets, say, the variable x, you build a BDD for the set $x = 0$ and intersect it with the BDD for $\exists x\, P$. What remains to show is how to calculate $\Phi(P)$, which we will first demonstrate on the semantic level.

Given $P \in K^d$, $\Phi(P)$ can be written as $\{\mathbf{z} : \exists t \in K \; s.t. \; \mathbf{z} - t \in P\}$. Before applying the existential quantifier we have a set $P' \in K^{d+1}$ representing all the tuples $(t, \mathbf{z})$ such that $\mathbf{z} - t \in P$. We will present a procedure that converts a $b \cdot d$-variable NDD for P into a $b \cdot (d+1)$-variable NDD for P' (with t as an additional K-variable, encoded using the boolean variables $t_0, t_1, \ldots, t_{b-1}$).

Eliminating the existential quantifier for t from P', we obtain the BDD for $\Phi(P)$. The procedure will initially create the NDD for the set $P_1 = \{(t, \mathbf{z}) : \mathbf{z} \ominus t \in P\}$ where $\ominus$ stands for subtraction modulo k. Then, by intersecting P_1 with the set $P_2 = \{(t, \mathbf{z}) : \mathbf{z} \geq t \cdot \mathbf{1}\}$ we get P' (see figure 3).

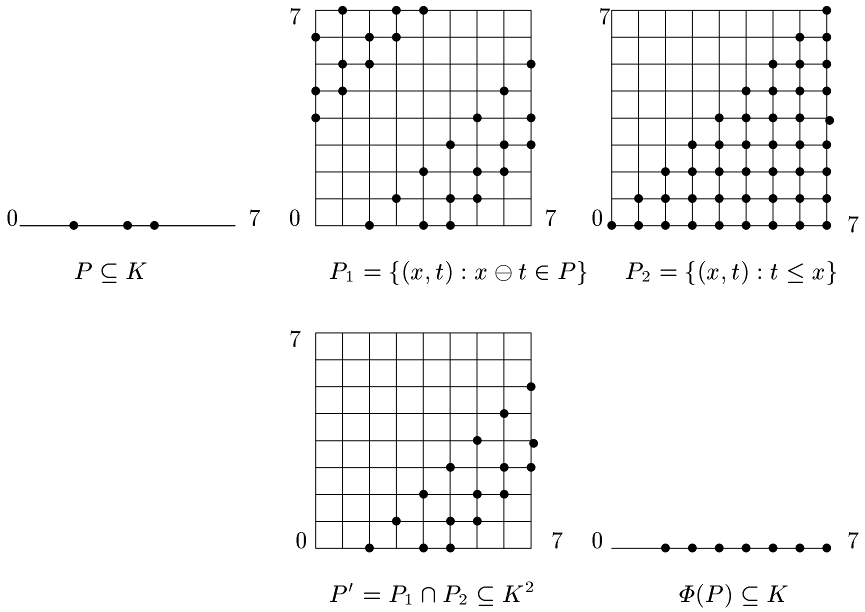

Fig. 3. Calculating $\Phi(P)$ via moving to $d+1$ dimensions (P') and then projecting away the time. We first make the subtraction modulo k (creating P_1) and then intersect with P_2 to get rid from overflow.

To illustrate the construction of P_1, we consider first the case that $d = 1$, i.e. only one clock. The recursive function $sub(B, borrow)$ presented in table 1 takes an NDD B for $P \subseteq K$ and produces an NDD for $P_1 \subseteq K^2$ as described above. The parameter $borrow$ represents the "borrow" bit which is propagated from right to left on performing binary subtraction. The external invocation of this function is done with $borrow = 0$. For simplicity of presentation, we assume that B has nodes for all variables, with entries of the form $bdd(x_i, L, L)$ in case the function is independent of x_i. An optimized version can be derived for the more general case of skipped variable tests.[8]

The effect of applying the function to an arbitrary decision tree over $\{0, \ldots, 3\}$ is depicted in figure 4. The extension to $d > 1$ is rather straightforward.

