

Efficient verification of timed automata with BDD-like data structures

Farn Wang

Department of Electrical Engineering, National Taiwan University, 1, Sec. 4, Roosevelt Rd., Taipei, Taiwan 106, R.O.C.
e-mail: farn@cc.ee.ntu.edu.tw

Published online: 16 April 2004 – © Springer-Verlag 2004

Abstract. We investigate the effect on efficiency of various design issues for BDD-like data structures of TA state space representation and manipulation. We find that the efficiency is highly sensitive to decision atom design and canonical form definition. We explore the two issues in detail and propose to use CRD (Clock-Restriction Diagram) for TA state space representation and present algorithms for manipulating CRD in the verification of TAs. We compare three canonical forms for zones, develop a procedure for quick zone-containment detection, and present algorithms for verification with backward reachability analysis. Three possible evaluation orderings are also considered and discussed. We implement our idea in our tool **Red** 4.2 and carry out experiments to compare with other tools and various strategies of **Red** in both forward and backward analysis. Finally, we discuss the possibility of future improvement.

Keywords: Data structures – BDD – Timed automata – Verification – Model checking

1 Introduction

Data structures are the groundwork for efficient algorithms, especially for high-complexity tasks like real-time system model checking [2]. Most modern model checkers

for real-time systems are built around symbolic manipulation procedures [11] of *zones*, which means a behaviorally equivalent convex state space of a timed automaton (TA). DBM (difference-bounded matrix) [9] has been generally considered the most efficient data structure in representing sets of zones. But a DBM can represent only a convex state space and can incur inefficiency when doing so.

In recent years, researchers have been trying to duplicate the success of BDD techniques in hardware verification for the verification of TAs [1, 7, 13, 14, 16, 17, 22]. 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, 13, 14, 16, 17, 22] have not performed as well as the popular DBM [9], which is a two-dimensional matrix and nothing BDD-like.

Let us first clarify an issue for convenience and precision of discussion. In BDD, the variables used in system descriptions (*system variables*) are directly used as decision variables. But in BDD-like data structures for dense-time state space [1, 7, 13, 14, 16, 17, 22], system variables may be different from those decision variables in the corresponding BDD-like data structures. For example, in CDD [7], system variables are clocks while decision variables are clock differences like $x - x'$. In this paper, we shall call the decision variables used in BDD-like data structures the *decision atoms*.

Having examined earlier BDD-like data structures, we believe that the efficiencies for the representation and manipulation of state spaces with such data structures are very sensitive to the following two basic issues.

- *The design of the decision atoms*, that is, the domains and the semantics of the decision atoms. Previous researchers did not investigate the relation between data

This work is partially supported by NSC, Taiwan, ROC under Grants NSC 90-2213-E-002-131 and NSC 90-2213-E-002-132 and by a grant from the Institute of Applied Science and Engineering Research, Academia Sinica, Taiwan, ROC.

A preliminary report on this work was published in the Proceedings of FORTE'2001 (Kluwer), RT-TOOLS 2001, and APLAS 2000 and is to appear in the Proceedings of VMCAI 2003 (Lecture Notes in Computer Science, Springer).

structure definitions and their representation/manipulation complexities [1, 7, 13, 14, 16, 17, 22]. We have identified the *representation fragmentation phenomenon*, which is caused by the semantics of decision atoms [7], in Sect. 5 and shown that the phenomenon can indeed affect the verification performance against several independently developed benchmarks in experiments.

- *The definition of canonical forms.* A convex space can be represented by more than one zone. 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. According to our observation, for BDD-like data structures, the more constraints are used in representing a chosen zone, the more space complexity is incurred and the less data sharing is possible. But on the other hand, too many constraints omitted in a zone may make it difficult to efficiently decide the containment relation between zones. Thus one of our goals is to research the possibility that through the design of new canonical forms, the appropriate balance between the width (number of paths) and depth (length of the longest path) of BDD-like data structures can be obtained.

Without a proper treatment of these issues, it is not possible to take full 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.

After going over the basic definitions of real-time system verifications and zones in Sects. 2 and 3, we shall first introduce our new BDD-like data structure, *CRD* (*Clock-Restriction Diagram*), and its operations in Sect. 4, for the convenience of discussion of the two issues. CRD shares the same shape as CDD [7], with the major difference that decision atom values in CDD are disjoint intervals while decision atom values in CRD are upperbounds, which are overlapping. For example, the CRD for the union of two zones $\{0 - x_1 \leq -3, x_2 - x_1 < -4, x_1 - x_3 < 6\}$ and $\{0 -$

$x_2 < -1, x_1 - x_3 < 6\}$ (constraints of the form $x - x' < \infty$ are omitted) is shown in Fig. 1a. If we change the upperbounds to interval representations, we get the structure in Fig. 1b, 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 Fig. 1c, has both greater depth and greater width than Figs. 1a and b. Note the CDD in Fig. 1c adheres to the CDD restriction that only decision atoms $x_i - x_j$ with $i < j$ are used. This means that each CDD decision atom value serves both as lowerbound and upperbound.

In Sect. 5, we shall discuss the issue of decision atom design and use CRD to compare it with another data structure. In particular, we shall illustrate with examples to show why subtle differences may incur significant performance differences in the manipulation and representation of state spaces. The section also serves as a short survey to compare with previous data structures.

In Sect. 6, we introduce a condition for the efficient detection of zone containment in a CRD, present an algorithm to eliminate contained zones, and discuss how to use the algorithm flexibly for better performance.

In Sect. 7, we discuss the issue of canonical form choice in depth. We have carried out numerous experiments with other tools (e.g., UPPAAL2k [6, 15] and Kronos [5, 10, 25]) and various canonical forms with some possible canonical form computation algorithms in order to gain better understanding of the issues. In the end, we propose a new canonical form [*difference-reduced closure (DRC) form*] that helps us strike a balance between the depth and width of CRDs and performs better than closure forms and reduced forms against many benchmarks. Algorithms to compute closure form and DRC form CRDs are also presented.

We also present algorithms for calculating the weakest precondition of discrete transitions and timed progress with CRDs. In particular, in the case of timed progress our algorithm does not rely on the δ variables (used in [11]) and is much more simplified. Such simplification is crucial for BDD-like data structures since with variable δ numerous new decision atoms have to be introduced in the intermediate representation and are likely to blow up the memory usage.

