Efficient Verification of Timed Automata with BDD-like Data-Structures*

Farn Wang
Institute of Information Science, Academia Sinica
Taipei, Taiwan 115, Republic of China
+886-2-27883799 ext. 1717; FAX +886-2-27824814; farn@iis.sinica.edu.tw
Tools available at: http://www.iis.sinica.edu.tw/~farn/red

Abstract

We examine the causes of inefficiencies of previous BDD-like data-structures for timed automata state-space representation and manipulation. We identify four issues, which can cause the inefficiencies: variable designs, normal form definitions, zone-containment relation, and normal form computation. We explore the four issues in details and propose to use CRD (Clock-Restriction Diagram) for timed automata state-space representation. Then instead of using the traditional approach of computing canonical form (i.e., unique normal form) for zones representing the same state-space, we propose the new technique of *magnitude derivation and downward redundancy elimination* to convert zones into a small range of "normalized" zones. To better understand the complexity of BDD-like data-structures with various techniques, we have carried out extensive experiments and report the performance of BDD-like data-structures w.r.t. various techniques, benchmarks, and other tools.

Keywords: data-structures, BDD, timed automata, verification, model-checking

1 Introduction

Data-structure is the groundwork for efficient algorithms, especially for high-complexity tasks like real-time system model-checking. Most modern model-checkers for real-time systems are built around symbolic manipulation procedures[13] of *zones*, which means a behaviorally equivalent convex state space of a timed automaton and is symbolically represented by a set of difference constraints between clock pairs. DBM (difference-bounded matrix)[10] is generally considered the most efficient data-structure in representing sets of zones. But a DBM can only represent a convex state-space and DBM-technology can incur inefficiency in representing concave state-spaces.

In the last several years, people have been trying to duplicate the success of BDD techniques in hardware verification for the verification of timed automata [1, 7, 16, 17, 18, 19, 23]. Fully symbolic verification technologies using BDD-like structures[4, 8] can be efficient in both space and time complexities with intensive data-sharing in the manipulation of state space representations. But so far, all BDD-like structures[1, 7, 16, 17, 18, 19, 23] have not performed as well as the popular DBM [10], which is a 2-dimensional matrix and nothing BDD-like.

After examining the previous BDD-like data-structures, we feel that the efficiencies for the representation and manipulation of state-spaces with BDD-like data-structures are very sensitive to the following four issues.

- *The design of the evaluation variables* that is, the domains and the semantics of the variables. Especially, we have identified the *representation fragmentation phenomenon* of CDD[7], which is caused by the semantics of CDD's variables, in subsection 5.1 and shown that the phenomenon can indeed affect the verification performance against several independently developed benchmarks.
- *The definition of normal forms.* A zone can be represented by more than one sets of constraints. To avoid the space-explosion caused by recording many zones representing the same state-space, traditionally people use the *canonical-form approach*, which records only a chosen *canonical form* (a unique normal form) zone to represent all zones representing a given convex state-space. We found out that the more constraints are used in representing a chosen zone, the more space-complexity are incurred and the less data-sharing is possible.

*The work is partially supported by NSC, Taiwan, ROC under grants NSC 90-2213-E-001-006, NSC 90-2213-E-001-035, and the by the Broadband network protocol verification project of Institute of Applied Science & Engineering Research, Academia Sinica, 2001.

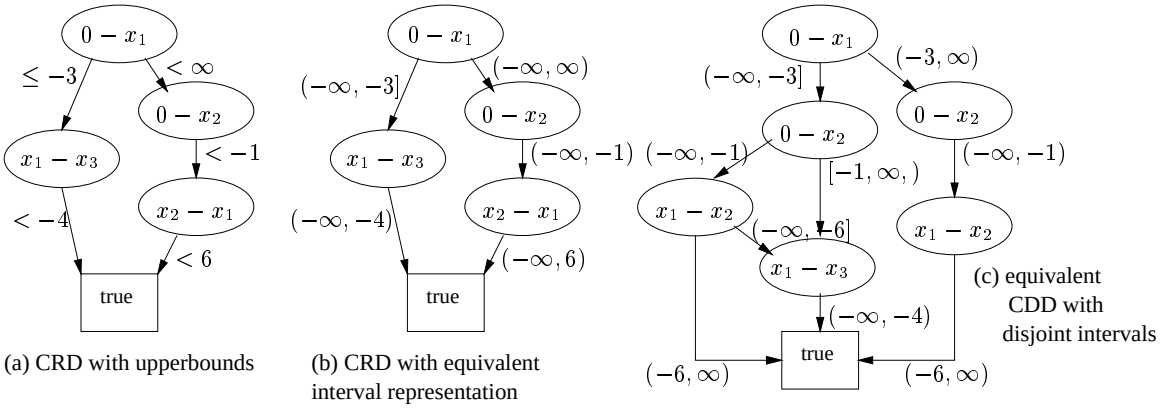


Figure 1: Differences between CRD and CDD

- *Zone-containment relation.* Zones may contain one another. If we use too small a constraint set to represent a zone, then we may not be able to efficiently decide the containment relations between distinct zones represented by different paths in a CRD. Unfortunately, this requirement for efficiency contradicts that for the last issue. Thus a proper balance needs to be found between representation efficiency and zone-containment-detecion efficiency. We believe this issue will also affect the performance of many BDD-like data-structures for dense-time state-space. For example, in CDD, it may also happen that some paths represent zones contained by others. But the effect on performance by eliminating contained zones was not discussed in [7].
- *Efficient calculation of normal forms.* In general it requires great time and space-complexities, with BDD-like data-structures, to compute the various canonical forms defined with tight bounds of clock differences. Moreover, the canonical-form approach does not support efficient implementation of specific functionalities, like elimination of empty zones.

Without a proper treatment of these four issues, we believe it is not possible to fully take advantage of the data-sharing capability of BDD-like data-structures. Straightforward adaptation from solutions for DBM, e.g. all-pair shortest-path canonical form, may result in low efficiency.

We have carried out numerous experiments with other tools (e.g. UPPAAL and Kronos) and various canonical forms with some possible normal-form computation algorithms in order to gain better understanding of the issues. We believe the observation we have made in the experiments will be very useful for the integration of current technologies toward the construction of an efficient and practical model-checkers for timed automata.

With a better understanding of the issues, we show how to use our new data-structure: *CRD (Clock-Restriction Diagram)*[20, 21] for better efficiency. CRD shares the same shape as CDD[7] with the major difference that variable values in CDD are disjoint intervals while variable values in CRD are upperbounds, which are structurally overlapping. For example, the CRD for the union of two zones: $\{0-x_1 \leq -3, x_1-x_3 < -4, x_2-x_1 < 6\}$ and $\{0-x_2 < -1, x_2-x_1 < 6\}\}$ (constraints of the form $x - x' < \infty$ are omitted) is in figure 1(a). If we change the upperbounds to interval reprsentations, we get the structure in figure 1(b), which is very like CDD but still different in that the interval labels from the root are not disjoint. The equivalent CDD for the same state-space, in figure 1(c), has both greater depth and greater width than figures 1(a) and (b). In section 5, we shall illustrate with examples to show why this subtle differences may incur significant performance differences in the manipulation and representation of state-spaces. Note the CDD in figure 1(c) adheres to the CDD restriction that only variables $x_i - x_j$ with $i < j$ are used.

In view of the vast time and space-complexities needed to compute various canonical forms with BDD-like data-structures, we switch to a drastic approach with two new techniques (Magnitude Derivation & Downward Redundancy Elimination, MD&DRE), which together only try to "normalize" zone representations by converting zones into a small range of *normal-form* zones representing the same state-space. In average, the two techniques work well with CRD and seem a good trade-off between data-structure conciseness and representation-manipulation efficiency. Another advantage of the MD&DRE techniques is that, in average, they support efficient elimination of inconsistent zones without having to go through the tight-bound derivations with all-pair shortest-path computation.

Finally, for BDD-like data-structures, evaluation ordering of variables are never too crucial to emphasize for efficiency. In this work, we also compare three evaluation ordering schemes and report their performances.

Compared with my previous work[20, 21], this manuscript reports the following new contributions. In [20, 21], CRD was defined and the effect of normal forms on CRD depths and data-sharing was preliminarily reported. Also

a new normal form called Cascade form for better detection of zone-containment relationship was proposed with construction algorithms. In this work, we look into the various issues in more detail and carry out experiments to justify our arguments. For example, we identify the *representation fragmentation phenomenon* of CDD[7] in subsection 5.1 and showed that it indeed blowed up the memory consumption in many benchmarks. Then we also developed new techniques (MD&DRE) to further improve the performance of our tool. In the end, we argue that BDD-like data-structures can be at least as competitive as DBM in the verification of dense-time systems.

In sections 2 and 3, we shall first define the basic concepts of timed automata verification and zones. In section 4, we present our CRD[20, 21] and its basic manipulations. Specifically, subsections 4.2 and 4.4 direct to symbolic algorithms in the appendices for basic functions like intersection and weakest precondtion calculation. In section 5, we shall examine the four causes for inefficiency of previous data-structures. Especially subsection 5.2 also serves as a short survey to compare with previous data-structures. In section 6, we introduce the techniques of MD&DRE. In section 7, we discuss some possibilities in evaluation orderings. In section 8, we report our implementations. In section 9, we report our experiments to compare

- techniques MD&DRE with various normal-form approaches and also with other tools;
- the performance difference with and without contained zone elimination;
- CRD's performance w.r.t. input timing constant magnitudes; and
- CDD and CRD's representation complexity in several benchmarks.

2 Timed automata verification

We use the widely accepted model of *timed automata*[2]. We assume familiarity with this model and will not go into much detail due to the page-limit. A *timed automaton* is a finite-state automaton equipped with a finite set of clocks which can hold nonnegative real-values. At any moment, the timed automaton can stay in only one *mode* (or *control location*). In its operation, one of the transitions can be triggered when the corresponding triggering condition is satisfied. Upon being triggered, the automaton instantaneously transits from one mode to another and resets some clocks to zero. In between transitions, all clocks increase their readings at a uniform rate.

For convenience, given a set Q of modes and a set X of clocks, we use $B(Q, X)$ as the set of all Boolean combinations of inequalities of the forms $\texttt{mode} = q$ and $x - x' \sim c$, where $\texttt{mode}$ is a special auxiliary variable, $q \in Q$, $x, x' \in X \cup \{0\}$, "$\sim$" is one of $\leq, <, =, >, \geq$, and c is an integer constant.

Definition 1 **timed automata** A timed automaton A is given as a tuple $\langle X, Q, I, \mu, T, \tau, \pi \rangle$ with the following restrictions. X is the set of clocks. Q is the set of modes. $I \in B(Q, X)$ is the initial condition on clocks. $\mu : Q \mapsto B(\emptyset, X)$ defines the invariance condition of each mode. $T \subseteq Q \times Q$ is the set of transitions. $\tau : T \mapsto B(\emptyset, X)$ and $\pi : T \mapsto 2^X$ respectively defines the triggering condition and the clock set to reset of each transition. ||

A *valuation* of a set is a mapping from the set to another set. Given an $\eta \in B(Q, X)$ and a valuation ν of X, we say ν *satisfies* η, in symbols $\nu \models \eta$, iff it is the case that when the variables in η are interpreted according to ν, η will be evaluated *true*.

Definition 2 **states** A state ν of $A = \langle X, Q, I, \mu, T, \tau, \pi \rangle$ is a valuation of $X \cup \{\texttt{mode}\}$ such that

- $\nu(\texttt{mode}) \in Q$ is the mode of A in ν; and
- for each $x \in X$, $\nu(x) \in \mathcal{R}^+$ such that $\mathcal{R}^+$ is the set of nonnegative real numbers and $\nu \models \mu(\nu(\texttt{mode}))$. ||

For any $t \in \mathcal{R}^+$, $\nu + t$ is a state identical to ν except that for every clock $x \in X$, $\nu(x) + t = (\nu + t)(x)$. Given $\bar{X} \subseteq X$, $\nu\bar{X}$ is a new state identical to ν except that for every $x \in \bar{X}$, $\nu\bar{X}(x) = 0$.

Definition 3 **runs** Given a timed automaton $A = \langle X, Q, I, \mu, T, \tau, \pi \rangle$, a *run* is an infinite sequence of state-time pair $(\nu_0, t_0)(\nu_1, t_1) \ldots (\nu_k, t_k) \ldots\ldots$ such that $\nu_0 \models I$ and $t_0 t_1 \ldots t_k \ldots\ldots$ is a monotonically increasing real-number (time) divergent sequence, and for all $k \geq 0$,

- for all $t \in [0, t_{k+1} - t_k]$, $\nu_k + t \models \mu(\nu_k(\texttt{mode}))$; and
- either $\nu_k(\texttt{mode}) = \nu_{k+1}(\texttt{mode})$ and $\nu_k + (t_{k+1} - t_k) = \nu_{k+1}$; or
  - $(\nu_k(\texttt{mode}), \nu_{k+1}(\texttt{mode})) \in T$ and
  - $\nu_k + (t_{k+1} - t_k) \models \tau(\nu_k(\texttt{mode}), \nu_{k+1}(\texttt{mode}))$ and
  - $(\nu_k + (t_{k+1} - t_k))\pi(\nu_k(\texttt{mode}), \nu_{k+1}(\texttt{mode})) = \nu_{k+1}$. ||

We can define the TCTL model-checking problem of timed automata as our verification framework. Due to page-limit, we here adopt the safety-analysis problem as our verification framework for simplicity. A safety analysis problem instance, $\mathrm{SA}(A, \eta)$ in notations, consists of a timed automata A and a safety state-predicate $\eta \in B(Q, X)$. A is *safe* w.r.t. to η, in symbols $A \models \eta$, iff for all runs $(\nu_0, t_0)(\nu_1, t_1) \ldots (\nu_k, t_k) \ldots\ldots$, for all $k \geq 0$, and for all $t \in [0, t_{k+1} - t_k]$, $\nu_k + t \models \eta$, i.e., the safety requirement is guaranteed.