[8] As usual in BDD applications, all calls are hashed so that repeated calls with the same arguments will not repeatedly traverse the complete subtrees.

```
function sub(B, borrow)
begin
if B is a leaf
  then return(B);
  else let B = bdd(x_i, L, R)
       if borrow = 0
         then return(bdd(t_i,bdd(x_i, (sub(L, 0), sub(R, 0))),
                              bdd(x_i, (sub(R, 1), sub(L, 0)))));
         else  return(bdd(t_i, bdd(x_i, (sub(R, 1), sub(L, 0)))
                              bdd(x_i, (sub(L, 1), sub(R, 1)))));
end
```

Table 1. The function *sub*.

4 Dense Time

The above construction is sufficient for analyzing timed automata under the discrete-time interpretation. It is however known that some timed automata can produce behaviors (state-sequences) under a dense semantics which are not possible under any discrete-time semantics. In this section we introduce a discretization scheme [GPV94][9] having the two following important properties: 1) It preserves the qualitative behavior of the automaton, that is, for every sequence of discrete transitions in the semantics of a timed automaton $\mathcal{A}$, there is a similar sequence in the semantics of its discretization $\widetilde{\mathcal{A}}$ and vice versa. 2) It is amenable to representation by NDD's.

For each clock value z_i, $i = 1, \ldots, d$, let I_i and f_i denote the integer and fractional parts of z_i, respectively. Two clock valuations $\mathbf{z} = (I_1 + f_1, \ldots, I_d + f_d)$ and $\mathbf{z}' = (I_1' + f_1', \ldots, I_d' + f_d')$ are defined to be *region equivalent*, written $\mathbf{z} \sim \mathbf{z}'$, if

$$\bigwedge_{i=1}^{d} \Big((I_i = I_i') \wedge (f_i > 0 \leftrightarrow f_i' > 0) \Big) \quad \wedge \quad \bigwedge_{i,j \in \{1,\ldots,d\}} (f_i > f_j \leftrightarrow f_i' > f_j').$$

We consider automata with $Z = K^d$. We will use a discretization step $\Delta = 1/(2d)$ and let $\widetilde{K} = \{n\Delta : 0 \leq n < 2kd\}$. In other words, we cut every unit interval into $2d$ equal segments and pick the endpoints. The discretized clock space (that is, the domain over which discretized clocks range) is

$$\widetilde{Z} = \widetilde{K}^d \cap \{(z_1, \ldots, z_d) : \forall i, j\, |z_i - z_j| = 2m\Delta\}.$$

Note that we take from $\widetilde{K}^d$ only points such that the difference between any pair of clock valuations is an *even* multiple of Δ (see figure 5). For any polyhedral

[9] Discovered independently by the authors but a year later.