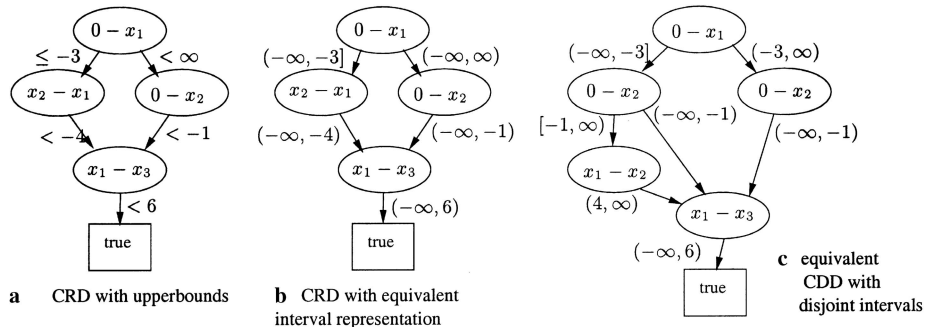


Fig. 1. Differences between CRD and CDD

In Sect. 10, we report our implementation and experiments to observe CRD’s performance with respect to various strategies, including three canonical forms, three evaluation orderings, backward/forward analysis, etc., and other tools. Recent achievements of our implementation, **Red**, include numerical coverage estimation in symbolic simulation of dense-time systems [24], speedup techniques for greatest fixpoint evaluation [23], and BDD-like data structures for linear hybrid automata [20]. Finally, in Sect. 11 we summarize this work and discuss possibilities for more performance enhancement.

2 Timed automata verification

Our system model is *timed automaton (TA)* [2], which is a finite-state automaton equipped with a finite set of clocks that can hold nonnegative real values. At any moment, the TA 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. 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 $P(Q, X)$ as the set of all Boolean combinations of atoms of the forms q and $x \sim c$, where $q \in Q$, $x \in X$, “ \sim ” is one of $\leq, <, =, >, \geq$, and c is an integer constant.

Definition 1. *Timed automata (TA)* A TA A is 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, $I \in P(Q, X)$ is the initial condition, $\mu : Q \mapsto P(\emptyset, X)$ defines the invariance condition of each mode, $T \subseteq Q \times Q$ is the set of transitions, and $\tau : T \mapsto P(\emptyset, X)$ and $\pi : T \mapsto 2^X$ define, respectively, the triggering condition and the clock set to reset each transition. ■

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

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

- There is a unique $q \in Q$ such that $\nu(q) = \text{true}$ and for all $q' \neq q$, $\nu(q') = \text{false}$;
- For each $x \in X$, $\nu(x) \in \mathcal{R}^+$ (the set of nonnegative reals) and $\forall q \in Q$, $\nu(q) \Rightarrow \nu \models \mu(q)$.

Given state ν and $q \in Q$ such that $\nu(q) = \text{true}$, we call q the mode of ν , in symbols ν^Q . ■

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 TA $A = \langle X, Q, I, \mu, T, \tau, \pi \rangle$, a run is an infinite sequence of state-time pairs $(\nu_0, t_0) (\nu_1, t_1) \dots (\nu_k, t_k) \dots$ such that $\nu_0 \models I$ and $t_0 t_1 \dots t_k \dots$ is a monotonically increasing real-number (time) divergent sequence, and for all $k \geq 0$,

- Invariance conditions are preserved in each interval, that is, for all $t \in [0, t_{k+1} - t_k]$, $\nu_k + t \models \mu(\nu_k^Q)$; and
- Either no transition happens at time t_k , that is, $\nu_k^Q = \nu_{k+1}^Q$ and $\nu_k + (t_{k+1} - t_k) = \nu_{k+1}$, or a transition happens at t_k , that is,
 - There is a transition $(\nu_k^Q, \nu_{k+1}^Q) \in T$; and
 - The corresponding transition is enabled, that is, $\nu_k + (t_{k+1} - t_k) \models \tau(\nu_k^Q, \nu_{k+1}^Q)$; and
 - The clocks are reset to zeros accordingly, that is, $(\nu_k + (t_{k+1} - t_k))\pi(\nu_k^Q, \nu_{k+1}^Q) = \nu_{k+1}$. ■

We can define the TCTL model checking problem [2] of TAs as our verification framework. Since the focus of this work is on data structures, for simplicity and conciseness we here adopt the safety analysis problem as our verification framework. A safety analysis problem instance, $SA(A, \eta)$ in symbols, consists of a TA A and a safety state predicate $\eta \in P(Q, X)$. A is safe with respect to η , in symbols $A \models \eta$, iff for all runs $(\nu_0, t_0)(\nu_1, t_1) \dots (\nu_k, t_k) \dots$ such that $\nu_0 \models I$, 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 Reviews on zones and their normal forms

Most modern model checkers are built around some symbolic manipulation procedures [11] of *zones* implemented in various data structures [1, 7, 9, 13, 14, 16, 17]. A zone is symbolically represented by a set of difference constraints between clock pairs.

3.1 Basic notations

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 all $d \in \mathcal{Z}$, let $d + \infty = \infty + d = \infty$.

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

The following notations help us explain the strictness of timing constraints. Given $(\sim_1, d_1), (\sim_2, d_2)$, 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 $d_1 < d_2 \vee (d_1 = d_2 \wedge (\sim_2 = < \Rightarrow \sim_1 = <))$. The following shorthand notations are also adopted: $(\sim_1, d_1) \sqsubset (\sim_2, d_2) \equiv (\sim_1, d_1) \sqsubseteq (\sim_2, d_2) \wedge (\sim_2,$

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

We also need the following notations to discuss the effects of transitivity of constraints. Given two $(\sim_1, c_1), (\sim_2, c_2) \in \mathcal{B}_{C_A;\eta}$, we define $(\sim, c) = (\sim_1, c_1) + (\sim_2, c_2)$:

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

3.2 Canonical forms

Many zones may represent the same convex subspace. Without proper management we can record many zones representing the same convex subspace and easily blow up the memory space. A straightforward *canonical* representation of a zone-characterizable convex subspace is its zone in *closure form* (called tight form in [9] and shortest-path closure in [12]). A zone ζ is in closure form if and only if all its constraint bounds are tight, i.e., for all sequences of elements $x_1, \dots, x_k \in X \cup \{0\}$, $\zeta(x_1, x_k) \sqsubseteq \sum_{1 \leq i < k} \zeta(x_i, x_{i+1})$, that is, $\zeta(x_1, x_k)$ is no less strict than $\sum_{1 \leq i < k} \zeta(x_i, x_{i+1})$. For convenience, given a zone ζ , we let ζ^C be its closure form.