3 Zones, closure form, and reduced form

A zone is symbolically represented by a set of difference constraints between clock pairs and means a behaviorally equivalent state subspace of a timed automaton. For convenience, let $\mathcal{Z}$ be the set of integers. Given $c \geq 0$ and $c \in \mathcal{Z}$, let $\mathcal{I}_c$ be $\{\infty\} \cup \{d \mid d \in \mathcal{Z}; -c \leq d \leq c\}$. Also for any $d \in \mathcal{Z}$, $d + \infty = \infty + d = \infty$.

Given an SA(A, η) with biggest timing constant C_A used in A and η, a zone is a set of constraints like $x - x' \sim d$, with $x, x' \in X \cup \{0\}$, $\sim \in \{\text{“}\leq\text{”}, \text{“}<\text{”}\}$, and $d \in \mathcal{I}_{C_A}$, such that when $d = \infty$, $\sim$ must be “$<$”. For convenience, let $\mathcal{B}_c = \{(\sim, d) \mid \sim \in \{\text{“}\leq\text{”}, \text{“}<\text{”}\}; d \in \mathcal{I}_c; d = \infty \Rightarrow \sim = \text{“}<\text{”}\}$. With respect to given X and C_A, the set of all zones is finite. Formally, a zone ζ can be defined as a mapping $(X \cup \{0\})^2 \mapsto \mathcal{B}_{C_A}$. Alternatively, we may also represent a zone ζ as the set $\{x - x' \sim d \mid \zeta(x, x') = (\sim, d)\}$. We shall use the two equivalent notations flexibly as we see fit.

There can be many zones representing the same convex subspace. A straightforward *canonical* representation of a zone-characterizable convex subspace is its zone in *closure form* (called shortest-path closure in [15]). A zone ζ is in closure form if and only if all its constraint bounds are tight, i.e., for any sequence of elements $x_1, \ldots, x_k \in X \cup \{0\}$, with $x_1 - x_k \sim d \in \zeta$ and $\forall 1 \leq i < k (x_i - x_{i+1} \sim_i d_i \in \zeta)$, either $d < \sum_{1 \leq i < k} d_i$ or $(d = \sum_{1 \leq i < k} d_i \wedge (\sim = \text{“}\leq\text{”} \Rightarrow \bigwedge_{1 \leq i < k} \sim_i = \text{“}\leq\text{”}))$. We can artificially designate the closure form of each zone as our canonical form of the corresponding state subspace characterized by the zone. For convenience, given a zone ζ, we let ζ^C be the notation for its closure form.

Another candidate for the canonical representation of zones is the *reduced form* (called shortest-path reduction in [15]) which records only minimum number of constraints for each zone according to some policy. We refer interested readers to [15, 20] for explanation how to convert a given zone ζ to its zone in reduced form, in symbols ζ^R, according to certain policy. It is shown in [15] that $\zeta^C = (\zeta^R)^C$; and DBM with zones in reduced form can be used as a canonical representation of timed automaton convex states-spaces and can significantly save space in verification.

4 Clock Restriction Diagram

4.1 Definitions

CRD[20, 21] is not a decision diagram for state space membership. Instead it is like a decision diagram for zone set membership. Each *evaluation variable* in a CRD is of the form $x - x'$, where x, x' are zeros or clocks, and the values of such variables range over $\mathcal{B}_{C_A}$, where C_A is the bound used in DBM for a given safety-analysis problem instance. Thus a value, say $(\leq, 5)$, of evaluation variable $x - x'$ describes the constraint of half-space $x - x' \leq 5$. A path from root to the only leaf node *true* in CRD represents a zone.

By fixing an evaluation ordering, we can construct a CRD just as BDD, CDD, or RED. In CRD, a missing constraint on the difference of a clock pair, say x, x', is interpreted as $x - x' < \infty$. Thus in the root node in figure 1(a), even no constraint is on $0 - x_1$ in zone of the right path, we still construct an arc with $0 - x_1 < \infty$ from the root node.

Given a set V of evaluation variables with $\mathit{true} \in V$, an *evaluation index* Ω over V is a 1-to-1 onto mapping from V to $\{0, 1, \ldots, |V| - 1\}$ such that $\Omega(\mathit{true}) = |V| - 1$. For convenience, for all $v, v' \in V$, we shall write $v \prec_\Omega v'$ iff $\Omega(v) < \Omega(v')$.

Definition 4 *Clock Restriction Diagram (CRD)* Given a set of variables $V = \{x - x' \mid x, x' \in X \cup \{0\}\} \cup \{\mathit{true}\}$, an evaluation index Ω over V, and a timing constant C_A, a CRD over V, Ω, and C_A is a tuple $D = (v, (\beta_1, D_1), \ldots, (\beta_n, D_n))$ with $n \geq 0$. In symbols, we write $\mathrm{var}(D) = v$. The restrictions are that $v \in V$ such that

- $v = \mathit{true}$ iff $n = 0$;
- if $v \neq \mathit{true}$, then for all $1 \leq i \leq n$, $\beta_i \in \mathcal{B}_{C_A}$ and D_i is a CRD over V, Ω, and C_A with $v \prec_\Omega \mathrm{var}(D_i)$;
- if $v \neq \mathit{true}$, then for all $1 \leq i < j \leq n$, $\beta_i \neq \beta_j$; and
- if $v \neq \mathit{true}$ and $n = 1$, then $\beta_1 \neq (<, \infty)$. ||

4.2 Basic set-oriented manipulations on CRD

For convenience of discussion, given a CRD, we may just represent it as the set of zones recorded in it. Definitions of set-union ($\cup$), set-intersection ($\cap$), and set-exclusion ($-$) of two zone sets respectively represented by two CRDs are straightforward. For example, given CRDs $D_1 : \{\zeta_1, \zeta_2\}$ and $D_2 : \{\zeta_2, \zeta_3\}$, $D_1 \cap D_2$ is the CRD for $\{\zeta_2\}$; $D_1 \cup D_2$ is for $\{\zeta_1, \zeta_2, \zeta_3\}$; and $D_1 - D_2$ is for $\{\zeta_1\}$. The complexities of the three manipulations are all $O(|D_1| \cdot |D_2|)$.

We need the following notation to conveniently define CRD manipulations which compare the strictness of elements in $\mathcal{B}_{C_A}$. Given $(\sim_1, d_1), (\sim_2, d_2) \in \mathcal{B}_{C_A}$, we say $(\sim_1, d_1)$ is *stricter* than $(\sim_2, d_2)$, in symbols $(\sim_1, d_1) \sqsubseteq (\sim_2, d_2)$, iff $d_1 < d_2 \vee (d_1 = d_2 \wedge (\sim_2 = \text{“}<\text{”} \Rightarrow \sim_1 = \text{“}<\text{”}))$. The following convenient notations are also adopted: $(\sim_1, d_1) \sqsubset (\sim_2$

WeakestPrecondition(D, q, q') {
  $D :=$ VariableEliminate($D \sqcap (\texttt{mode} = q') \sqcap \bigsqcap_{x \in \pi(q,q')} x = 0, \texttt{mode}$);
  for $x \in \pi(q, q')$, $D :=$ ClockEliminate(Bypass(D, x), x);
  $D :=$ Bypass($D \sqcap (\texttt{mode} = q) \sqcap \tau(q, q') \sqcap \mu(q), 0$);
  return D;
}

Table 1: calculation of timed weakest precondtion

$, d_2) \equiv (\sim_1, d_1) \sqsubseteq (\sim_2, d_2) \wedge (\sim_2, d_2) \not\sqsubseteq (\sim_1, d_1)$, $(\sim_1, d_1) \sqsupset (\sim_2, d_2) \equiv (\sim_2, d_2) \sqsubset (\sim_1, d_1)$, and $(\sim_1, d_1) \sqsupseteq (\sim_2, d_2) \equiv (\sim_2, d_2) \sqsubseteq (\sim_1, d_1)$.

Given two zones ζ_1 and ζ_2, $\zeta_1 \sqcap \zeta_2$ is a new zone representing the space-intersection of ζ_1 and ζ_2. Formally speaking, for every x, x', $\zeta_1 \sqcap \zeta_2(x, x') = \zeta_1(x, x')$ if $\zeta_1(x, x') \sqsubseteq \zeta_2(x, x')$; or $\zeta_2(x, x')$ otherwise. Space-intersection ($\sqcap$) of two CRDs D_1 and D_2, in symbols $D_1 \sqcap D_2$, is a new CRD for $\{\zeta_1 \sqcap \zeta_2 \mid \zeta_1 \in D_1; \zeta_2 \in D_2\}$. Our current algorithm of the manipulation has complexity $O(|D_1| \cdot |D_2|)$ and can be found in appendix A.

4.3 CRD+BDD

It is possible to combine CRD and BDD into one data-structure for fully symbolic manipulation. Since CRD only has one sink node: *true*, it is more compatible with BDD without FALSE terminal node which is more space-efficient than ordinary BDD. There are two things we need to take care of in this combination. The first is about the interpretation of default values of variables. In BDD, when we find a variable is missing during valuating variables along a path, the variable's value can be interpreted as either TRUE or FALSE. But in CRD, when we find a variable for constraint $x - x'$ is missing along a path, then the constraint is interpreted as $x - x' < \infty$.

The second is about the interpretation of CRD manipulations to BDD variables. Straightforwardly, "$\cup$" and "$\cap$" on Boolean variables are respectively interpreted as "$\vee$" and "$\wedge$" on Boolean variables. $D_1 - D_2$ on Boolean variables is interpreted as $D_1 \wedge \neg D_2$ when the root variable of either D_1 or D_2 is Boolean. For $D_1 \sqcap D_2$, the manipulation acts as "$\wedge$" when either of the root variables are Boolean. Due to page-limit, we shall omit the proof for the soundness of such interpretation.

From now on, we shall call it CRD+BDD a combination structure of CRD and BDD. As will be seen in section 7 and in our experiments, the evaluation ordering among variables in CRD+BDD may greatly affect the efficiency.

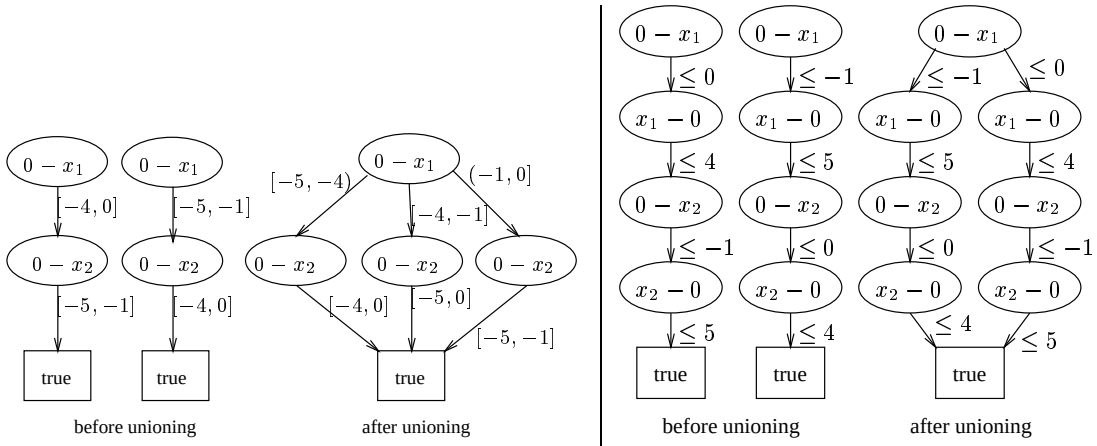
4.4 Weakest precondition computation of CRD

Our model-checking algorithm computes the reachable state-space fixpoint with iterative backward weakest precondition calculation. High-level description of the algorithm is very much like the one presented in [22] and will not be described in this manuscript. But we shall present a symbolic CRD manipulation algorithm for the computation of weakest precondition since, to our knowledge, in the literature no such algorithms for timed automata state-space representation in BDD-like data-structure has been presented.

To compute the weakest precondition before a transition rule $(q, q') \in T$, we use the procedure in table 1. Procedure VariableEliminate(D, w) returns a CRD identical to D except that variable w is removed. Procedure ClockEliminate(D, x) returns a CRD identical to D except that all clock difference variables with respect to clock x are removed. Procedure Bypass(D, x) returns a new CRD which represents the same state-space as D does. For each zone $\zeta \in D$ and for each two clocks $x_1, x_2 \in X$, Bypass(D, x) will add a constraint on variable $x_1 - x_2$ to ζ derived from the constraints on $x_1 - x$ and $x - x_2$ in ζ. Algorithms, taking advantage of data-sharing capability of CRD, for procedures VariableEliminate(), ClockEliminate(), and Bypass() can be found in appendix B, C, and D.