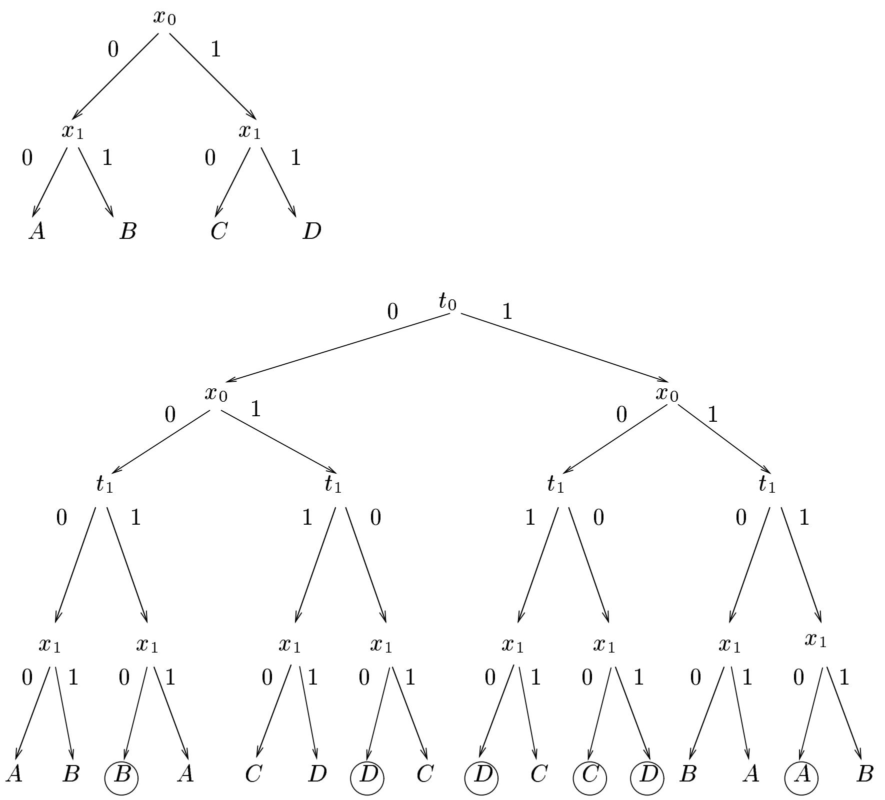

Fig. 4. Applying *sub* to an arbitrary decision tree over $\{0, 1, 2, 3\}$. The circled leaves will become zero when we intersect with $\{(x, t) : x \geq t\}$.

set P, we let its discretization be $\widetilde{P} = P \cap \widetilde{Z}$. It is not hard to see that, for every k-polyhedral set P, we have $P \neq \emptyset$ iff $\widetilde{P} \neq \emptyset$. Another important property of this scheme is the following:

Claim 1. *Let $\boldsymbol{z} = \widetilde{\boldsymbol{z}} + \varepsilon$ for some $\widetilde{\boldsymbol{z}} \in \widetilde{Z}$, $|\varepsilon| < \Delta$. Then*

$$\boldsymbol{z} \in P \Rightarrow (\widetilde{\boldsymbol{z}} \in P \vee \widetilde{\boldsymbol{z}} + \Delta \in P)$$

(and hence at least one of them belongs also to $\widetilde{P}$).

Proof: If $\widetilde{\mathbf{z}} = (z_1, \ldots, z_d) \in P$ we are done, otherwise there is one or more inequalities of the form $z_i > c_i$ satisfied by $\widetilde{\mathbf{z}} + \varepsilon$ but not by $\widetilde{\mathbf{z}}$ (which implies that $z_i = 2dn\Delta$ for some integer n). These inequalities must be satisfied by $\widetilde{\mathbf{z}} + \Delta$ as well. On the other hand, if there is an inequality of the form $z_j < c_j$ satisfied by $\widetilde{\mathbf{z}} + \varepsilon$ but not by $\widetilde{\mathbf{z}} + \Delta$, we have $z_j = (2dn - 1)\Delta$, which contradicts the

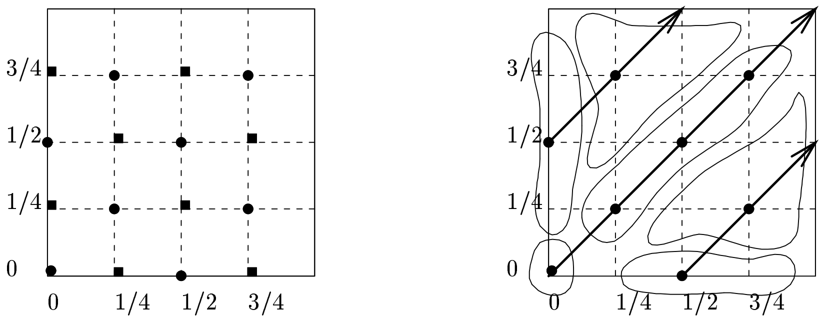

Fig. 5. Left: discretizing $[0,1)^2$: the circled points are the elements of $\widetilde{Z}$ while the squared points belong to $\widetilde{K}^d - \widetilde{Z}$. Right: illustration of claims 1 and 2.

assumption $\widetilde{\mathbf{z}} \in \widetilde{Z}$. In addition, $\widetilde{\mathbf{z}} + \varepsilon$ and $\widetilde{\mathbf{z}} + \Delta$ satisfy together every diagonal inequality (of the form $z_i - z_j < c$) and we can conclude that $\widetilde{\mathbf{z}} + \Delta \in P$. ▟
Note that this fails to be true for points outside $\widetilde{Z}$. Consider $\mathbf{z} = (0, 3/4)$ and $P = (0 < x < 1) \wedge (0 < y < 1) \wedge (y > x)$. Here $\mathbf{z} + \varepsilon \in P$ but neither $\mathbf{z}$ nor $\mathbf{z} + \Delta = (1/4, 1)$ belong to P.