Another candidate for the canonical representation of zones is the *reduced form* (called shortest-path reduction in [12]), which records only the minimum number of constraints for each zone according to some policy. A zone ζ is in its reduced form if each constraint $\zeta(x, x') \neq (<, \infty)$ is not derivable from other constraints in ζ . In other words, ζ is in reduced form if the corresponding state space can no longer be represented if any nontrivial constraint is omitted. The reduced form of zones stands at the other opposite extreme from the closure form, which always has the largest number of nontrivial constraints. We introduced the reduced form in this paper for performance comparison. We refer the interested reader to [12, 18] for an explanation of how to convert a given zone ζ to its zone in reduced form, in symbols ζ^R . It is shown in [12] that $\zeta^C = (\zeta^R)^C$ and zones in reduced form can significantly save space in verification. However, reduced forms are not unique and can be defined with various policies in choosing the representative for clocks with the same readings and in traversing a simple cycle.

4 Clock-Restriction Diagram

CRD [18, 19] is not a decision diagram for state space membership. Instead, it is like a decision diagram for zone set membership. Each *decision atom* in a CRD is of the form $x - x'$, where x, x' are zeros or clocks, and the values of such decision atoms range over $\mathcal{B}_{C_A;\eta}$. Thus a value,

say, (≤ 5) , of decision atom $x - x'$ describes the constraint of half-space $x - x' \leq 5$. A path from the root to the only leaf node *true* in CRD represents a zone. A CRD represents the set of all states in the zones corresponding to each of its paths. In CRD, a missing constraint on the difference of a clock pair, say, x, x' , is interpreted as $x - x' < \infty$. From the root node in Fig. 1a, even if no constraint is on $0 - x_1$ in the zone of the right path, we still construct an arc with $0 - x_1 < \infty$.

4.1 Definitions

By fixing an evaluation ordering, we can construct a CRD such as BDD, CDD, or RED. Given a set V of decision atoms with $true \in V$, an *evaluation index* Ω over V is a 1-to-1 onto mapping from V to $\{0, 1, \dots, |V| - 1\}$ such that $\Omega(true) = |V| - 1$. 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)* Suppose we are given a set of decision atoms $\bar{V} = \{x - x' \mid x, x' \in X \cup \{0\}\}$, an evaluation index Ω over V , and a timing constant $C_{A;\eta}$. *true* is conveniently represented as $0 - 0$. Then, a CRD over V , Ω , and $C_{A;\eta}$ is a tuple $D = (v, (\beta_1, D_1), \dots, (\beta_n, D_n))$ with $n \geq 0$. The restrictions are that $v \in V$ such that

- $v = true$ iff $n = 0$;
- If $v \neq true$, then for all $1 \leq i \leq n$, $\beta_i \in \mathcal{B}_{C_A;\eta}$ and D_i is a CRD like (v_i, \dots) with $v \prec_{\Omega} v_i$;
- If $v \neq true$, then for all $1 \leq i < j \leq n$, $\beta_i \sqsubset \beta_j$; and
- If $v \neq true$ and $n = 1$, then $\beta_1 \neq (<, \infty)$.

We use “()” to represent the CRD for false. ■

In our algorithms, false does not participate in comparison of evaluation orderings among decision atoms.

4.2 Basic set-oriented manipulations on CRD

For convenience of discussion, we may represent a CRD 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 algorithm for operator \cup is as follows. The main procedure calls recursive procedure $\text{rec}\cup()$. For convenience of presentation, we may represent a CRD structure like $(x - x', (\beta_1, B_1), \dots, (\beta_n, B_n))$ symbolically as $(x - x', (\beta_i, B_i)_{1 \leq i \leq n})$.

set Ψ ; /* database of already-processed cases */

```

 $\cup(B, D)$  {
  if  $B = false$ , return  $D$ ; else if  $D = false$ , return  $B$ ;
   $\Psi := \emptyset$ ; return  $\text{rec}\cup(B, D)$ ;
}

```

$\text{rec}\cup(B, D)$ where $B = (x_B - x'_B, (\beta_i, B_i)_{1 \leq i \leq n})$, $D =$

```

( $x_D - x'_D, (\alpha_j, D_j)_{1 \leq j \leq m}$ ) {
  if  $B$  is true or  $D$  is true, return true;
  else if  $\exists H, (B, D, H) \in \Psi$ , return  $H$ ; ..... (1)
  else if  $x_B - x'_B \prec_{\Omega} x_D - x'_D$ , construct CRD
     $H := (x_B - x'_B, (\beta_i, \text{rec}\cup(B_i, D))_{1 \leq i \leq n})$ ;
  else if  $x_B - x'_B \succ_{\Omega} x_D - x'_D$ , construct CRD
     $H := (x_D - x'_D, (\alpha_j, \text{rec}\cup(B, D_j))_{1 \leq j \leq m})$ ;
  else {
     $i := n; j := m; H := \text{false}$ ;
    while  $i \geq 1$  and  $j \geq 1$ , do {
      if  $\beta_i = \alpha_j$ , {
         $H := H \cup (x_B - x'_B, (\beta_i, \text{rec}\cup(B_i, D_j)))$ ;
         $i --; j --$ ;
      }
      else if  $\beta_i \sqsubseteq \alpha_j$ ,
        {  $H := H \cup (x_B - x'_B, (\alpha_j, D_j)); j --$ ; }
      else if  $\alpha_j \sqsubseteq \beta_i$ ,
        {  $H := H \cup (x_B - x'_B, (\beta_i, B_i)); i --$ ; }
    }
    if  $i \geq 1, H := H \cup (x_B - x'_B, (\beta_h, B_h)_{1 \leq h \leq i})$ ;
    if  $j \geq 1, H := H \cup (x_B - x'_B, (\alpha_k, D_k)_{1 \leq k \leq j})$ ;
  }
   $\Psi := \Psi \cup \{(B, D, H)\}$ ; return  $H$ ; ..... (2)
}

```

The procedures for \cap and $-$ are all similar. In the algorithm, we use set Ψ to record the pairs of substructures that have been processed in order to take advantage of the data-sharing capability of BDD-like data structures. At line (1), if the pair has been processed, we then directly return the stored result. At line (2), when a new pair is fully processed with result H , we save H in set Ψ for future usage. This technique is used throughout all our procedures. The complexities of these three manipulations can all be done in time $O(|B| \cdot |D|)$.