5 Issues on efficiencies of data-structures

For formal discussion, we introduce the following concepts about BDD-like data-structures. A *decision-path* in a BDD-like data-structure is a path from root to terminal (*false* or *true*) in the data-structure. The *depth* of a BDD-like data-structure is the length of the longest decision-path in the data-structure. The *width* of a BDD-like data-structure is the number of decision paths in the data-structure. The depth of a BDD-like data-structure represents the size of the largest convex constraint set it represents while the width represents the number of convex constraint sets it represents. The depth and width of a BDD-like data-structure can be used as a convenient metric to roughly measure how efficient a state-space is represented. In general, we would like to design



(a) unioning of CDDs

(b) unioning of CRDs

Figure 2: Comparison between variable semantics of CDD and CRD

- a BDD-like data-structures for zones to miminize both the depth and width of CRDs (or CDDs) at the same time to contain the representation complexity; and
- efficient maipulation routines to convert zones to their *normal forms*, to eliminate zones contained by other zones, and to eliminate empty zones.

5.1 Design of data-structures

In the design of BDD-like data-structures, two aspects need to be considered, i.e. the domain of variables and semantics of variable values. Since BDD-like data-structures exhibit exponential blowup w.r.t. the size of variable domain, in general it is good to keep the variable domain small. The semantics of variable values is about how we should interpret the values of variables and has a much subtle effect on space-complexity. We feel that it will help the readers understand this issue if we compare CRD with the previous data-structures. DBM-technology generally handles the complexity of timing constant magnitude very well. Since a DBM can only represent conjunctive relations and there is no data-sharing among DBMs, when the number of clocks increases, its performance may degrade rapidly.

NDD[1] uses binary encoding for clock readings and its performance is very sensitive to timing-constant magnitude.

DDD[16, 17] uses Boolean variables like $x - y \sim d$ to encode representations for dense-time state-spaces. The approach is very similar to Wang et al's work [23] and is likely to be inefficient since the size of variable domain is propotional to the timing constants and thus exponential to the input size.

RED[18, 19] encodes the ordering of fractional parts of clock readings in the variable ordering and has achieved very high space-efficiency for symmetric systems with large number of clocks and small timing constants. RED is indeed a canonical representation of timed automaton state subspaces. But for large timing constants, RED's performance degrades rapidly.

Finally, we want to compare our CRD with CDD[7], which is a decision diagram for state-space membership and has a very similar structure to CRD. The major difference between CRD and CDD is that the arcs from a node in CDD are labeled with "DISJOINT" intervals while those from a node in CRD are labeled with upperbounds, which are structurally overlapping. Due to this little difference, for some state-spaces, CDD may demand exponential size of memory. For example, we have the following state-space for n clocks:

$$\bigvee_{1\leq i\leq n} \bigwedge_{1\leq j\leq n}((i+j)\%n) \leq x_j \leq 2n + ((i+j)\%n) \tag{1}$$

Here "%" represents the modulo operator. When clock count n is 2, the compositions of the state-spaces in CDD and in CRD are in figure 2 (a) and (b) respectively. As can be seen, the CDD union operation will produce a CDD of three paths out of two zones while the CRD union operation will basically maintain the structures of the component zones. In fact, our experiment shows that the state-space of (1) exhibits an exponential blow up in CDD representation with respect to clock counts in the following table.

clock counts		2	3	4	5	6	7	9	11	13	15
CDD	node counts	4	12	31	73	162	346	1479	6064	24469	98166
	arc counts	6	23	78	238	663	1721	10056	52427	256674	1210285
CRD	node counts	7	16	29	46	67	92	154	232	326	436
	arc counts	8	18	32	50	72	98	162	242	338	450

Such exponential blowup, we believe, is due to the "DISJOINT" requirements of CDD on intervals. The requirement, although makes sense in mathematics, but actually contradicts the characteristics of zones which are noncanonical representations of convex state-spaces and may intersect with one another. Thus when union operation is performed, intervals will intersect each other into fragments. Such fragmentation phenomenon not only blows up the memory space requirement, but also destroys the manipulation results on zones. Such manipulation results can be generated from closure form computation or from zone-containment reduction. But since the union operation of CDD tends to restructure the zones, it may likely make previous effort in analysis of state-spaces in vain.

To justify our argument, we have also endeavored to implement some CDD manipulation procedures and carried out an experiment to observe the effect of the *representation fragmentation phenomenon* on representation complexity. The experiment is reported in subsection 9.3 and indeed confirms our arguments on the effect of the pheonomenon. Moreover, the experiment shows that the phenomenon not only blows up the representations of the just-mentioned toy example, but may also blow up the space-usage of independently developed benchmarks.

After the discussion of this subsection, we hope that the readers have been convinced that proper design of the data-structures is very crucial to the efficiency. However, we feel that previously people have not paid proper attention to this issue.

5.2 Definition of zone normal forms

Since many zones (or different decision paths) can represent the same convex state-space, without proper control, it is likely that BDD-like data-structures will end up with recording many zones for the same state-space. In previous research, people devised various algorithms to convert zones to their *canonical forms*. It is our observation that different canonical form definitions may incur drastically different representation efficiency and manipulation efficiency with BDD-like data-structures. In this and the following subsections, we shall use CRD to discuss the effect of various canonical forms on representation and manipulation efficiency.

The most popular canonical form is *closure form*[10, 7] (called shortest-path closure in [15]). A zone is in its *closure form* if all its clock difference constraints are tight. Such tight bounds can be computed with all-pair shortest-path algorithms. Unfortunately, the symbolic version of such algorithms with BDD-like data-structures are very expensive in both time and space-complexity. Closure form is the natural choice for DBM[10] and adopted in CDD[7] and DDD[16, 17]. We adopt the tradition that a CRD is in its closure form if all its zones are in closure form. Same tradition will be honored with other canonical forms.

The second choice is *reduced form* (called shortest-path reduction in [15]) in which a zone contains minimal number of clock difference constraints chosen by a policy. Note even the reduced form is not unique since we can choose many smallest constraint sets for the same state-space.

The detailed constraint information in closure form CRD not only incurs extra space-complexity but also severely harms the data-sharing capability of BDD-like data-structures. For example, we have the closure form and a possible reduced form CRDs for the following two zones.

$$\begin{array}{rlll} & (0 & -x_2 \leq -2 \wedge x_2 - & x_1 \leq 0 \quad \wedge x_1 - x_3 \leq -3 \wedge x_3 - 0 \leq 5) \\ \vee & (0 & - & x_1 \leq -2 \quad \wedge x_1 - x_3 \leq -3 \wedge x_3 - 0 \leq 5) \end{array} \tag{1}$$

The derived constraints $x_2 - x_3 \leq -3$ and $x_3 - x_2 \leq 3$ in the closure form prevents data-sharing. For detail information, we refer the readers to [21].

5.3 Efficient detection of zone-containment relation

Another problematic characteristic of zones is that they can contain one another and different decision-paths may represent non-disjoint zones. Without capability to efficiently detecting zone containment relation, it may happens that fixpoint algorithms will waste time and space in iterating through smaller and smaller state-spaces which are contained by zones already in the BDD-like data-structures. The traditional wisdom of *canonical forms* calculation does not handle this issue well.

To efficiently decide that one zone ζ_1 is a subspace of another zone ζ_2, we have to have the following *straightforward containment requirement (SCR)* on both ζ_1 and ζ_2:

$$\mathrm{SCR}(\zeta_1, \zeta_2)\text{: } \forall x, x'(\zeta_1(x, x') \sqsubseteq \zeta_2(x, x')).$$

If the containment relation between ζ_1 and ζ_2 cannot be decided with SCR, then in general we have to do an all-pair shortest-path computation, which is very expensive for BDD-like data-structures, to derive the tight bounds of all clock difference constraints.

On one hand, closure form CRD can efficiently support the detection of SCR since all its bounds are tight. But as described in subsection 5.2, closure form CRDs ususally introduce longest depth and may easily blow up the space-complexity. To get a balance in choosing canonical forms for representation efficiency (see subsection 5.2) and SCR detection efficiency, Wang proposed a new canonical form called *cascade form* CRD[21], which is very much like reduced form except that we add more constraints with respect to clocks with the same reading to assist the efficient detection of SCR. For detail information, we refer the readers to [21].

5.4 Efficient computation of the normal forms

It can be very expensive in both time and space (for huge intermediate CRDs only to be garbage-collected in the end) to compute closure form CRDs with its great depths of decision-paths and low data-sharing capability.

Since reduced form and cascade form are also defined on tight bounds of clock differences, a straightforward approach to compute them is to first compute their closure form counterpart explicitly. However, such an approach will nullify any performance advantage of reduced form and cascade form since resources are still consumed to compute their closure form counterpart. In [21], Wang proposed an algorithm to implicitly compute the all-pair shortest-path information with 4 auxiliary variables to avoid the high complexity associated with the great decision-path lengths in closure form CRDs. But in general, to compute all-pair shortest-path information either implicitly or explicitly is very expensive with BDD-like data-structures.

Another but not minor issue which has been overlooked in previous research is about the capability of normal forms to facilitate efficient elimination of empty zones. Intuitively, the earlier we can eliminate empty zones, the less burdensome the manipulation of BDD-like data-structures is. With the traditional canonical-form approach, such empty zones are eliminated in the course of deriving tight bounds with all-pair shortest-path information. No efficient solutions have been developed to enhance efficiency in this issue.

6 Magnitude derivation & downward redundancy elimination

Since canonical forms are expensive to compute, we decide to experiment with efficient techniques to normalize zones representing the same state-space to a small range of zones. If the "normalization algorithm" is efficient enough in both time and space complexity in average cases, then perhaps it is a good deal to trade a little increase in CRD widths for the reduction in total time and space consumptions. Normalized zones will be called as *normal zones*. There can be more than one normal zones for a convex state-space.

The first technique is called *magnitude derivation (MD)*. A clock difference constraint is called a *magnitude constraint* iff one of its clocks is 0. Technique MD only makes magnitude constraints tight in a zone. The technique is motivated by the observation that most timing constraints in timed automata are written with magnitude constraints. Traditional canonical-form approaches eliminate inconsistent zones while tightening the bounds. But with the MD technique, we expect a lot of the inconsistent zones will be eliminated when conjuncting with invariance conditions and triggering conditions written with magnitude constraints. The algorithm for magnitude derivation is presented in the following.

```
set of pairs of CRD+BDD        R, R'; /* two static set variables */
MD(D) {
  D' := false;
  while D' ≠ D, { D' := D; R := ∅; D := recMD(D); }
  return D;
}
clock          ZMID; /* one static clock variable */
element in B_{C_A}    ZB; /* one static upperbound variable */
recMD(D) {
  if D := true, then return D; else if ∃D', (D, D') ∈ R, then return D';
  D' := false;
  if D = (x − x', ((∼_1, c_1), D_1), ..., ((∼_n, c_n), D_n)),
    if x = 0 ∧ x' ≠ 0 {
      for i := n to 1, {
```

$D'_i := \texttt{recMD}(D_i)$;
ZMID $:= x'$; ZB $:= (\sim_i, c_i)$; $R' := \emptyset$; $D' := D' \cup (x - x' \sim_i c_i \sqcap \texttt{MDfromZero}(D'_i))$;
}
ZB $:= (<, \infty)$;
}
if $x \neq 0 \wedge x' = 0$ {
for $i := n$ to 1, {
$D'_i := \texttt{recMD}(D_i)$;
ZMID $:= x$; ZB $:= (\sim_i, c_i)$; $R' := \emptyset$; $D' := D' \cup (x - x' \sim_i c_i \sqcap \texttt{MDtoZero}(D'_i))$;
}
ZB $:= (<, \infty)$;
}
else for $i := n$ to 1, $D' := D' \cup (x - x' \sim_i c_i \sqcap \texttt{recMD}(D_i))$;
else /* $D = (v, ([l_1, u_1], D_1), \ldots, ([l_n, u_n], D_n))$, */
for $i := n$ to 1, $D' := D' \cup (l_i \leq v \leq u_i \sqcap \texttt{recMD}(D_i))$;
$R := R \cup \{(D, D')\}$; return D';
}

Procedure $\texttt{recMD}(D)$ iteratively calls procedures $\texttt{MDfromZero}(D'_i)$ and $\texttt{MDtoZero}(D'_i)$. At each calling of $\texttt{MDfromZero}(D'_i)$, the procedure will try to derive the tight bound on all constraints like $0 - x$ from two constraints on $0 -$ ZMID and ZMID $- x$ recorded in D'_i. Also each calling of $\texttt{MDtoZero}(D'_i)$, the procedure will try to derive the tight bound on all constraints like $x - 0$ from two constraints on $x -$ ZMID and ZMID $- 0$ recorded in D'_i.

Note that R is the database used to record the CRD nodes which have been processed. If a CRD node D is already processed before, then its recorded result can be returned without further processing. This capability to take advantage of data-sharing is very common in the manipulation algorithms of BDD-like data-structures.

The following basic routine is invoked in the execution of MDtoZero() and MDfromZero().