The discretized forward projection $\widetilde{\Phi} : 2^{\widetilde{Z}} \to 2^{\widetilde{Z}}$ is the restriction of Φ to points in $\widetilde{Z}$ and time values in $\widetilde{K}$:

$$\widetilde{\Phi}(\widetilde{P}) = \{\widetilde{\mathbf{z}}' \in \widetilde{Z} : \exists \widetilde{\mathbf{z}} \in \widetilde{P} \, \exists \widetilde{t} \in \widetilde{K} \;\; s.t. \; \widetilde{\mathbf{z}}' = \widetilde{\mathbf{z}} + \widetilde{t}\}.$$

Claim 2 Discretization Preserves Forward Projection. *For every k-polyhedral set P and P' such that $P' = \Phi(P)$*

$$\widetilde{\Phi}(\widetilde{P}) = \widetilde{P'}.$$

Proof: One direction, $\widetilde{\Phi}(\widetilde{P}) \subseteq \widetilde{P'}$ is obvious because $\widetilde{P} \subseteq P$ and $\widetilde{K} \subseteq K$. For the other direction, suppose some $\widetilde{\mathbf{z}}' \in \Phi(P) \cap \widetilde{Z}$, implying that $\widetilde{\mathbf{z}}'$ can be written as $(n_1\Delta, \ldots, n_d\Delta)$, and that for some $\mathbf{z} \in Z$, $t \in K$, $\mathbf{z} + t = \widetilde{\mathbf{z}}'$ hence $\mathbf{z} = (n_1\Delta - t, \ldots, n_d\Delta - t)$. Let $t = m\Delta + t'$ for some $t' < \Delta$. Then $\mathbf{z} = ((n_1 - m)\Delta + t', \ldots, (n_d - m)\Delta + t')$. According to the previous claim either $\widetilde{\mathbf{z}}$ or $\widetilde{\mathbf{z}} + \Delta$ is in $\widetilde{P}$ and their temporal successor $\widetilde{\mathbf{z}}'$ is in $\widetilde{\Phi}(\widetilde{P})$. ▟

Having shown that forward projection (as well as boolean operations) on Z and K can be imitated by discretized operations on $\widetilde{Z}$ and $\widetilde{K}$, the only remaining problem is concerned with the reset operator. The problem is that $\widetilde{Z}$ is not closed under reset functions – for example, resetting the first coordinate of $(\Delta, \Delta) \in \widetilde{Z}$ we obtain $(0, \Delta) \in \widetilde{K}^d - \widetilde{Z}$ (because the difference between the points is not an even multiple of Δ). This is important because claim 2 does not hold on $\widetilde{K}^d$ but only on $\widetilde{Z}$. In order to calculate successors on the discretization we need an "adjustment" operator, which, after applying a reset, will delete points that went out of $\widetilde{Z}$ and replace each of them by one or more region-equivalent points in $\widetilde{Z}$. This extra operator can be viewed as the price we pay for dense reasoning.

For each $m \in \{0, \ldots, d-1\}$, let us define a function $\alpha_m : \widetilde{K}^d \to 2^{\widetilde{Z}}$ as follows:

$$\alpha_m(\mathbf{z}) = \{\mathbf{z}' : \bigwedge_{i=1}^{d} \left((I_i = I'_i) \wedge \begin{pmatrix} f_i = 0 \wedge f'_i = 0 \\ \vee \\ f_i = (2l+1)\Delta \wedge l < m \wedge f'_i = (2l+2)\Delta \\ \vee \\ f_i = (2l+1)\Delta \wedge l > m \wedge f'_i = (2l)\Delta \end{pmatrix} \right)\}$$

The function α_m returns a non-empty set if and only if all the non-zero fractional parts of $\mathbf{z}$ are odd multiples of Δ, and none of them falls in the interval $[2m\Delta, (2m+2)\Delta]$. Its effect is to add Δ to all fractional parts f_i satisfying $0 < f_i < 2m\Delta$ and subtract Δ from all fractional parts satisfying $(2m+2)\Delta < f_i$. Zero fractional parts are left unchanged. One can see that if $\mathbf{z}$ satisfies these conditions then $\alpha_m(\mathbf{z}) = \{\mathbf{z}'\}$ and $\mathbf{z}' \sim \mathbf{z}$. This operator is illustrated in figure 6. Based on this family of functions we define $\alpha : \widetilde{K}^d \to 2^{\widetilde{Z}}$ as

$$\alpha(\mathbf{z}) = \begin{cases} \mathbf{z} & if\ \mathbf{z} \in \widetilde{Z} \\ \bigcup_{m=0}^{d-1} \alpha_m(\mathbf{z}) & otherwise \end{cases}$$