Given two zones ζ_1 and ζ_2 , $\zeta_1 \wedge \zeta_2$, the space intersection of ζ_1, ζ_2 , is a new zone such that for every x, x' , $\zeta_1 \wedge \zeta_2(x, x') = \zeta_1(x, x')$ if $\zeta_1(x, x') \sqsubseteq \zeta_2(x, x')$; or $\zeta_2(x, x')$ otherwise. Space intersection of two CRDs B and D , in symbols $B \wedge D$, is a new CRD for $\{\zeta_1 \wedge \zeta_2 \mid \zeta_1 \in B; \zeta_2 \in D\}$.

```

set  $\Psi$ ;
 $\wedge(B, D)$  {
  if  $B$  is false or  $D$  is false, return false;
   $\Psi := \emptyset$ ; return  $\text{rec}\wedge(B, D)$ ;
}
 $\text{rec}\wedge(B, D)$  where  $B = (x_B - x'_B, (\beta_i, B_i)_{1 \leq i \leq n})$ ,  $D = (x_D - x'_D, (\alpha_j, D_j)_{1 \leq j \leq m})$  {
  if  $B$  is true, return  $D$ ; else if  $D$  is true, return  $B$ ;
  else if  $\exists H, (B, D, H) \in \Psi$ , return  $H$ ;
  else if  $x_B - x'_B \prec_{\Omega} x_D - x'_D$ , construct CRD
     $H := (x_B - x'_B, (\beta_i, \text{rec}\wedge(B_i, D))_{1 \leq i \leq n})$ ;
  else if  $x_B - x'_B \succ_{\Omega} x_D - x'_D$ , construct CRD
     $H := (x_D - x'_D, (\alpha_j, \text{rec}\wedge(B, D_j))_{1 \leq j \leq m})$ ;
  else {

```

```

 $i := n; j := m; H := \text{false}; \dot{B} := \text{false}; \dot{D} := \text{false}; \dots (3)$ 
while  $i \geq 1$  and  $j \geq 1$ , do {
  if  $\beta_i = \alpha_j$ , {
     $\dot{B} := \dot{B} \cup B_i; \dot{D} := \dot{D} \cup D_j; \dots (4)$ 
     $H := H \cup (x_B - x'_B, (\beta_i, \text{rec}\wedge(B_i, \dot{D})) \cup (\beta_i, \text{rec}\wedge(\dot{B}, D_j)))$ ;
     $i --; j --$ ;
  }
  else if  $\beta_i \sqsubseteq \alpha_j$ , {
    while  $j \geq 1 \wedge \beta_i \sqsubseteq \alpha_j$ , do {  $\dot{D} := \dot{D} \cup D_j; j --$ ; } ..... (5)
     $H := H \cup (x_B - x'_B, (\beta_i, \text{rec}\wedge(B_i, \dot{D})))$ ;
  }
  else if  $\alpha_j \sqsubseteq \beta_i$ , {
    while  $i \geq 1 \wedge \alpha_j \sqsubseteq \beta_i$ , do {  $\dot{B} := \dot{B} \cup B_i; i --$ ; } ..... (6)
     $H := H \cup (x_B - x'_B, (\alpha_j, \text{rec}\wedge(\dot{B}, D_j)))$ ;
  }
  if  $i \geq 1, H := H \cup (x_B - x'_B, (\beta_h, \text{rec}\wedge(B_h, \dot{D}))_{1 \leq h \leq i})$ ;
  if  $j \geq 1, H := H \cup (x_B - x'_B, (\alpha_k, \text{rec}\wedge(\dot{B}, D_k))_{1 \leq k \leq j})$ ;
}
 $\Psi := \Psi \cup \{(B, D, H)\}$ ; return  $H$ ;
}

```

In particular, in lines (4), (5), and (6), we use auxiliary variable \dot{B}, \dot{D} to record the accumulative spaces that have to intersect with D_j and B_i , respectively. Our algorithm is in time $O(|B|^2 \cdot |D|^2)$.

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 a FALSE terminal node that is more space efficient than ordinary BDD. There are two things we need to take care of in this combination. The first concerns the interpretation of default values of decision atoms. In BDD, when we find a decision atom is missing while evaluating decision atoms along a path, the decision atom's value can be interpreted as either true or false. But in CRD, when we find a decision atom $x - x'$ is missing along a path, the decision atom is interpreted as $x - x' < \infty$.

The second concerns the interpretation of CRD manipulations to BDD decision atoms. Straightforwardly, “ \cup ” and “ \cap ” on BDD decision atoms are respectively interpreted as “ \vee ” and “ \wedge ” on BDD decision atoms. $B - D$ on BDD decision atoms is interpreted as $B \wedge \neg D$ when the root decision atom of either B or D is Boolean. For $B \wedge D$, the manipulation is just like the Boolean conjunction “ \wedge ”.

From now on, we shall call CRD+BDD a combination structure of CRD and BDD.

5 Comparison of designs of data structures with previous research

In the design of BDD-like data structures, two aspects need to be considered, i.e., the domain of decision atoms

and the semantics of decision atom values. Since BDD-like data structures exhibit exponential blowup with respect to the size of the decision atom domain, in general it is good to keep the decision atom domain small. The semantics of decision atom values is about how to interpret the values of decision atoms and has a very subtle effect on space complexity. We believe that it will help the reader to understand this issue if we compare CRD with earlier data structures. DBM technology [9] 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.