- ZoneLbAdd$((\sim_1, c_1), (\sim_2, c_2))$ {
if $c_1 + c_2 > C_A$, return $(<, \infty)$;
else if $c_1 + c_2 < -C_A \vee (c_1 + c_2 = -C_A \wedge (\sim_1 =$ '$<$'$\vee \sim_2 =$ '$<$'$))$, return $(<, -C_A)$;
else return $(\sim, c_1 + c_2)$ such that $\sim =$ '$<$' $\Leftrightarrow (\sim_1 =$ '$<$'$\vee \sim_2 =$ '$<$'$))$;
}

Again, in the following two procedures, we use R' to take advantage of the data-sharing feature of CRD.

$\texttt{MDfromZero}(D)$ {
if $D := true$, then return D;
else if $\exists D', (D, D') \in R'$, then return D';
$D' := false$;
if $D = (x - x', ((\sim_1, c_1), D_1), \ldots, ((\sim_n, c_n), D_n))$,
if $x =$ ZMID,
if $x' = 0$,
for $i := n$ to 1, {
$(\sim, c) := \texttt{ZoneLbAdd}((\sim_i, c_i), \text{ZB})$;
if $(\sim, c) \sqsubset (\leq, 0)$, then { $R' := R' \cup \{(D, D')\}$; return D'; }
$D' := D' \cup (x - x' \sim_i c_i \sqcap \texttt{MDfromZero}(D_i))$;
}
else
for $i := n$ to 1, {
$(\sim, c) := \texttt{ZoneLbAdd}((\sim_i, c_i), \text{ZB})$;
if $(\sim, c) \sqsupset (\leq, 0)$, then $(\sim, c) := (\leq, 0)$;
$D' := D' \cup (x - x' \sim_i c_i \sqcap 0 - x' \sim c \sqcap \texttt{MDfromZero}(D_i))$;
}
else for $i := n$ to 1, $D' := D' \cup (x - x' \sim_i c_i \sqcap \texttt{MDfromZero}(D_i))$;
else /* $D = (v, ([l_1, u_1], D_1), \ldots, ([l_n, u_n], D_n))$, */
for $i := n$ to 1, $D' := D' \cup (l_i \leq v \leq u_i \sqcap \texttt{MDfromZero}(D_i))$;
$R' := R' \cup \{(D, D')\}$; return D';
}
$\texttt{MDtoZero}(D)$ {
if $D := true$, then return D;
else if $\exists D', (D, D') \in R'$, then return D';
$D' := false$;
if $D = (x - x', ((\sim_1, c_1), D_1), \ldots, ((\sim_n, c_n), D_n))$,

```
    if x' = ZMID,
      if x = 0,
        for i := n to 1, {
          (∼, c) := ZoneLbAdd((∼_i, c_i), ZB);
          if (∼, c) ⊏ (≤, 0), { R' := R' ∪ {(D, D')}; return D'; }
          D' := D' ∪ (x − x' ∼_i c_i ⊓ MDtoZero(D_i));
        }
      else
        for i := n to 1, {
          (∼, c) := ZoneLbAdd((∼_i, c_i), ZB);
          if (∼, c) ⊐ (≤, 0), { R' := R' ∪ {(D, D')}; return D'; }
          D' := D' ∪ (x − x' ∼_i c_i ⊓ x − 0 ∼ c ⊓ MDtoZero(D_i));
        }
    else for i := n to 1, D' := D' ∪ (x − x' ∼_i c_i ⊓ MDtoZero(D_i));
  else /* D = (v, ([l_1, u_1], D_1), ..., ([l_n, u_n], D_n)), */
    for i := n to 1, D' := D' ∪ (l_i ≤ v ≤ u_i ⊓ MDtoZero(D_i));
  R' := R' ∪ {(D, D')}; return D';
}
```

The second technique is called *downward redundancy elimination (DRE)*. For convenience, let $\lfloor(\leq, c)\rfloor = c$, $\lfloor(<, c)\rfloor = c-1$, $\lceil(\leq, c)\rceil = c$, $\lceil(<, c)\rceil = c+1$, and $\text{diff}(\sim_1, c_1, \sim_2, c_2) = \lfloor(\sim_1, c_1)\rfloor - \lceil(\sim_2, c_2)\rceil$. We identify two types of constraint redundancies with respect to a variable ordering. Given a zone ζ and an evaluation index Ω, a constraint $x_1 - x_2 \sim c \in \zeta$ is

- *type I downward redundant* iff there are $x_1 - x_3 \sim_1 c_1, x_3 - x_2 \sim_2 c_2 \in \zeta$ such that $x_1 - x_3 \prec_\Omega x_1 - x_2$, $x_3 - x_2 \prec_\Omega x_1 - x_2$, $c_1 + c_2 < c \vee (c_1 + c_2 = c \wedge (\sim = \text{'}\leq\text{'} \vee \sim_1 = \text{'}<\text{'} \vee \sim_2 = \text{'}<\text{'})$.
- *type II downward redundant* iff there are $x_1 - x_3 \sim_1 c_1, x_3 - x_2 \sim_2 c_2 \in \zeta$ such that either
  - $x_1 - x_3 \prec_\Omega x_1 - x_2 \prec_\Omega x_3 - x_2$ and $c_2 \leq \text{diff}(\sim, c, \sim_1, c_1)$; or
  - $x_3 - x_2 \prec_\Omega x_1 - x_2 \prec_\Omega x_1 - x_3$ and $c_1 \leq \text{diff}(\sim, c, \sim_2, c_2)$.

The technique DRE is to iteratively pick downward redundant constraints from zones in a CRD and eliminate them until there are no more downward redundant constraints left. Since we pick the downward redundant constraints according to the evaluation ordering, all the downward redundant constraints can be eliminated in one traversal of the given CRD.

The algorithm for downward redundancy elimination is in the following. It traverse through all the CRD nodes and for each clock constraint variable, ZX − ZY, it then calls `EliminateOneGroupDRE`(D'_i) to see if any downward redundancies constructed with constraints ZX − ZY can be identified and eliminated in D'_i.

```
set of pairs of CRD+BDD   R, R'; /* two static set variables */
DRE(D) {
  R := ∅; return recDRE(D);
}
clock          ZX, ZY; /* two static clock variables */
element in B_{C_A}   ZB; /* one static upperbound variable */
recDRE(D) {
  if D := true, then return D;
  else if ∃D', (D, D') ∈ R, then return D';
  D' := false;
  if D = (x − x', ((∼_1, c_1), D_1), ..., ((∼_n, c_n), D_n)),
    for i := n to 1, do {
      D'_i := recDRE(D_i));
      ZX := x; ZY := x'; ZB := (∼_i, c_i); R' := ∅;
      D' := D' ∪ (x − x' ∼_i c_i ⊓ EliminateOneGroupDRE(D'_i));
    }
  else /* D = (v, ([l_1, u_1], D_1), ..., ([l_n, u_n], D_n)), */
    for i := n to 1, D' := D' ∪ (l_i ≤ v ≤ u_i ⊓ recDRE(D_i));
  R := R ∪ {(D, D')}; return D';
}
```

Procedure `EliminateOneGroupDRE`(D) uses three new basic routines as follows.

- DiffExtract$(D, x - x', (\sim_1, c_1), (\sim_2, c_2))$ returns a CRD+BDD identical to D except that all paths representing zones whose constraints on $x - x'$, say $(\sim, c)$, do not fall within $(\sim_1, c_1)$ and $(\sim_2, c_2)$, i.e. $(\sim, c) \sqsubset (\sim_1, c_1) \vee (\sim_2, c_2) \sqsubset (\sim, c)$, are eliminated.

- DiffSubtract$(D, x-x', (\sim_1, c_1), (\sim_2, c_2))$ returns a CRD+BDD identical to D except that all paths representing zones whose constraints on $x - x'$, say $(\sim, c)$, fall within $(\sim_1, c_1)$ and $(\sim_2, c_2)$, i.e. $(\sim_1, c_1) \sqsubset (\sim, c) \sqsubset (\sim_2, c_2)$, are eliminated.
- `ZoneLbSubtract`$((\sim_1, c_1), (\sim_2, c_2))$ {
  if $(\sim_1, c_1) = (<, -C_A) \vee (\sim_2, c_2) = (<, \infty)$, then return $(<, -C_A)$;
  if $(\sim_1, c_1) = (<, \infty)$, then $\iota_1 := C_A$; else if $\sim_1 =$ '$<$', then $\iota_1 := c_1 - 1$; else $\iota_1 := c_1$;
  if $(\sim_2, c_2) = (<, -C_A)$, then $\iota_2 := -C_A$; else $\iota_2 := c_2$;
  if $\iota_1 - \iota_2 > C_A$, return $(<, \infty)$; else if $\iota_1 - \iota_2 < -C_A$, return $(<, C_A)$; else return $(\leq, \iota_1 - \iota_2)$;
  }

`EliminateOneGroupDRE`(D) {
  if $D := \mathit{true}$, then return D;
  else if $\exists D', (D, D') \in R'$, then return D';
  $D' := \mathit{false}$;
  if $D = (x - x', ((\sim_1, c_1), D_1), \ldots, ((\sim_n, c_n), D_n))$, {
    if $x =$ ZY, then
      if $x' =$ ZX, then /* elimination of negative cycles. */
        for $i := n$ to 1, do {
          $(\sim, c) :=$ `ZoneLbAdd`(ZB, $(\sim_i, c_i)$);
          if $(\sim, c) \sqsubset (\leq, 0)$, then { $R' := R' \cup \{(D, D')\}$; return D'; }
          else $D' := D' \cup (x - x' \sim_i c_i \sqcap$ `EliminateOneGroupDRE`$(D_i))$;
        }
      else /* Type I DRE */
        for $i := n$ to 1, do {
          $(\sim, c) :=$ `ZoneLbAdd`(ZB, $(\sim_i, c_i)$);
          if $x' = 0 \wedge (\sim, c) \sqsubset (0, \leq)$, then { $R' := R' \cup \{(D, D')\}$; return D'; }
          else if ZX $= 0 \wedge (\sim, c) \sqsupset (0, \leq)$, then $D'_i :=$ `EliminateOneGroupDRE`$(D_i))$;
          else {
            $D'_i :=$ `EliminateOneGroupDRE`$(D_i))$;
            if $x - x' \prec_\Omega$ ZX $- x'$ {
              $D^2_i :=$ `VariableEliminate`(`DiffExtract`$(D'_i,$ ZX $- x', (\sim, c), (<, \infty)),$ ZX $- x')$;
              $D'_i :=$ `DiffSubtract`$(D'_i,$ ZX $- x', (\sim, c), (<, \infty)) \cup D^2_i$;
            }
          }
          $D' := D' \cup (x - x' \sim_i c_i \sqcap D'_i)$;
        }
    else if $x' =$ ZX, /* Type I DRE */
      for $i := n$ to 1, do {
        $(\sim, c) :=$ `ZoneLbAdd`(ZB, $(\sim_i, c_i)$);
        if ZY $= 0 \wedge (\sim, c) \sqsubset (\leq, 0)$, then { $R' := R' \cup \{(D, D')\}$; return D'; }
        else if $x = 0 \wedge (\sim, c) \sqsupset (\leq, 0)$, then $D'_i :=$ `EliminateOneGroupDRE`$(D_i))$;
        else {
          $D'_i :=$ `EliminateOneGroupDRE`$(D_i))$;
          if $x - x' \prec_\Omega x -$ ZY {
            $D^2_i :=$ `VariableEliminate`(`DiffExtract`$(D'_i, x -$ ZY$, (\sim, c), (<, \infty)), x -$ ZY$)$;
            $D'_i :=$ `DiffSubtract`$(D'_i, x -$ ZY$, (\sim, c), (<, \infty)) \cup D^2_i$;
          }
        }
        $D' := D' \cup (x - x' \sim_i c_i \sqcap D'_i)$;
      }
    else if $x =$ ZX, /* Type II DRE */
      for $i := n$ to 1, do {
        $D'_i :=$ `EliminateOneGroupDRE`$(D_i))$;
        if $x - x' \prec_\Omega$ ZY $- x' \wedge (\sim_i, c_i) \sqsubset (<, \infty)$, {
          $(\sim, c) :=$ `ZoneLbSubtract`$((\sim_i, c_i),$ ZB$)$;
          if $(\sim, c) \sqsupset (<, -C_A)$, {
            $D^2_i :=$ `DiffExtract`$(D'_i,$ ZY $- x', (<, -C_A), (\sim, c))$;
            $D'_i := (x - x' \sim_i c_i \sqcap$ `DiffSubtract`$(D'_i,$ ZY $- x', (<, -C_A), (\sim, c))) \cup D^2_i$;
          }
          else $D'_i := D'_i \sqcap x - x' \sim_i c_i$;

}
else $D'_i := D'_i \sqcap x - x' \sim_i c_i$;
$D' := D' \cup D'_i$;
}
else if $x' = \mathrm{ZY}$, /* Type II DRE */
for $i := n$ to 1, do {
$D'_i :=$ `EliminateOneGroupDRE`(D_i));
if $x - x' \prec_\Omega x - \mathrm{ZX}$, {
$(\sim, c) :=$ `ZoneLbSubtract`$((\sim_i, c_i), \mathrm{ZB})$;
if $(\sim, c) \sqsupset (<, -C_A) \wedge (\sim_i, c_i) \sqsubset (<, \infty)$, {
$D_i^2 :=$ `DiffExtract`$(D'_i, x - \mathrm{ZX}, (<, -C_A), (\sim, c))$;
$D'_i := (x - x' \sim_i c_i \sqcap$ `DiffSubtract`$(D'_i, x - \mathrm{ZX}, (<, -C_A), (\sim, c))) \cup D_i^2$;
}
else $D'_i := D'_i \sqcap x - x' \sim_i c_i$;
}
else $D'_i := D'_i \sqcap x - x' \sim_i c_i$;
$D' := D' \cup D'_i$;
}
else /* No DRE applicable */ for $i := n$ to 1, $D' := D' \cup \cup (x - x' \sim_i c_i \sqcap$ `EliminateOneGroupDRE`$(D_i))$;
else /* $D = (v, ([l_1, u_1], D_1), \ldots, ([l_n, u_n], D_n))$, */
for $i := n$ to 1, $D' := D' \cup (l_i \le v \le u_i \sqcap$ `EliminateOneGroupDRE`$(D_i))$;
$R' := R' \cup \{(D, D')\}$; return D';
}

In section 9, we shall report experiments on various canonical forms and the MD&DRE techniques.