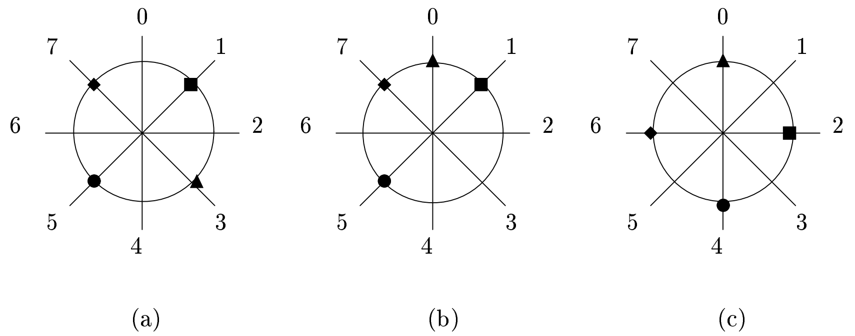

Fig. 6. The effect of the adjustment operator α_1 for $K = [0, 1)^4$ and $\Delta = 1/8$ (all points are multiples of Δ): (a) A point $\mathbf{z}_1 = (1, 3, 5, 7) \in \widetilde{Z}$. (b) After restting the second clock we obtain $\mathbf{z}_2 = (1, 0, 5, 7) \notin \widetilde{Z}$. (c) Applying α_1 we push the non-zero clocks toward the "hole" around 3 and obtain $\mathbf{z}_3 = (2, 0, 4, 6)$ which is region-equivalent to $\mathbf{z}_2$. Note that in this graphical representation the passage of time is via clock-wise rotation.

It is not hard to see that the application of α to any $P \subseteq \widetilde{K}^d$ yields a subset $P' \subseteq \widetilde{Z}$ such that, for every $\mathbf{z} \in P$, there is at least one $\mathbf{z}' \in P'$ satisfying $\mathbf{z} \sim \mathbf{z}'$. Based on this function, we can define for every reset function $R : Z \to Z$ a discretized reset function $\widetilde{R} : \widetilde{Z} \to 2^{\widetilde{Z}}$ as $\widetilde{R}(\widetilde{\mathbf{z}}) = \alpha(R(\widetilde{\mathbf{z}}))$. It follows that, for every

P, $\widetilde{R}(\widetilde{P})$ has elements of every region equivalence class which are represented in $R(P)$.

This is all we need: we just add to the NDD solution for integer time is $d \cdot \log(2d)$ bits to represent the finer grid and to replace every reset function $R_{qq'}$ by its adjusted version $\widetilde{R_{qq'}}$. The same arithmetical calculation of time successors described in section 3 will work, when we add the fractional bits to the clock and time variables. The adjustment operation seems to be the hard part of the calculation (we have only implemented the discrete-time representation so far) but at least this operation is performed only on the fractional bits (and hence is sensitive to the number of clocks but not to their ranges). In fact there is a trade-off between two discretization schemes (see [GPV94]), one with $\Delta = 1/(d+1)$, where resets behave normally but the evolution of time is distorted and loses some of its arithmetical content, and the other one we describe here, were time evolution remains arithmetic while resetting is more involved.

5 Concluding Remarks

5.1 Related Work

Various tools for the analysis of timed and hybrid automata have been developed recently, e.g. KRONOS [DOY94], UPPAL [BGK$^+$] and Hy-Tech [AHH93]. The first two represent polyhedral sets by DBMs. An alternative approaches is to transform the timed automaton into a huge discrete automaton (the region graph) and than encode its using boolean variables and BDD's.[10]

The idea of extending BDD's for the purpose of solving arithmetical constraints has been proposed by Rauzy [Rau95]. The structures he proposes are, however, not canonical. Our method can be applied as well outside the analysis of timed automata, e.g., as a decision procedure for some decidable theories in bounded arithmetics (see also [WB95]). In fact, the forward projection calculation can be easily adapted to clocks having non-uniform rates in $\{0, -1, +1\}$ and can be applied to the analysis of larger classes of hybrid systems and to programs with bounded integer variables.