As far as we know, the first paper that discusses how to use BDD to encode zones (actually for asynchronous systems with clock jitters) was by Wang et al. in 1993 [22]. They discussed how to use BDD with decision atoms like $x_i + c \leq x_j + d$ to model check timed automata. Here c and d are timing constants with magnitude $\leq C_A$. Each decision atom can assume a Boolean truth value. The approach may suffer from bad performance since the size of the decision atom domain is already proportional to the timing constants and thus exponential to the input size. However, the researchers did not report implementation and experiments. In 1996, Balarin implemented the same scheme and reported experiments with approximation techniques [3]. In 1999, Moller et al. used the same idea to devise a data structure called DDD and discussed many manipulation techniques [13, 14].

In 1997, NDD [1] used binary encoding for clock readings and its performance was very sensitive to timing-constant magnitude.

RED [16, 17] encodes the ordering of fractional parts of clock readings in the evaluation ordering and has achieved very high space efficiency for symmetric systems with large numbers of clocks and small timing constants. RED is indeed a canonical representation of TA 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 structure very similar to that of 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 slight difference, for some state spaces, CDD may demand an exponential memory size. For example, we have the following state space for n clocks:

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

Here “%” represents the modulo operator. When clock count n is 2, the compositions of the state spaces in CDD and in CRD are as in Fig. 2a and b, respectively. As shown in the figure, the CDD union operation produces a CDD of three paths out of two zones, while the CRD union operation basically maintains the structure of the component zones. In fact, our experiment shows that the state space of (1) exhibits an exponential blowup in CDD representations with respect to clock counts in Table 1. Such exponential blowup, we believe, is due to the “disjoint” requirements of CDD on intervals. Although the requirement makes sense in mathematics, it actually contradicts

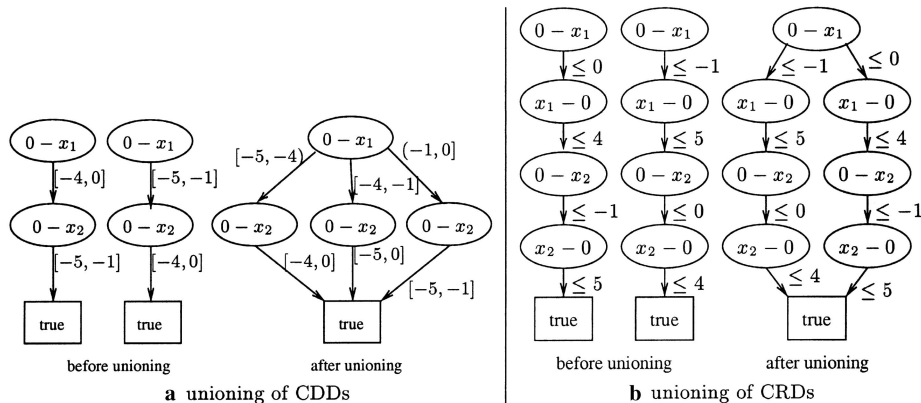


Fig. 2. Comparison between decision atom semantics of CDD and CRD

Table 1. Performance comparison between CDD and CRD with respect to a benchmark family

	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

the characteristics of zones, which are noncanonical representations of convex state spaces and may intersect each other. Thus when union operation is performed, intervals will intersect each other into fragments. Such a 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 destroy previous efforts in state space analysis.

To justify our argument, we have also endeavored to implement some CDD manipulation procedures and have carried out an experiment to observe the effect of the *representation fragmentation phenomenon* on representation complexity. The experiment is reported in Sect. 10.4 and indeed confirms the effect with respect to independently developed benchmarks.

6 Efficient detection of contained zones

Another problematic characteristic of zones is that they can contain one another, and different decision paths in the BDD extensions may contain one another. Without the capability to efficiently detect zone-containment relations, fixpoint algorithms may waste time and space in iterating through smaller and smaller state spaces that are contained by zones already in the BDD-like data structures. The traditional wisdom of *canonical forms* calculation does not address this issue.

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 ζ_1 and ζ_2 :

$$SCR(\zeta_1, \zeta_2) : \forall x, x' (\zeta_1(x, x') \sqsubseteq \zeta_2(x, x')).$$

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.

We have implemented a procedure `slim(D)` to eliminate all zones contained by other zones in D . The procedure in turn is built upon another diadic procedure `exclude(B, D)`, which in turn eliminates all zones in B that are contained by some zones in D .

```

set  $\Phi$ ;
exclude(B, D) {
  if  $B = false$ , return  $false$ ;
   $\Phi := \emptyset$ ; return rec_exclude(B, D);
}

rec_exclude(B, D) where  $B = (x_B - x'_B, (\beta_i, B_i)_{1 \leq i \leq n})$ ,
 $D = (x_D - x'_D, (\alpha_j, D_j)_{1 \leq j \leq m})$  {
  if  $D$  is true, return  $false$ ; else if  $D$  is false, return  $B$ ;
  else if  $\exists H, (B, D, H) \in \Phi$ , return  $H$ ;

```

```

else if  $x_B - x'_B \prec_{\Omega} x_D - x'_D$ , construct
   $H := (x_B - x'_B, (\beta_1, \text{rec\_exclude}(B_i, D))_{1 \leq i \leq n})$ ;
else if  $x_B - x'_B \succ_{\Omega} x_D - x'_D$ , return  $B$ ;
else {
   $j := m$ ;  $H := false$ ;  $\dot{D} := false$ ;
  for  $i := n$  to 1, do {
    while  $j \geq 1 \wedge \beta_i \sqsubseteq \alpha_j$ , do {  $\dot{D} := \dot{D} \cup D_j$ ;  $j --$ ; }
     $H := H \cup (x_B - x'_B, (\beta_i, \text{rec\_exclude}(B_i, \dot{D})))$ ;
  } }
   $\Phi := \Phi \cup \{(B, D, H)\}$ ; return  $H$ ;
}