7 Variable ordering in CRD+BDD

The manipulation efficiency of BDD-like data-structure is strongly related to the variable evaluation ordering. Traditional wisdom is that we should place two strongly related variables close to each other in the ordering. We consider three possibilities of variable-ordering to test how CRD+BDD reacts to variable orderings.

- NIL, with no interleaving between discrete variables and clock constraints in the ordering.
- HIL, only with interleaving between discrete variables and magnitude clock constraints, and
- FIL, with full interleaving between discrete variables and clock constraints.

When interleaving is implemented, we consider the precedence of process identifiers and variable declaration ordering to define the interleaving ordering. For example, we may have a system with two processes with global discrete variable u, global clock x, local discrete variable w_1 and local clock x_1 for process 1, and local discrete variable w_2 and local clock x_2 for process 2. The evaluation orderings among $u, w_1, w_2, 0-x, 0-y_1, y_1-0, x-y_1, 0-y_2$ are

$$\begin{array}{ll}
\text{NIL}: & u \prec_{NIL} w_1 \prec_{NIL} w_2 \prec_{NIL} 0-x \prec_{NIL} 0-y_1 \prec_{NIL} y_1-0 \prec_{NIL} x-y_1 \prec_{NIL} 0-y_2 \\
\text{HIL}: & u \prec_{HIL} 0-x \prec_{HIL} w_1 \prec_{HIL} 0-y_1 \prec_{HIL} y_1-0 \prec_{HIL} w_2 \prec_{HIL} 0-y_2 \prec_{HIL} x-y_1 \\
\text{FIL}: & u \prec_{FIL} 0-x \prec_{FIL} w_1 \prec_{FIL} 0-y_1 \prec_{FIL} y_1-0 \prec_{FIL} x-y_1 \prec_{FIL} w_2 \prec_{FIL} 0-y_2
\end{array}$$

Our experiments show performance data very compatible with the traditional wisdom. That is, FIL is more efficient than HIL, which is in turn more efficient than NIL. Due to page-limit, we shall leave the details of the three ordering definitions and their experiment data to appendix E.

8 Implementation

We have implemented our CRD-technology in version 3.1 of our tool `red`, whose previous version (versions 1.0, 2.0, and 3.0) was announced in [18, 19, 20, 21]. `red` 3.1 supports TCTL-model-checking of real-time systems with multiprocesses, pointer data-structures, and synchronizations (synchronous send and receive) from one process to another. The new version, together with benchmarks, will soon be available at:

`http://www.iis.sinica.edu.tw/~farn/red`

Each process can use *global* and *local* variables of type *clock*, *discrete*, and *pointer*. Pointer variables either contain value NULL or the identifiers of processes. Thus in the models input to `red`, we allow complicate dynamic networks to be constructed with pointers.

At this moment, `red` runs with backward reachability analysis, evaluation ordering FIL, and MD&DRE normalizations by default. We have also implemented the reduction by elimination of inactive variables[14, 22], which is always executed. A variable is *inactive* in a state iff it is not read in any computation from the state before its content is overwritten. Contents of inactive variables can be omitted from state information without any effect on the computations.

9 Experiments

We carried out three experiments to see in reality how well MD&DRE performs compared to other tools and other implementation strategies of CRD-technologies. We choose the following nine targets for comparison.

- *Kronos*[5, 11, 24] (version 2.4 release 4), which supports forward and backward model-checking of TCTL[2].
- *UPPAAL2k*[6] (version 3.2.4), which supports on-the-fly forward analysis with reduced zone representations and various searching and reduction strategies.
- *CRD/Closure+FIL*: CRD in closure form with evaluatoin ordering FIL,
- *CRD/Cascade+FIL*: CRD in cascade form[20] with evaluation ordering FIL,
- *CRD/Closure+DRE+FIL*: DRE after computing closure form CRD with evaluation ordering FIL,
- *CRD/MD&DRE+FIL*: MD&DRE of CRD with evaluation ordering FIL,
- *CRD/NC/MD&DRE+FIL*: MD&DRE of CRD with evaluation ordering FIL but without contained zone elimination (see subsection 5.3),
- *CRD/MD&DRE+HIL*: MD&DRE of CRD with evaluation ordering HIL, and
- *CRD/MD&DRE+NIL*: MD&DRE of CRD with evaluation ordering NIL.

Kronos and UPPAAL2k are perhaps the two best-known model-checkers for real-time systems with DBM-technology. We choose UPPAAL2k and Kronos because first, DBM is now generally considered as the best data-structure for timed automata verification, and second, these two tools are mature and have been very successful. Also comparison with these two tools also gives us some rough feeling about how well CRD-technology performs against DBM-technology with both forward analysis (i.e. UPPAAL2k), and backward analysis (i.e. Kronos).

In subsection 9.1, we observe the performances of some targets w.r.t. to the number of clocks. In subsections 9.2, we compare the performances of Kronos, UPPAAL2k, CRD with Cascade+FIL, and CRD with MD&DRE+FIL w.r.t. to timing-constant magnitude complexity respectively. Due to page-limit, the report on experiment with different variable orderings is left to appendix E.

We use six benchmarks to compare the performance.

- *Fischer's timed mutual exclusion algorithm* [3, 14, 18, 22]: The algorithm relies on a global lock and a local clock per process to control access to the critical section. Two timing constants used are 10 and 19. The property to be verified is that at any moment, no more than two processes are in the critical section.
- *CSMA/CD benchmark*[24]: Basically, this is the ethernet bus arbitration protocol with the idea of collision-and-retry. The timing constants used are 26, 52, and 808. We want to verify that at any moment, at most one process is in the transmission mode for no less than 52 time units.
- *FDDI token-ring mutual exclusion protocol* [11, 5]: We need one process to model the network and the other processes to model the stations. For each station process, two local clocks are needed. Each station process can use the token to transmit message in mandatory synchronous mode and optional asynchronous mode. The biggest timing constant used is 50*m+20, where m is the number of stations. We want to verify that at any moment, at most one stations is holding the token.
- *Scheduling problem of real-time operating system (PATHOS)*[3]: In the system, each process runs with a distinct priority in a period equal to the number of processes. Priority among processes must be observed by the scheduling policy. We want to verify that no deadlines will be missed.
- *Safeness of a leader-election algorithm*: Each process has a local pointer `parent` and a local clock. All processes initially come with its `parent` = NULL. Then a process with its `parent` = NULL may broadcast its request to be adopted by a parent. Another process with its `parent` = NULL may respond. Then the process with smaller identifier will become the parent of the other process in the requester-responder pair. The biggest timing constant used is 2. We want to verify that at any moment, there is at least a process who is a child to no other processes.
- *Bounded liveness of a leader-election algorithm*: It is the same systems used in the fifth benchmark. But we assume that a process with `parent` = NULL will finish an iteration of the algorithm in 2 time units. We want to verify that after $2\lceil \log_2 m \rceil$ time units, where m is the number of processes, the algorithm will finish.

9.1 Performance w.r.t. number of clocks

The first experiment compares the performace, w.r.t. number of clcoks, of the following six targets: Kronos 2.4.4, UPPAAL2k 3.2.4, CRD/Closure+FIL, CRD/Cascade+FIL, CRD/Closure+DRE+FIL, CRD/NC&MD&DRE+FIL CRD/MD&DRE+FIL. Table 2 shows performance data. The data in the second column to the right is collected with NC&MD&DRE, where NC means that no contained zones are detected and eliminated. Except for the FDDI benchmark, UPPAAL is invoked with forward analysis and options "`-aS0WD`." For the FDDI benchmark, UPPAAL is invoked with options "`-S0TDda`." Kronos is invoked with backward analysis.

Discussion The performance data shows that CRD-technologies are more space-efficient and scales better w.r.t. number of clocks than the DBM-technologies. The comparison between the rightest two column also implies that zone-containment relation and contained zone elimination can indeed enhance verification performance. For time complexities, MD&DRE is only outperformed in the following two cases.

- By DBM-technologies against *leader election bounded liveness* with small number of clocks. DBM-technologies usually have good time complexities with its cubic complexity all-pair shortest-path algorithm on matrices.
- By CRD/Closure+DRE+FIL against *Pathos*. This happens because of the strict timing precedence among the clocks imposed by the strict priority structure among the processes.

But in average, CRD/MD&DRE+FIL outperforms the other targets. Thus the reasoning behind the design of techniques MD&DRE seems justified.

The experiment helps us to better understand the issues raised in section 5. For example, the performance data falls very compatible with our arguments in section 5.

9.2 Performance w.r.t. timing constant magnitude complexity

The performance of some previous technologies, e.g. NDD and RED, does not scale very well to the magnitude of timing constant. The data in table 3 is collected by runnning Kronos, UPPAAL2k, CRD/Cascade+FIL, and CRD/MD&DRE+FIL against Fischers' mutual exclusion algorithm with various timing constant magnitudes. The table shows that CRD is at least as good as DBM technology as long as performance scalability with respect to timing constant complexity is concerned. In fact, the data may imply that the performance of CRD-technology can be irrelevant to the timing constant complexity in average cases.

9.3 Performance with representation fragmentation

To observe the effect of representation fragmentation phenomenon of CDD, we have endeavored to implement some CDD manipulation routines. This enables us to collect data, in table 4, of sizes of reachable state-space representations in CDD and CRD for the six benchmarks used in subsection 9.1. The state-space representation of benchmarks leader election safeness and liveness are very similar except that one more global clock is used for the liveness benchmark. This blow-up is attributed to the representation fragmentaion phenomenon of CDD.

From the table, we observe that CDD demonstrates exponential blow-up for three of the benchmarks (Fischer's, FDDI, pathos) in comparison with CRD. For the other three benchmarks (CSMA/CD, leader election safeness, liveness), CDD performs better than CRD with a nearly constant factor. This is understandable because in CDD, both lowerbound and upperbound are specified with the same CDD variable while in CRD, two separate variables have to be used. For instance, the constraint of $3 < x_1 - x_2 < 5$ can be specified with one variable in CDD. But in CRD, it is specified with two variables as $x_1 - x_2 < 5 \wedge x_2 - x_1 < -3$. This explains why when CDD performs better, it only performs better by a constant factor.

From the experiment, we argue that when the representation fragmentation phenomenon does not happen, CDD performs roughly the same as CRD (within a constant factor). But when the phenomenon happens, it takes a big toll out of the CDD space consumption. This experiment report justifies our proposal of the new data-structure of CRD.

10 Conclusion

To efficiently model-check timed automata, it takes deep understanding of the subtle interaction between data-structures and algorithms. We feel that previous BDD-like data-structures did not perform as well as DBM because such subtlety has not been paid proper attention. We have identified some of the issues in the design of efficient BDD-like data-structures and manipulation algorithms for timed automata state-spaces. We have carried out extensive experiments to justify our arguments. We have also developed techniques MD&DRE for CRD normalization. We believe the experience reported in this manuscript will be very valuable toward the implementation of industry-usable model-checkers for real-time systems.