5.2 Experimental Results

For experimentation we have used a system developed at VERIMAG for representing and manipulating communicating automata augmeneted with bounded variables [BFK96]. This system takes such automata and translates them into BDDs using one of several publicly-available BDD packages – we have used the CUDD package [S95] of Colorado University. We have incorporated a discrete-time version of the NDD representation into that system and tested its performance on various timed automata corresponding to digital circuits with delays

[10] In [CC95] this approach has been applied to the degenerate case of one-clock automata.

(the exact definitions and the translation procedure from circuits to timed automata are described in [MP95]). We will report here the results obtained with two generic families of automata, for which we tried to calculate all the reachable configurations starting from an initial state.

The first family consists of one-state automaton having n clocks and n transitions. The automaton (see figure 7-a) can let time progress as long as none of the clocks has reached its upper bound u_i. Whenever a clock C_i reaches the lower bound $l_i < u_i$, a self-looping transition which resets C_i can be taken. These automata allow us to isolate the complexity of representing and manipulating polyhedral sets from that of treating the discrete state-space.

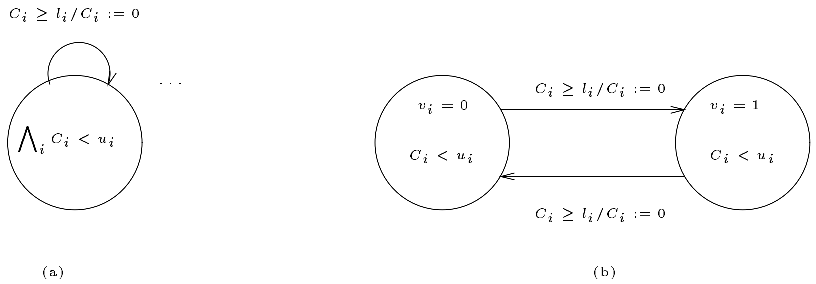

Fig. 7. The automata used for experiments: (a) A one-state automaton with n clocks and n transitions. (b) A basic two-state automaton with one clock.

The second family consists of a product of n two-state automata of the type appearing in (figure 7-b). Such a product is the natural way to model n independent non-deterministic input oscillators and it is a necessary ingredient in any attempt to do exhaustive timing analysis of asynchronous circuits.

The main difference between the two examples is that in the second we have 2^n discrete states. While the set of reachable configuration for this example will be of the form $\{(q, P_1)\}_{q \in \{0,1\}^n}$, the set of reachable clock configurations of the first automaton will be $\bigcup_q P_q$, which seems in general to be simpler to represent. In fact, for the constants we have chosen all the clock space is eventually reachable, but some very "hard" sets are encountered in the intermediate stages of the fixed-point calculations.

We have taken clock values in the range 0..15, let $l_i = 9$, $u_i = 12$ and compared the results with those of KRONOS which uses DBMs. For this set of examples the NDD results were much better. It should be noted, however, that DBMs implement the richer dense semantics and are *not sensitive to the range of clock values* as long as they do not cross the maximal integer value. In contrast, the performance of NDDs depends critically on the number of bits used to encode clock values. Moreover, the examples were chosen so that they generate intermediate polyhedral sets which are very "non-convex", which makes life hard for the DBM implementation (but also for NDDs). Finally the current forward simulation algorithm of KRONOS keeps all reachable regions in a form of a simulation graph[11] and this turns out to be inefficient for these examples – we believe that changing this implementation detail will allow KRONOS to treat larger examples. The results are summarized in table 2. They were obtained on a SUN Ultra-Sparc 1 with 256MB of memory.