```

Then the algorithm for `slim()` is as follows:

```

set  $\Psi$ ;
slim(D) {  $\Psi := \emptyset$ ; return rec_slim(D); }

rec_slim(D) with  $D = (x - x', (\alpha_i, D_i)_{1 \leq i \leq m})$  {
  if  $D$  is true or false, return  $D$ ;
  else if  $\exists H, (D, H) \in \Psi$ , return  $H$ ;
  else {
     $i := m$ ;  $H := false$ ;  $\dot{D} := false$ ;
    for  $i := m$  to 1, do {
       $H := H \cup (x - x', (\alpha_i, \text{exclude}(\text{rec\_slim}(D_i), \dot{D})))$ ;
       $\dot{D} := \dot{D} \cup D_i$ ;
    } }
     $\Psi := \Psi \cup \{(D, H)\}$ ; return  $H$ ;
  }
}

```

However, it bears mentioning that the elimination of contained zones may not result in smaller CRD+BDDs. It may happen that with the omission of contained zones, some part of CRD+BDD structures becomes unsharable. In our implementation, we use the procedure for contained-zone elimination in a flexible way. Every time we apply the procedure, we shall check if the new slim CRD+BDD is indeed smaller than the original one. If it is, then we replace the original one with the new slim one. In this flexible way, we have found performance improvement for many benchmarks. See Sect. 10.2 for our experimental report.

7 Canonical forms for CRD

We have come up with various canonical forms for zones (e.g., [18, 19]). Each canonical form demonstrates a different space complexity and time complexity in its representation and manipulation. For conciseness of presentation, in this paper we only discuss three canonical forms: *closure form*, *reduced form*, and *difference-reduced closure (DRC) form*. We adopt the tradition that a CRD is in its closure (or reduced, or DRC) form if all its zones (represented by root-leaf paths) are in closure (or reduced, or DRC, respectively) form.

Closure form and reduced forms are interesting to experiment with since they stand, as it were, at the two extremes of the spectrum. The closure form tends to make

the deepest CRDs, while reduced forms usually give rise to the widest CRDs. The DRC form is then especially designed to balance between the width and the depth of CRDs. In general, DRC has shown better performance than the others.

7.1 Closure form

Closure form [7, 9] is the most popular canonical form for DBM. It has been a popular choice for many BDD-like data structures as well. A nice property of closure form with $SCR()$ is that it supports fast zone-containment detection.

Lemma 1. *Given two zones ζ, ζ' in their closure forms with $\zeta \subseteq \zeta'$, $SCR(\zeta, \zeta')$ is true.*

Proof. Suppose that $\zeta \subseteq \zeta'$ but $SCR(\zeta, \zeta')$ is false. This means that there are clocks x_1, x_2 such that $\zeta'(x_1, x_2) \sqsubset \zeta(x_1, x_2)$. Since ζ and ζ' are both tight in their closure form zone representations, we know that ζ cannot be contained by ζ' . This is a contradiction, and the lemma is proven. ■

For DBM, the closure form can be computed with an $O(|X|^3)$ all-pair shortest-path algorithm. The simple implementation looks like this:

```
closure( $\zeta$ ) {
  for  $x \in X$ , for  $x_1, x_2 \in X$ ,
    if  $\zeta(x_1, x) + \zeta(x, x_2) \sqsubset \zeta(x_1, x_2)$ ,
       $\zeta(x_1, x_2) := \zeta(x_1, x) + \zeta(x, x_2)$ ;
  return  $\zeta$ ;
}
```

A straightforward implementation of the above procedure with CRD is very inefficient since the decision atoms in DBM are unordered and randomly accessible while the ones in CRD can be expensive to access if decision atom access ordering contradicts the evaluation ordering (Ω). To make it less costly to implement procedure $\text{closure}()$ with CRD, we choose to respect the evaluation ordering in CRDs and rewrite the procedure as follows:

```
closure $_{\prec}$ ( $\zeta$ ) {
  for  $x \in X$ , {
    for  $x_1, x_2 \in X$  such that  $x_1 - x \prec_{\Omega} x - x_2$ ,
      if  $\zeta(x_1, x) + \zeta(x, x_2) \sqsubset \zeta(x_1, x_2)$ ,
         $\zeta(x_1, x_2) := \zeta(x_1, x) + \zeta(x, x_2)$ ;
    for  $x_1, x_2 \in X$  such that  $x_1 - x \succ_{\Omega} x - x_2$ ,
      if  $\zeta(x_1, x) + \zeta(x, x_2) \sqsubset \zeta(x_1, x_2)$ ,
         $\zeta(x_1, x_2) := \zeta(x_1, x) + \zeta(x, x_2)$ ;
  }
  return  $\zeta$ ;
}
```

Another observation is that the **if** statements can be implemented with the intersection operation, i.e., $\zeta :=$

$\zeta \wedge \{x_1 - x_2(\zeta(x_1, x) + \zeta(x, x_2))\}$, where $x_1 - x_2(\sim, c)$ is shorthand for $x_1 - x_2 \sim c$.