benchmarks	concurrency	Kronos 2.4.4	UPPAAL 3.2.4	CRD/ Closure+FIL	CRD/Closure +DRE+FIL	CRD/ Cascade+FIL	CRD/NC/ MD&DRE+FIL	CRD/ MD&DRE+FIL
Fischer's mutual exclusion	3 processes	0.02s	0.01s	0.2s/17k	0.21s/17k	0.66s/47k	0.27s/18k	0.2s/18k
	4 processes	0.20s	0.09s	1.28s/41k	1.26s/43k	3.87s/166k	1.85s/52k	1.23s/43k
	5 processes	3.87s	2.97s	5.41s/94k	5.14s/90k	18.84s/467k	9.88s/148k	5.03s/94k
	6 processes	O/M	292.56s	17.11s/181k	16.50s/165k	74.12s/1129k	56.38s/431k	15.99s/191k
	7 processes	O/M	O/M	53.76s/341k	49.17s/332k	251.02s/2510k	358.88s/1522k	47.46s/384k
	8 processes	O/M	O/M	168.2s/812k	143.16s/757k	783.0s/5289k	2060.75s/5309k	139.98s/851k
	9 processes	O/M	O/M	493.6s/1875k	431.7s/1783k	N/A	10874.58s/18153k	417.00s/2032k
	10 processes	O/M	O/M	N/A	N/A	N/A	N/A	1160.17s/4736k
	11 processes	O/M	O/M	N/A	N/A	N/A	N/A	3178.93s/10897k
	12 processes	O/M	O/M	N/A	N/A	N/A	N/A	8684.63s/24802k
	13 processes	O/M	O/M	O/M	O/M	O/M	N/A	23599.32s/55978k
CSMA/CD	bus+3 senders	0.01s	0.01s	0.10s/103k	0.10s/103k	0.43s/103k	0.15s/103k	0.09s/103k
	bus+4 senders	0.02s	0.04s	0.21s/181k	0.21s/181k	1.33s/181k	0.31s/180k	0.20s/181k
	bus+5 senders	0.12s	0.46s	0.43s/292k	0.43s/292k	4.05s/292k	0.66s/291k	0.41s/292k
	bus+6 senders	0.86s	13.87s	0.92s/580k	0.92s/568k	14.54s/470k	1.38s/455k	0.84s/458k
	bus+7 senders	O/M	752.42s	2.62s/1496k	2.54s/1472k	50.14s/974k	2.99s/1015k	1.88s/723k
	bus+8 senders	O/M	O/M	5.93s/3834k	6.09s/3779k	158.67s/2048k	6.86s/2350k	4.65s/1681k
	bus+9 senders	O/M	O/M	16.61s/9709k	15.52s/9578k	466.48s/4344k	16.48s/5416k	12.45s/3971k
	bus+10 senders	O/M	O/M	46.98s/24232k	43.48s/23994k	1325.93s/9277k	N/A	35.86s/9284k
	bus+11 senders	O/M	O/M	165.51s/59762k	156.29s/59194k	3679.13s/19869k	N/A	115.14s/21470k
	bus+12 senders	O/M	O/M	830.59s/145366k	808.24s/144452k	10107.70s/42575k	N/A	424.69s/49198k
	bus+13 senders	O/M	O/M	O/M	O/M	28039.41s/91093k	N/A	1892.97s/111699k
	bus+14 senders	O/M	O/M	O/M	O/M	N/A	N/A	N/A
FDDI token-ring passing	11 stations	175.61s	30.96s	1.21s/345k	1.21s/345k	1.22s/345k	1.90s/366k	1.19s/345k
	12 stations	O/M	118.35s	1.82s/456k	1.86s/456k	1.85s/456k	2.60s/501k	1.79s/456k
	20 stations	O/M	O/M	16.90s/1311k	16.98s/1311k	16.88s/1311k	19.63s/1430k	16.59s/1311k
	30 stations	O/M	O/M	81.66s/2893k	81.57s/2893k	81.37s/2893k	111.23s/3030k	81.95s/289k
	40 stations	O/M	O/M	265.43s/5962k	265.45s/5926k	266.02s/5962k	358.44s/6406k	266.85s/5962k
	50 stations	O/M	O/M	705.6s/9902k	710.4s/9902k	N/A	865.26s/10588k	652.85s/9903k
	60 stations	O/M	O/M	N/A	N/A	N/A	1403.60s/14726k	1263.30s/14018k
pathos	3 processes	0.0s	0.01s	0.07s/25k	0.08s/25k	0.45s/44k	0.08s/25k	0.07s/25k
	4 processes	O/M	0.11s	0.33s/52k	0.34s/52k	4.28s/113k	0.39s/51k	0.28s/52k
	5 processes	O/M	8.02s	1.37s/99k	1.25s/89k	49.55s/299k	1.47s/89k	1.06s/89k
	6 processes	O/M	O/M	5.91s/284k	4.7s/194k	718.52s/1103k	5.71s/231k	3.97s/218k
	7 processes	O/M	O/M	36.89s/894k	23.99s/580k	O/M	26.41s/668k	21.12s/636k
	8 processes	O/M	O/M	195.04s/2960k	123.3s/1878k	O/M	123.04s/2110k	109.25s/1972k
	9 processes	O/M	O/M	922.52s/9915k	520.2s/6276k	O/M	527.72s/6803k	508.73s/6267k
	10 processes	O/M	O/M	3473s/33498k	1991s/21375k	O/M	N/A	2259.6s/20241k
	11 processes	O/M	O/M	17248s/112971k	O/M	O/M	N/A	8898s/65243k
	12 processes	O/M	O/M	O/M	O/M	O/M	N/A	O/M
leader	3 processes	0.06s	0.00s	0.04s/46k	0.04s/46k	0.05s/46k	0.18s/67k	0.05s/46k
	4 processes	2.19s	0.01s	0.15s/94k	0.15s/94k	0.15s/94k	0.73s/218k	0.15s/94k
	5 processes	O/M	0.09s	0.41s/165k	0.40s/165k	0.40s/165k	2.88s/753k	0.42s/165k
	6 processes	O/M	0.74s	1.19s/2632k	1.19s/263k	1.21s/263k	9.45s/2389k	1.28s/263k
	7 processes	O/M	3.34s	3.52s/479k	3.46s/479k	3.47s/479k	29.72s/7161k	3.54s/479k
	8 processes	O/M	O/M	8.66s/819k	8.65s/819k	8.67s/819k	89.13s/20882k	8.82s/819k
	9 processes	O/M	O/M	19.09s/1366k	19.17s/1366k	19.07s/1366k	297.34s/57298k	19.21s/1366k
	10 processes	O/M	O/M	37.88s/21k	38.01s/2130k	38.11s/2130k	N/A	38.14s/2130k
	11 processes	O/M	O/M	70.14s/3280k	69.67s/3280k	69.84s/3280k	N/A	70.32s/3280k
	12 processes	O/M	O/M	122.4s/4806k	122.8s/4806k	122.8s/4806k	N/A	124.2s/4806k
	13 processes	O/M	O/M	210.3s/6946k	206.6s/6946k	206.9s/6946k	N/A	207.9s/6946k
	14 processes	O/M	O/M	342.1s/9683k	342.2s/9683k	294.3s/8430k	N/A	340.2s/9683k
lbound	3 processes	0.06s	0.00s	0.24s/67k	0.23s/58k	1.08s/75k	0.25s/60k	0.17s/45k
	4 processes	2.14s	0.02s	2.62s/308k	1.97s/233k	15.21s/313k	1.75s/208k	1.24s/172k
	5 processes	O/M	0.08s	26.88s/1239k	16.38s/1149k	237.8s/1770k	10.99s/862k	10.78s/846k
	6 processes	O/M	0.65s	130.1s/4959k	78.30s/4744k	2188s/6691k	61.96s/3336k	78.24s/3312k
	7 processes	O/M	3.14s	1518s/19614k	708.28s/19006k	10981s/21800k	290.82s/11042k	458.5s/11693k
	8 processes	O/M	O/M	6370s/68952k	3341s/67471k	N/A	1285.94s/33079k	2756s/37977k
	9 processes	O/M	O/M	O/M	O/M	N/A	6676.47s/93042k	19289s/114418k

data collected on a Pentium 4 1.7GHz with 256MB memory running LINUX;
s: seconds; k: kilobytes of memory in data-structure; O/M: Out of memory; N/A: not available;

Table 2: Performance data of scalability w.r.t. number of processes

Tools	# proc	$C_A = 38$	$C_A = 76$	$C_A = 152$	$C_A = 304$	$C_A = 608$	$C_A = 1216$
Kronos	3	0.04s	0.03s	0.03s	0.03s	0.03s	0.04s
	4	0.21s	0.20s	0.20s	0.21s	0.21s	0.21s
UPPAAL	3	0.01s	0.01s	0.01s	0.01s	0.01s	0.01s
	4	0.09s	0.09s	0.09s	0.09s	0.09s	0.09s
Cascade	3	0.65s/46k	0.69s/46k	0.67s/46k	0.67s/46k	0.66s/46k	0.68s/46k
+FIL	4	3.89s/166k	3.89s/166k	3.86s/166k	3.96s/166k	3.92s/166k	3.91s/166k
MD&DRE	3	0.22s/18k	0.20s/18k	0.21s/19k	0.215s/18k	0.21s/18k	0.20s/18k
+FIL	4	1.25s/43k	1.24s/43k	1.24s/43k	1.23s/43k	1.23s/43k	1.23s/43k

data collected on a Pentium 4 1.7GHz with 256MB memory running LINUX;
s: seconds; k: kilobytes of memory in data-structure; O/M: Out of memory; N/A: not available;

Table 3: Performance data of scalability w.r.t. timing-constant magnitude

Acknowledgment

The author would like to thank Ms. Yu-Feng Chen, Mr. Geng-Dian Huang, and Mr. Fang Yu who helped collecting the huge set of experiment data.

References

[1] Asaraain, Bozga, Kerbrat, Maler, Pnueli, Rasse. Data-Structures for the Verification of Timed Automata. Proceedings, HART'97, LNCS 1201.

[2] R. Alur, C. Courcoubetis, D.L. Dill. Model Checking for Real-Time Systems, IEEE LICS, 1990.

[3] F. Balarin. Approximate Reachability Analysis of Timed Automata. IEEE RTSS, 1996.

[4] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L.Dill, L.J. Hwang. Symbolic Model Checking: 10^{20} States and Beyond, IEEE LICS, 1990.

[5] M. Bozga, C. Daws. O. Maler. Kronos: A model-checking tool for real-time systems. 10th CAV, June/July 1998, LNCS 1427, Springer-Verlag.

[6] J. Bengtsson, K. Larsen, F. Larsson, P. Pettersson, Wang Yi. UPPAAL - a Tool Suite for Automatic Verification of Real-Time Systems. Hybrid Control System Symposium, 1996, LNCS, Springer-Verlag.

[7] G. Behrmann, K.G. Larsen, J. Pearson, C. Weise, Wang Yi. Efficient Timed Reachability Analysis Using Clock Difference Diagrams. CAV'99, July, Trento, Italy, LNCS 1633, Springer-Verlag.

[8] R.E. Bryant. Graph-based Algorithms for Boolean Function Manipulation, IEEE Trans. Comput., C-35(8), 1986.

[9] E. Clarke, O. Grumberg, M. Minea, D. Peled. State-Space Reduction using Partial-Ordering Techniques, STTT 2(3), 1999, pp.279-287.

[10] D.L. Dill. Timing Assumptions and Verification of Finite-state Concurrent Systems. CAV'89, LNCS 407, Springer-Verlag.

[11] C. Daws, A. Olivero, S. Tripakis, S. Yovine. The tool KRONOS. The 3rd Hybrid Systems, 1996, LNCS 1066, Springer-Verlag.

[12] E.A. Emerson, A.P. Sistla. Utilizing Symmetry when Model-Checking under Fairness Assumptions: An Automata-Theoretic Approach. ACM TOPLAS, Vol. **19**, Nr. 4, July 1997, pp. 617-638.

[13] T.A. Henzinger, X. Nicollin, J. Sifakis, S. Yovine. Symbolic Model Checking for Real-Time Systems, IEEE LICS 1992.

[14] P.-A. Hsiung, F. Wang. User-Friendly Verification. Proceedings of 1999 FORTE/PSTV, October, 1999, Beijing. Formal Methods for Protocol Engineering and Distributed Systems, editors: J. Wu, S.T. Chanson, Q. Gao; Kluwer Academic Publishers.

benchmarks	concurrency	CDD			CRD			Size Ratio
		#nodes	#arcs	size	#nodes	#arcs	size	CDD/CRD
Fischer's mutual exclusion	3 processes	53	103	156	66	113	179	0.872
	4 processes	103	217	320	127	232	359	0.891
	5 processes	174	382	556	208	393	601	0.925
	6 processes	276	620	896	311	599	910	0.985
	7 processes	429	977	1406	438	854	1292	1.088
	8 processes	673	1545	2218	591	1162	1753	1.265
	9 processes	1088	2508	3596	772	1527	2299	1.564
	10 processes	1834	4234	6068	983	1953	2936	2.067
CSMA/CD	bus+3 senders	38	52	90	46	53	99	0.909
	bus+4 senders	48	67	115	54	63	117	0.983
	bus+5 senders	58	82	140	62	73	135	1.037
	bus+6 senders	68	97	165	70	83	153	1.078
	bus+7 senders	78	112	190	78	93	171	1.111
	bus+8 senders	88	127	215	86	103	189	1.138
	bus+9 senders	98	142	240	94	113	207	1.159
	bus+10 senders	108	157	265	102	123	225	1.178
FDDI token-ring passing	3 stations	46	62	108	59	72	131	0.824
	4 stations	92	128	220	115	140	255	0.863
	5 stations	226	390	616	204	250	454	1.357
	6 stations	851	1667	2518	418	501	919	2.740
	7 stations	4640	10051	14691	876	1026	1902	7.724
	8 stations	32404	75098	107502	1858	2133	3991	26.936
pathos schedulliblity	3 processes	26	34	60	27	32	59	1.017
	4 processes	102	152	254	94	119	213	1.192
	5 processes	375	616	991	273	350	623	1.591
	6 processes	1512	2714	4226	781	1002	1783	2.370
	7 processes	6773	13112	19885	2256	2886	5142	3.867
	8 processes	33134	68228	101362	6577	8382	14959	6.776
	9 processes	176507	381586	558093	19288	24490	43778	12.748
	10 processes	1027935	2304923	3332858	56779	71845	128624	25.912
leader election safeness	3 processes	125	155	280	150	180	330	0.848
	4 processes	287	365	652	337	413	750	0.869
	5 processes	546	703	1249	630	780	1410	0.886
	6 processes	925	1200	2125	1053	1312	2365	0.899
	7 processes	1447	1887	3334	1630	2040	3670	0.908
	8 processes	2485	3145	5630	2735	3345	6080	0.926
	9 processes	3536	4479	8015	3866	4732	8598	0.932
	10 processes							
leader election bounded liveness	3 processes	157	180	337	183	205	388	0.869
	4 processes	499	621	1120	555	651	1206	0.929
	5 processes	1886	2428	4314	1848	2184	4032	1.070
	6 processes	5033	6785	11818	4596	5618	10214	1.157
	7 processes	9952	13803	23755	8737	10922	19659	1.208
	8 processes	17064	24089	41153	14543	18385	32928	1.250
	9 processes	41011	58941	99952	29662	37223	66885	1.494

data collected on a Pentium 4 1.7GHz with 256MB memory running LINUX;
#node: number of nodes; #arcs: number of arcs; sizes = #node + #arcs;

Table 4: Representation complexity of reachable state-spaces in CDD and CRD

[15] K.G. Larsen, F. Larsson, P. Pettersson, Y. Wang. Efficient Verification of Real-Time Systems: Compact Data-Structure and State-Space Reduction. IEEE RTSS, 1998.