no	*one state*				*many states*			
	DBM		*NDD*		*DBM*		*NDD*	
	time	*regions*	*time*	*max ndd*	*time*	*regions*	*time*	*max ndd*
2	0.0	17	0.1	51	0.1	35	0.2	107
3	0.4	175	0.3	235	1.5	783	2.1	837
4	53.9	2561	1.0	699	9:01.4	28974	9.9	3226
5	5:43:29.0	48499	2.7	1647	-	-	35.1	10617
6	-	-	5.9	3311	-	-	2:15.2	35640
7	-	-	10.9	5741	-	-	12:35.6	85863
8	-	-	18.8	10559	-	-	54:09.0	225057
9	-	-	29.3	19102	-	-	5:21:13.8	539092
10	-	-	48.6	29524	-	-	-	-
11	-	-	1:32.1	43947	-	-	-	-
12	-	-	2:58.8	62433	-	-	-	-
13	-	-	5:58.9	84696	-	-	-	-
14	-	-	13:09.3	114210	-	-	-	-
15	-	-	31:41.3	147016	-	-	-	-
16	-	-	1:23:38.6	190976	-	-	-	-
17	-	-	4:00:04.6	237484	-	-	-	-
18	-	-	11:59:32.1	299762	-	-	-	-

Table 2. Comparative results of the NDD and DBM implementations.

Our intermediate conclusion is that the analysis of timed automata with *many clocks* is not yet feasible. We have managed to handle additional non-trivial examples (such as an interconnected chain of XOR gates) with 10-13 clocks, but a closer investigation of polyhedral sets and their various representation schemes is needed in order to push performance limitations forward.

Acknowledgment: We thank S. Yovine, K. Daws and S. Tripakis for useful discussions concerning verification of timed automata in general and the KRONOS implementation in particular, and P. Raymond for some help in BDDs.

[11] This is the number appearing in the "regions" column in the table.

References

[AD94] R. Alur and D.L. Dill, A Theory of Timed Automata, *Theoretical Computer Science* 126, 183–235, 1994.

[AHH93] R. Alur, T.A. Henzinger, and P.-H. Ho. Automatic symbolic verification of embedded systems. In *IEEE Real-time Systems Symposium*, pages 2–11, 1993.

[BGK$^+$] J. Bengtsson, W.O.D. Griffioen, K.J. Kristoffersen, K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi. Verification of an audio protocol with bus collision using UPPAAL. In R. Alur and T.A. Henzinger (Eds. *Proc. CAV'96*, LNCS 1102, 244–256, Springer, 1996.

[Bry86] R.E. Bryant, Graph-based Algorithms for Boolean Function Manipulation, *IEEE Trans. on Computers* C-35, 677-691, 1986.

[BCM$^+$93] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang, Symbolic Model-Checking: 10^{20} States and Beyond, *Proc. LICS'90*, Philadelphia, 1990.

[CC95] S.V. Campos and E.M. Clarke. Real-time symbolic model checking for discrete time models, in T. Rus and C. Rattray (Eds.), *AMAST Series in Computing: Theories and Experiences for Real-Time System Development*, World Scientific, 1995.

[DOY94] C. Daws, A. Olivero and S. Yovine, Verifying ET-LOTOS Programs with KRONOS, *Proc. FORTE'94*, Bern, 1994.

[Dil89] D.L. Dill, Timing Assumptions and Verification of Finite-State Concurrent Systems, in J. Sifakis (Ed.), *Automatic Verification Methods for Finite State Systems*, LNCS 407, Springer, 1989.

[BFK96] M. Bozga, J.-C. Fernandez, A. Kerbrat, A Symbolic μ-calculus Model Checker for Automata with Variables, Unpublished Manuscript, VERIMAG, 1996.

[GPV94] A. Göllü, A. Puri and P. Varaiya, Discretization of Timed Automata, *Proc. 33rd CDC*, 1994.

[HNSY94] T. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine, Symbolic Model-checking for Real-time Systems, *Information and Computation* 111, 193–244, 1994.

[MP95] O. Maler and A. Pnueli, Timing Analysis of Asynchronous Circuits using Timed Automata, in P.E. Camurati and H. Eveking (Eds.), *Proc. CHARME'95*, 189-205, LNCS 987, Springer, 1995.

[McM93] K.L. McMillan, *Symbolic Model-Checking: an Approach to the State-Explosion problem*, Kluwer, 1993.

[S95] F. Somenzi, CUDD: CU Decision Diagram Package, 1995.

[Rau95] A. Rauzy, Toupie = μ-calculus + constraints. In P. Wolper, editor, *Proc. CAV'95*, LNCS 939, 114–126, Springer, 1995.

[WB95] P. Wolper and B. Boigelot. An automata-theoretic approach to presburger arithmetic constraints. In *Proc. Static Analysis Symposium*, LNCS 983, 21–32, Springer, 1995.