For convenience, given $\alpha = (\sim, c)$, we use the special notation $x - x'\alpha$ to represent the CRD of $(x - x', ((\sim, c), true))$ for characterizing subspace $x - x' \sim c$. The algorithm for the procedure with CRDs follows.

```
closure $_{\prec}$ ( $D$ )
  { for  $x \in X$ ,  $D := \text{xtive}(D, x)$ ; return  $D$ ; }

clock LEFT, MID, RIGHT; upperbound  $\beta$ ; set  $\Psi, \Phi$ ;
xtive( $D, x$ )
  {  $\Psi := \emptyset$ ; MID :=  $x$ ; return  $\text{rec\_xtive}(D)$ ; }

rec\_xtive( $D$ ) with  $D = (x - x', (\alpha_i, D_i)_{1 \leq i \leq m})$  {
  if  $D = true$  or  $D = false$ , return  $D$ ;
  else if  $\exists H, (D, H) \in \Psi$ , return  $H$ ;
   $H := false$ ;
  if  $x$  is MID, for  $i := 1$  to  $n$ , {
     $D' := \text{rec\_xtive}(D_i)$ ; RIGHT :=  $x'$ ;  $\beta := \alpha_i$ ;  $\Phi := \emptyset$ ;
     $H := H \cup (x - x'\alpha_i \wedge \text{rec\_xtive\_right}(D'))$ ; ..... (7)
  }
  else if  $x'$  is MID, for  $i := 1$  to  $n$ , {
     $D' := \text{rec\_xtive}(D_i)$ ; LEFT :=  $x$ ;  $\beta := \alpha_i$ ;  $\Phi := \emptyset$ ;
     $H := H \cup (x - x'\alpha_i \wedge \text{rec\_xtive\_left}(D'))$ ; ..... (8)
  }
  else  $H := \bigcup_{1 \leq i \leq n} (x - x'\alpha_i \wedge \text{rec\_xtive}(D_i))$ ;
   $\Psi := \Psi \cup \{(D, H)\}$ ; return  $H$ ;
}

rec\_xtive\_left( $D$ ) with  $D = (x - x', (\alpha_i, D_i)_{1 \leq i \leq m})$  {
  if  $D$  is  $true$  or  $false$ , return  $D$ ;
  else if  $\exists H, (D, H) \in \Phi$ , return  $H$ ;
  if  $x$  is MID,
     $H := \bigcup_{1 \leq i \leq n} \left( x - x'\alpha_i \wedge \text{LEFT} - x'(\beta + \alpha_i) \right)$ ; ... (9)
  else
     $H := \bigcup_{1 \leq i \leq n} (x - x'\alpha_i \wedge \text{rec\_xtive\_left}(D_i))$ ; . (10)
   $\Phi := \Phi \cup \{(D, H)\}$ ; return  $H$ ;
}

rec\_xtive\_right( $D$ ) with  $D = (x - x', (\alpha_i, D_i)_{1 \leq i \leq m})$  {
  if  $D$  is  $true$  or  $false$ , return  $D$ ;
  else if  $\exists H, (D, H) \in \Phi$ , return  $H$ ;
  if  $x'$  is MID,
     $H := \bigcup_{1 \leq i \leq n} \left( x - x'\alpha_i \wedge x - \text{RIGHT}(\beta + \alpha_i) \right)$ ; .. (11)
  else
     $H := \bigcup_{1 \leq i \leq n} (x - x'\alpha_i \wedge \text{rec\_xtive\_right}(D_i))$ ; (12)
   $\Phi := \Phi \cup \{(D, H)\}$ ; return  $H$ ;
}
```

In the outer recursion with procedure $\text{rec_xtive}()$, we traverse through the given CRD to pick up instantiations of decision atom $x_1 - x$. Once such a decision atom has been instantiated, we enter the inner procedure calls with $\text{rec_xtive_left}()$ and $\text{rec_xtive_right}()$ to pick up instantiations of decision atom $x - x_2$.

Note that in lines (7)–(12) we use operator “ \wedge ” instead of notations like $(x - x', (\beta_i, B_i)_{1 \leq i \leq n})$. When we use the latter, we have to make sure that all decision atoms in B_1, \dots, B_n follow $x - x'$. But when deducing new constraints on $x - x''$ transitively from constraints on $x - x'$ and $x' - x''$, respectively, it is not always true that $x - x'' \prec x - x'$ and $x - x'' \prec x' - x''$.

7.2 Reduced form

Since a zone in its reduced form always has the least number of constraints among all representations, a CRD in reduced form (reduced CRD for short) always has the shortest depth among all CRDs for the same set of zones.

Note again that reduced CRD is not a canonical representation of dense-time state space. Nevertheless, on average, reduced CRD can increase the possibility for data sharing to a greater extent than other canonical forms [1, 7, 16, 17]. Here is an example that shows that closure CRD may prevent data sharing. We may have two zones represented by $(x_1 - x_2 < 3 \wedge x_3 - x_1 \leq 6) \vee (x_1 - x_4 \leq 5 \wedge x_3 - x_1 \leq 6)$. The two zones share constraint $x_3 - x_1 \leq 6$. The two zones are already represented in their reduced forms. Their closure forms are $(x_1 - x_2 < 3 \wedge x_3 - x_1 \leq 6 \wedge x_3 - x_2 < 9)$ and $(x_1 - x_4 \leq 5 \wedge x_3 - x_1 \leq 6 \wedge x_3 - x_4 \leq 11)$, respectively. The reduced CRD and closure CRD of the set of the two zones are depicted in Figs. 3a and b. As shown in the figure, because of the evaluation ordering, the closure CRD does not allow more data sharing than the reduced CRD.

However, reduced CRDs can be bad at helping us detect the containment relation between zones (paths in the CRDs) and may hurt in the width of the CRDs. For example, we may have a CRD with the following two zones:

$$(x_1 - x_3 \leq 0 \wedge x_3 - x_2 \leq -2 \wedge x_2 - x_4 \leq -3 \wedge x_4 - x_1 \leq 5) \vee (x_1 - x_2 \leq -2 \wedge x_2 - x_4 \leq -3 \wedge x_4 - x_1 \leq 5).$$

Or equivalently, the two zones can be represented as

$$(x_1 = x_3 \wedge x_1 = x_2 - 2 \wedge x_2 = x_4 - 3) \vee (x_1 = x_2 - 2 \wedge x_2 = x_4 - 3).$$

As can be seen, the zone in the second line is really a superset of the one in the first line. One reduced form of the CRD is shown in Fig. 3c, which has two paths while the closure CRD should have only one path. In general, it is not possible to decide the zone-containment relation without carrying out the $O(|X|^3)$ shortest-path algorithm in some way.

To implement reduced form with CRDs is a very complex task. We shall not present its normalization algorithm with CRD here. The reader is referred to [18, 19] for more information.

7.3 Difference-reduced closure (DRC) form

A clock constraint $x - x' \sim c$ is a *magnitude constraint* if either x or x' is clock zero, i.e., the constant of zero. Magnitude constraints are important because in our experience most of the timing constraints appear in the model descriptions and the specifications as magnitude constraints. A clock constraint that is not a magnitude constraint is called a *difference constraint*. The difference-reduced closure (DRC) form is identical to closure form except that a difference constraint is omitted from a zone representation if the difference constraint is *magnitude redundant*, i.e., it can be derived from two magnitude constraints in the same zone. Apparently, with the DRC form, we may have less constraints in the representation than with closure form. But in the meantime, we still have preserved many magnitude constraints for the efficient decision of invariance conditions, triggering conditions, and specification correctness. But the following lemma shows that it can be difficult to check zone containment with the DRC form.