[16] J. Moller, J. Lichtenberg, H.R. Andersen, H. Hulgaard. Difference Decision Diagrams, in proceedings of Annual Conference of the European Association for Computer Science Logic (CSL), Sept. 1999, Madreid, Spain.

[17] J. Moller, J. Lichtenberg, H.R. Andersen, H. Hulgaard. Fully Symbolic Model-Checking of Timed Systems using Difference Decision Diagrams, in proceedings of Workshop on Symbolic Model-Checking (SMC), July 1999, Trento, Italy.

[18] F. Wang. Efficient Data-Structure for Fully Symbolic Verification of Real-Time Software Systems. TACAS'2000, March, Berlin, Germany. in LNCS 1785, Springer-Verlag.

[19] F. Wang. Region Encoding Diagram for Fully Symbolic Verification of Real-Time Systems. the 24th COMPSAC, Oct. 2000, Taipei, Taiwan, ROC, IEEE press.

[20] F. Wang. RED: Model-checker for Timed Automata with Clock-Restriction Diagram. Workshop on Real-Time Tools, Aug. 2001, Technical Report 2001-014, ISSN 1404-3203, Dept. of Information Technology, Uppsala University.

[21] F. Wang. Symbolic Verification of Complex Real-Time Systems with Clock-Restriction Diagram, to appear in Proceedings of FORTE, August 2001, Cheju Island, Korea.

[22] F. Wang, P.-A. Hsiung. Automatic Verification on the Large. Proceedings of the 3rd IEEE HASE, November 1998.

[23] F. Wang, A. Mok, E.A. Emerson. Symbolic Model-Checking for Distributed Real-Time Systems. In proceedings of 1st FME, April 1993, Denmark; LNCS 670, Springer-Verlag.

[24] S. Yovine. Kronos: A Verification Tool for Real-Time Systems. International Journal of Software Tools for Technology Transfer, Vol. 1, Nr. 1/2, October 1997.

A Procedure $\sqcap()$

We assume two basic routines

- CRD$(x - x', (\sim, c), D)$ construct the single arc CRD of $(x - x', ((\sim, c), D))$.
- BDD$(v, [l, u], D)$ construct the single arc BDD whose root variable is v, arc label is $[l, u]$, and whose only child is CRD+BDD D.

set R;
$\sqcap(\dot{D}, \ddot{D})$ {
  $R := \emptyset$;
  return rec$\sqcap$ $(\dot{D}, \ddot{D})$;
}

rec$\sqcap(\dot{D}, \ddot{D})$ {
  if var$(\dot{D})$ is *true*, return $\ddot{D}$;
  else if var$(\ddot{D})$ is *true*, return $\dot{D}$;
  else if $\exists D', (\dot{D}, \ddot{D}, D') \in R$, return D'; ..... (1)
  else if var$(\dot{D}) \prec_\Omega$ var$(\ddot{D})$,
    if var$(\dot{D})$ is like $x - x'$ and $\dot{D} = (x - x', ((\sim_1, c_1), D_1), \ldots, ((\sim_n, c_n), D_n))$, {
      $D' := \bigcup_{1 \le i \le n}$ CRD$(x - x', (\sim, c_i),$ rec$\sqcap(D_i, \ddot{D}))$;
    else if var$(\dot{D})$ is a discrete variable like v and $\dot{D} = (v, ((l_1, u_1), D_1), \ldots, ((l_n, u_n), D_n))$, then
      $D' := \bigcup_{1 \le i \le n}$ BDD$(v, [l_i, u_i],$ rec$\sqcap(D_i, \ddot{D}))$;
  }
  else if var$(\dot{D}) \succ_\Omega$ var$(\ddot{D})$,
    if var$(\ddot{D})$ is like $x - x'$ and $\ddot{D} = (x - x', ((\sim_1, c_1), D_1), \ldots, ((\sim_n, c_n), D_n))$, {
      $D' := \bigcup_{1 \le i \le n}$ CRD$(x - x', (\sim, c_i),$ rec$\sqcap(\dot{D}, D_i))$;
    else if var$(\ddot{D})$ is a discrete variable like v and $\ddot{D} = (v, ((l_1, u_1), D_1), \ldots, ((l_n, u_n), D_n))$, then
      $D' := \bigcup_{1 \le i \le n}$ BDD$(v, [l_i, u_i],$ rec$\sqcap(\dot{D}, D_i))$;
  }
  else if var$(\dot{D})$ is like $x - x'$, $\dot{D} = (x - x', ((\dot{\sim}_1, \dot{c}_1), \dot{D}_1), \ldots, ((\dot{\sim}_n, \dot{c}_n), \dot{D}_n))$,
  and $\ddot{D} = (x - x', ((\ddot{\sim}_1, \ddot{c}_1), \ddot{D}_1), \ldots, ((\ddot{\sim}_m, \ddot{c}_m), \ddot{D}_m))$, then {
    $i := n; j := m; D' :=$ *false*; $\dot{L} :=$ *false*; $\ddot{L} :=$ *false*;;
    while $i \ge 1$ or $j \ge 1$, do {
      if $i \ge 1 \wedge j \ge 1 \wedge \dot{\sim}_i, \dot{c}_i) = (\ddot{\sim}_j, \ddot{c}_j$, {
        $\dot{L} := \dot{L} \cup \dot{D}_i$; $\ddot{L} := \ddot{L} \cup \ddot{D}_j$;
        $D' := D' \cup$ CRD$(x - x', (\dot{\sim}_i, \dot{c}_i),$ rec$\sqcap(\dot{L}, \ddot{L}))$;
        $i := i - 1; j := j - 1$;
      }
      else if $i = 0 \vee (i \ge 1 \wedge j \ge 1 \wedge \dot{\sim}_i, \dot{c}_i) \sqsubseteq (\ddot{\sim}_j, \ddot{c}_j)$, {
        $\ddot{L} := \ddot{L} \cup \ddot{D}_j$;
        $D' := D' \cup$ CRD$(x - x', (\ddot{\sim}_j, \ddot{c}_j),$ rec$\sqcap(\dot{L}, \ddot{L}))$;
        $j := j - 1$;
      }
      else if $j = 0 \vee (i \ge 1 \wedge j \ge 1 \wedge \ddot{\sim}_j, \ddot{c}_j) \sqsubseteq (\dot{\sim}_i, \dot{c}_i)$, {
        $\dot{L} := \dot{L} \cup \dot{D}_i$;
        $D' := D' \cup$ CRD$(x - x', (\dot{\sim}_j, \dot{c}_i),$ rec$\sqcap(\dot{L}, \ddot{L}))$;
        $i := i - 1$;
      }
    }
  }
  else if var$(\dot{D})$ is a discrete variable like v, $\dot{D} = (v, ((\dot{l}_1, \dot{u}_1), \dot{D}_1), \ldots, ((\dot{l}_n, \dot{u}_n), \dot{D}_n))$,
  and $\ddot{D} = (v, ((\ddot{l}_1, \ddot{u}_1), \ddot{D}_1), \ldots, ((\ddot{l}_m, \ddot{u}_m), \ddot{D}_m))$, then {
    $i := n; j := m; D' :=$ *false*;
    while $i \ge 1$ or $j \ge 1$, do
      if $i = 0 \vee (i \ge 1 \wedge j \ge 1 \wedge \dot{u}_i < \ddot{l}_j)$, then $j := j - 1$;

else if $j = 0 \vee (i \geq 1 \wedge j \geq 1 \wedge \dot{l}_i > \ddot{u}_j)$, then $i := i - 1$;
else {
$l = \max(\dot{l}_i, \ddot{l}_j); u := \min(\dot{u}_i, \ddot{u}_j)$; $D' := D' \cup \mathrm{BDD}(v, [l, u], \mathrm{rec}\sqcap(\dot{D}_i, \ddot{D}_j))$;
if $\dot{l}_i = l$, then $i := i - 1$; else $j := j - 1$;
}
}
$R := R \cup \{(\dot{D}, \ddot{D}, D')\}$; ..... (2)
return D';
}

B Procedure VariableEliminate()

clock MID;
set R;
VariableEliminate(D, w) {
MID $:= w$; $R := \emptyset$;
return recVariableEliminate(D);
}
recVariableEliminate(D) {
if either var(D) is *true* or var$(D) \succ_\Omega$ MID, return D;
else if $\exists D', (D, D') \in R$, return D'; ..... (1)
else if var(D) is like $x - x'$ and $D = (x - x', ((\sim_1, c_1), D_1), \ldots, ((\sim_n, c_n), D_n))$,
if MID is $x - x'$, $D' = \bigcup_{1 \leq i \leq n} D_i$; else $D' := \bigcup_{1 \leq i \leq n} (x - x' \sim c_i \sqcap \mathrm{recVariableEliminate}(D_i))$;
else if var(D) is a discrete variable like v and $D = (v, ((l_1, u_1), D_1), \ldots, ((l_n, u_n), D_n))$, then
if MID is v, $D' = \bigcup_{1 \leq i \leq n} D_i$; else $D' := \bigcup_{1 \leq i \leq n} (l_i \leq v \leq u_i \sqcap \mathrm{recVariableEliminate}(D_i))$;
$R := R \cup \{(D, D')\}$; ..... (2)
return D';
}

C Procedure ClockEliminate()

clock MID;
set R;
ClockEliminate(D, x) {
MID $:= x$; $R := \emptyset$;
return recClockEliminate(D);
}
recClockEliminate(D) {
if either var(D) is *true*, return D;
else if $\exists D', (D, D') \in R$, return D'; ..... (1)
else if var(D) is like $x_1 - x_2$ and $D = (x_1 - x_2, ((\sim_1, c_1), D_1), \ldots, ((\sim_n, c_n), D_n))$,
if MID is x_1 or MID is x_2, $D' = \bigcup_{1 \leq i \leq n} D_i$; else $D' := \bigcup_{1 \leq i \leq n} (x_1 - x_2 \sim c_i \sqcap \mathrm{recClockEliminate}(D_i))$;
else if var(D) is a discrete variable like v and $D = (v, ((l_1, u_1), D_1), \ldots, ((l_n, u_n), D_n))$, then
$D' := \bigcup_{1 \leq i \leq n} (l_i \leq v \leq u_i \sqcap \mathrm{recClockEliminate}(D_i))$;
$R := R \cup \{(D, D')\}$; ..... (2)
return D';
}

D Procedure Bypass()

Given two $(\sim_1, c_1), (\sim_2, c_2) \in \mathcal{B}_{C_A}$, we define $(\sim, c) = (\sim_1, c_1) + (\sim_2, c_2)$ such that

- if $c_1 + c_2 > C_A$, then $(\sim, c) = (<, \infty)$;
- else if $c_1 + c_2 < -C_A$ or $c_1 + c_2 = -C_A \wedge (\sim_1 = \text{'}<\text{'} \vee \sim_2 = \text{'}<\text{'})$, then $(\sim, c) = (<, -\infty)$;
- else $c = c_1 + c_2$, $\sim = \text{'}\leq\text{'}$ when $\sim_1 = \sim_2 = \text{'}\leq\text{'}$, and $\sim = \text{'}<\text{'}$ when $\sim_1 = \text{'}<\text{'} \vee \sim_2 = \text{'}<\text{'}$.

```
Bypass(D, x) {
   for all x_1 ∈ X − {x}, D := BypassRight(BypassLeft(D, x_1, x), x, x_1);
   return D;
}

clock LEFT, MID, RIGHT;
int B;
set R;
BypassLeft(D, x_1, x) {
   LEFT := x_1; MID := x; B := ∞; R := ∅;
   return recBypassLeft(D);
}

recBypassLeft(D) {
   if var(D) is true, return D;
   else if ∃D', (D, B, D') ∈ R, return D'; ..... (1)a
   D' := false;
   if var(D) is like x − x' and D = (x − x', ((∼_1, c_1), D_1), ..., ((∼_n, c_n), D_n)), {
      if x = LEFT and x' = MID, then
         for i := 1 to n, { B := β_i; D' := D' ∪ (x − x' ∼_i c_i ⊓ recBypassLeft(D_i)); }
      else if x = MID and x' ≠ LEFT,
         for i := 1 to n, { (∼, c) := B + β_i; D' := D' ∪ (x − x' ∼ c ⊓ recBypassLeft(D_i)); }
      else for i := 1 to n, D' := D' ∪ (x − x' ∼ c ⊓ recBypassLeft(D_i));
   }
   else if var(D) is a discrete variable like v and D = (v, ((l_1, u_1), D_1), ..., ((l_n, u_n), D_n)), then
      D' := ⋃_{1≤i≤n}(l_i ≤ v ≤ u_i ⊓ recBypassLeft(D_i));
   R := R ∪ {(D, B, D')}; ..... (2)a
   return D';
}

BypassRight(D, x, x_2) {
   MID := x; RIGHT := x_2; B := ∞; R := ∅;
   return recBypassRight(D);
}

recBypassRight(D) {
   if var(D) is true, return D;
   else if ∃D', (D, B, D') ∈ R, return D'; ..... (1)b
   D' := false;
   if var(D) is like x − x' and D = (x − x', ((∼_1, c_1), D_1), ..., ((∼_n, c_n), D_n)), {
      if x = MID and x' = RIGHT, then
         for i := 1 to n, { B := β_i; D' := D' ∪ (x − x' ∼_i c_i ⊓ recBypassRight(D_i)); }
      else if x ≠ RIGHT and x' = MID,
         for i := 1 to n, { (∼, c) := B + β_i; D' := D' ∪ (x − x' ∼ c ⊓ recBypassRight(D_i)); }
      else for i := 1 to n, D' := D' ∪ (x − x' ∼ c ⊓ recBypassRight(D_i));
   }
   else if var(D) is a discrete variable like v and D = (v, ((l_1, u_1), D_1), ..., ((l_n, u_n), D_n)), then
      D' := ⋃_{1≤i≤n}(l_i ≤ v ≤ u_i ⊓ recBypassRight(D_i));
   R := R ∪ {(D, B, D')}; ..... (2)b
   return D';
}
```

Note that in lines (1)a, we test if the same computation has been done before. The computation can be identified by the current CRD and the bound on the arc LEFT − MID. If it has been done before, then we can save the computation power and return the recorded result value. If it has not been done before, then we will have to do it and record the result value in line (2)a. Similar remarks for line (1)b and (2)b.

E Definitions of the three evaluation orderings

To avoid confusion, we shall carefully distinguish between the declared variables (DV) in automata and the evaluation variables (EV) in the corresponding CRD+BDDs. Here, we consider the evaluation ordering of EVs of systems with concurrent processes with local and global DVs. For non-clock DVs, the same variable names are used as the EV names in the CRD+BDD. For clock DVs $x, x' \in \{0\} \cup X$, the relevant EVs in the CRD+BDD are *clock difference EVs* with names like $x - x'$. Given a timed automaton A, we let Δ_A be the set of non-clock EVs and Γ_A be the set of clock difference EVs.

Let us define the following notations before we present our schemes of variable ordering. For convenience of discussion, we shall have the following notations. The input language of our implementation allows the declaration of processes with corresponding local variables. The processes are labeled with process identifier $1 \ldots m$. For each local nonclock DV u (or local clock DV x), we let $\mathrm{proc}(u)$ (or $\mathrm{proc}(x)$ respectively) be the identifier of the process to which u (or x) is local. For each global nonclock variable u (or global clock x), we let $\mathrm{proc}(u) = 0$ (or $\mathrm{proc}(x) = 0$). Also $\mathrm{proc}(0) = 0$. Given a clock difference EV $x - x'$ in a CRD+BDD, we let $\mathrm{proc}(x - x') = \max(\mathrm{proc}(x), \mathrm{proc}(x'))$. We here extend the meaning of evaluation index $\Omega()$ in section 4 to both clock difference EVs and discrete EVs. For any two EVs $w, w' \in \Delta_A \cup \Gamma_A$, we write $w \prec_\Omega w'$ to denote that w precedes w' in evaluation ordering Ω.

According to the ordering a DV is declared in the input, we also define attribute offset(). Given two DV u, u', $\mathrm{offset}(u) < \mathrm{offset}(u')$ iff u is declared before u' in the input file. Note that two distinct DVs u, u' may be declared with the same local name. Given a clock difference EV $x - x'$, we let $\mathrm{offset}(x - x') = \max(\mathrm{offset}(x), \mathrm{offset}(x'))$.

We shall experiment with three variable-ordering: NIL (with no interleaving between discrete EVs and clock EVs), HIL (with half interleaving between discrete EVs and clock EVs), and FIL (with full interleaving between discrete EVs and clock EVs). We have general rules and special rules for the decision of precedence relation among EVs.

General rules

There are two general rules applied to all these three ordering.

- For every two distinct clocks x, x', we put clock difference variable like $x' - x$ immediately after $x - x'$ in the variable ordering. This arrangement allows us to efficiently check for some trivial negative cycles.
- Given two EVs w, w', if the precedence ordering cannot be determined by the following special rules, then $w \prec_\Omega w'$ iff $\mathrm{offset}(w) < \mathrm{offset}(w')$ for all $\Omega \in \{\mathrm{NIL}, \mathrm{HIL}, \mathrm{FIL}\}$.

Special rules

Given a clock difference EV $x - x'$ in a CRD+BDD, we let $\mathrm{proc}_{\min}(x - x') = \min(\mathrm{proc}(x), \mathrm{proc}(x'))$. Also, $\mathrm{offset}_{\min}(x - x') = \min(\mathrm{offset}(x), \mathrm{offset}(x'))$. For discrete variables w, $\mathrm{proc}_{\min}(w) = \mathrm{proc}(w)$ and $\mathrm{offset}_{\min}(w) = \mathrm{offset}(w)$. Given two EVs v and v', we write $v \sqsubset v'$ iff one of the following four condition is true: (1) $\mathrm{proc}(v) < \mathrm{proc}(v')$, (2) $\mathrm{proc}(v) = \mathrm{proc}(v') \wedge \mathrm{proc}_{\min}(v) < \mathrm{proc}_{\min}(v')$, (3) $\mathrm{proc}(v) = \mathrm{proc}(v') \wedge \mathrm{proc}_{\min}(v) = \mathrm{proc}_{\min}(v') \wedge \mathrm{offset}(v) < \mathrm{offset}(v')$, or (4) $\mathrm{proc}(v) = \mathrm{proc}(v') \wedge \mathrm{proc}_{\min}(v) = \mathrm{proc}_{\min}(v') \wedge \mathrm{offset}(v) = \mathrm{offset}(v') \wedge \mathrm{offset}_{\min}(v) < \mathrm{offset}_{\min}(v')$. Intuitively $v \sqsubset v'$ means that v precedes v' according to the process ordering and the syntax ordering.

The special rules for the three orderings are as follows.

NIL: (1) $\forall u \in \Delta_A \forall x - x' \in \Gamma_A (u \prec_{NIL} x - x')$, that is, all discrete EVs precede those clock difference EVs.
(2) $\forall x - x', y - y' \in \Gamma_A (x - x' \prec_{NIL} y - y'$ iff $x - x' \sqsubset y - y')$.

HIL: (1) $\forall u \in \Delta_A \forall x - x' \in \Gamma_A$ s.t. $x = 0$ or $x' = 0 (u \prec_{HIL} x - x'$ iff $\mathrm{proc}(u) \leq \mathrm{proc}(x - x'))$.
(2) $\forall x - 0, 0 - x, y - y' \in \Gamma_A$ s.t. $0 \notin \{y, y'\} (x - 0 \prec_{HIL} y - y' \wedge 0 - x \prec_{HIL} y - y')$.
(3) $\forall x - x', y - y' \in \Gamma_A$ s.t. $0 \notin \{x, x', y, y'\} (x - x' \prec_{HIL} y - y'$ iff $x - x' \sqsubset y - y')$.

FIL: (1) $\forall u \in \Delta_A \forall x - x' \in \Gamma_A (u \prec_{FIL} x - x'$ iff $\mathrm{proc}(u) \leq \mathrm{proc}(x - x'))$.
(2) $\forall x - x', y - y' \in \Gamma_A (x - x' \prec_{FIL} y - y'$ iff $x - x' \sqsubset y - y')$.

E.1 Performance w.r.t. variable-ordering

We compare the performance of `red` with MD&DRE techniques w.r.t. the three variable-ordering discussed in section 7. The benchmarks used are the same as the six used in the last subsection. The performance data is in table 5 and is very compatible with the traditional experience about BDD-like data-structures. That is, variables with strong relation should be placed near to each other. Ordering NIL does not abide by this experience and places all clock constraints together. Ordering HIL only places magnitude constraints close to their corresponding local discrete variables. Ordering FIL is the most compatible with the experience and places local cock constraints like $x_1 - x_2 \sim c$ close to their corresponding local discrete variables of process $\max(\mathrm{proc}(x_1), \mathrm{proc}(x_2))$.

benchmarks	concurrency	-Ob	-Oh	no
Fischer's mutual exclusion cline2-5	3 processes	0.21s/18k	0.21s/19k	0.2s/18k
	4 processes	1.50s/55k	1.30s/57k	1.23s/43k
	5 processes	7.72s/150k	5.67s/174k	5.03s/94k
	6 processes	40.60s/438k	22.87s/495k	15.99s/191k
	7 processes	199.61s/1255k	91.98s/1348k	47.46s/384k
	8 processes	840.44s/3542k	383.57s/3518k	139.98s/851k
	9 processes	2812.61s/9971k	1362.88s/8897k	417.00s/2032k
	10 processes	9299.48s/28271k	O/M	1160.17s/4736k
	11 processes	O/M	O/M	3178.93s/10897k
	12 processes	O/M	O/M	8684.63s/24802k
CSMA/CD	bus+3 senders	0.10s/98k	0.10s/103k	0.09s/103k
	bus+4 senders	0.32s/180k	0.25s/181k	0.20s/181k
	bus+5 senders	0.97s/312k	0.61s/292k	0.41s/292k
	bus+6 senders	3.70s/783k	1.68s/724k	0.84s/458k
	bus+7 senders	12.61s/2039k	5.78s/2021k	1.88s/723k
	bus+8 senders	38.06s/5267k	17.19s/5550k	4.65s/1681k
	bus+9 senders	113.00s/13536k	52.87s/15500k	12.45s/3971k
	bus+10 senders	318.80s/34637k	194.47s/42439k	35.86s/9284k
	bus+11 senders	988.90s/91881k	1062.61s/113657k	115.14s/21470k
	bus+12 senders	O/M	O/M	424.69s/49198k
	bus+13 senders	O/M	O/M	1892.97s/111699k
FDDI token-ring passing	11 stations	258.78s/34202k	1.19s/345k	1.19s/345k
	12 stations	O/M	1.85s/456k	1.79s/456k
	20 stations	O/M	16.70s/1311k	16.59s/1311k
	30 stations	O/M	81.96s/2893k	81.95s/2893k
	40 stations	O/M	264.86s/5962k	266.85s/5962k
	50 stations	O/M	705.46s/9902k	652.85s/9903k
	60 stations	O/M	N/A	1263.30s/14018k
pathos	3 processes	0.06s/26k	0.06s/25k	0.07s/25k
	4 processes	0.28s/54k	0.31s/52k	0.28s/52k
	5 processes	1.18s/103k	1.17s/116k	1.06s/89k
	6 processes	5.94s/268k	5.27s/394k	3.97s/218k
	7 processes	36.98s/872k	32.21s/1478k	21.12s/636k
	8 processes	198.62s/2944k	182.39s/5762k	109.25s/1972k
	9 processes	892s/10273k	999s/22245k	508.73s/6267k
	10 processes	3532.70s/35435k	5177.02s/85218k	2259.6s/20241k
	11 processes	19430.84s/122775k	O/M	88975.46s/65243k
leader	3 processes	0.04s/46k	0.04s/46k	0.05s/46k
	4 processes	0.14s/98k	0.14s/94k	0.15s/94k
	5 processes	0.45s/197k	0.43s/165k	0.42s/165k
	6 processes	2.25s/476k	1.18s/263k	1.28s/263k
	7 processes	9.65s/1138k	3.45s/479k	3.54s/479k
	8 processes	34.49s/2767k	8.68s/819k	8.82s/819k
	9 processes	112.85s/6790k	19.17s/1366k	19.21s/1366k
	10 processes	357.17s/16888k	37.86s/2130k	38.14s/2130k
	11 processes	1136.94s/42108k	69.62s/3280k	70.32s/3280k
	12 processes	4003.46s/106543k	121.43s/4806k	124.16s/4806k
	13 processes	O/M	208.14s/6946k	207.84s/6946k
	14 processes	O/M	335.88s/9683k	340.13s/9683k
lbound	3 processes	0.18s/51k	0.25s/57k	0.17s/45k
	4 processes	1.41s/198k	2.26s/292k	1.24s/172k
	5 processes	12.95s/816k	19.50s/1540k	10.78s/846k
	6 processes	66.04s/3167k	103.92s/6894k	78.24s/3312k
	7 processes	500.38s/12206k	1018.21s/28552k	458.45s/11693k
	8 processes	2130.12s/45965k	6039.28s/117430k	2755.94s/37977k
	9 processes	O/M	O/M	19288.73s/114418k

data collected on a Pentium 4 1.7GHz with 256MB memory running LINUX;
s: seconds; k: kilobytes of memory in data-structure; O/M: Out of memory; N/A: not available;
v

Table 5: Performance data w.r.t. variable-ordering