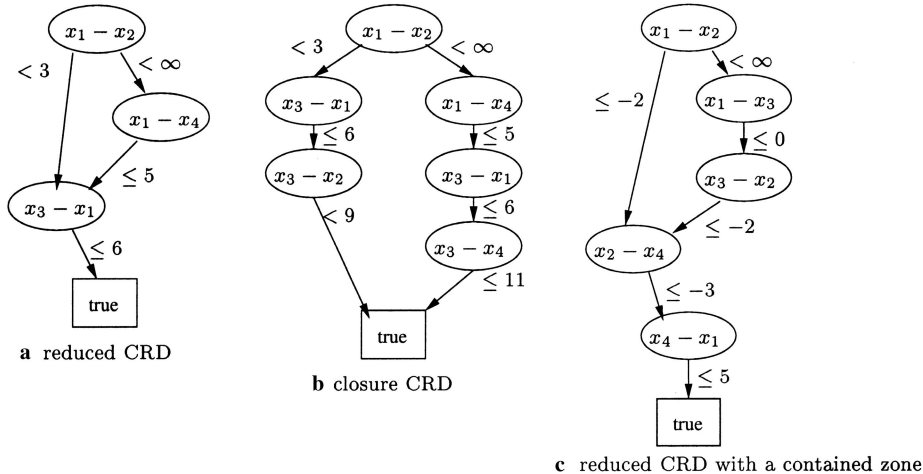


Fig. 3. Comparisons between closure CRD and reduced CRD

Lemma 2. *Given two zones ζ, ζ' in their DRC forms with $\zeta \subseteq \zeta'$, $\text{SCR}(\zeta, \zeta')$ may not be true.*

Proof. Consider $\zeta = \{0 - x_1 \leq 0, x_1 - 0 \leq 3, 0 - x_2 \leq 0\}$ and $\zeta' = \{0 - x_1 \leq 0, 0 - x_2 \leq 0, x_1 - x_2 \leq 3\}$. They are both in the DRC form and $\zeta \subseteq \zeta'$, but they do not satisfy $\text{SCR}(\zeta, \zeta')$. ■

Thus to choose between the closure form and the DRC form becomes a compromise decision between the width and depth of CRDs. In Sect. 10.6, we shall see that in many cases it pays off to use the DRC form. The algorithm of DRC normalization uses procedure $\text{rm_ineq}(D, x - x', \alpha)$ to eliminate identified redundant constraints. For each zone ζ in D , if $\alpha \sqsubseteq \zeta(x, x')$, then the procedure removes constraints for difference $x - x'$ from ζ (or its equivalent, replaced by $x - x' < \infty$). The algorithm is as follows.

```

upperbound  $\beta$ ; clock LEFT, RIGHT; set  $\Psi$ ;
rm_ineq( $D, x - x', \alpha$ ) {
  if  $D$  is false, return  $D$ ;
  LEFT =  $x$ ; RIGHT =  $x'$ ;
   $\beta = \alpha$ ;  $\Psi := \emptyset$ ; return rec_rm_ineq( $D$ );
}

rec_rm_ineq( $D$ ) w.  $D = (x - x', (\alpha_i, D_i)_{1 \leq i \leq m})$  {
  if  $D$  is true or LEFT - RIGHT  $\prec x - x'$ , return  $D$ ;
  else if  $\exists H, (D, H) \in \Psi$ , return  $H$ ;
   $H := \text{false}$ ;
  if  $x$  is LEFT and  $x'$  is RIGHT, for  $i := 1$  to  $n$ , do
    if  $\alpha_i \sqsubseteq \beta$ ,  $H := H \cup (x - x', (\alpha_i, \text{rec\_rm\_ineq}(D_i)))$ ;
    else  $H := H \cup \text{rec\_rm\_ineq}(D_i)$ ;
  else  $H := (x - x', (\alpha_i, \text{rec\_rm\_ineq}(D_i))_{1 \leq i \leq n})$ ;
   $\Psi := \Psi \cup \{(D, H)\}$ ; return  $H$ ;
}

```

To make an efficient implementation of DRC, we need to respect the evaluation ordering in the recursive procedure. Thus, in the evaluation orderings discussed in Sect. 8, we have made sure that for any given clocks $x, x' \in X$, $0 - x' \prec_{\Omega} x - x'$ and $x - 0 \prec_{\Omega} x - x'$. With this restriction, the algorithm of DRC normalization is as follows.

```

clock LEFT, RIGHT; upperbound  $\beta$ ; set  $\Psi, \Phi$ ;
DRC( $D$ ) {
   $D := \text{rec\_DRC}(D)$ ;  $\Psi := \emptyset$ ;
  return rec_DRC(closure_ $\prec$ ( $D$ ));
}

rec_DRC( $D$ ) with  $D = (x - x', (\alpha_i, D_i)_{1 \leq i \leq m})$  {
  if  $D$  is true or false, return  $D$ ;
  else if  $\exists H, (D, H) \in \Psi$ , return  $H$ ;
   $H := \text{false}$ ;
  if  $x$  is 0, for  $i := 1$  to  $n$ , {
     $D' := \text{rec\_DRC}(D_i)$ ; RIGHT :=  $x'$ ;  $\beta := \alpha_i$ ;  $\Phi := \emptyset$ ;
     $H := H \cup (x - x', (\alpha_i, \text{rec\_DRC\_right}(D')))$ ;
  }
}

```

```

else if  $x'$  is 0, for  $i := 1$  to  $n$ , {
   $D' := \text{rec\_DRC}(D_i)$ ; LEFT :=  $x$ ;  $\beta := \alpha_i$ ;  $\Phi := \emptyset$ ;
   $H := H \cup (x - x', (\alpha_i, \text{rec\_DRC\_left}(D')))$ ;
}
else  $H := (x - x', (\alpha_i, \text{rec\_DRC}(D_i))_{1 \leq i \leq n})$ ;
 $\Psi := \Psi \cup \{(D, H)\}$ ; return  $H$ ;
}

rec_DRC_left( $D$ ) with  $D = (x - x', (\alpha_i, D_i)_{1 \leq i \leq m})$  {
  if  $D$  is true, return  $D$ ;
  else if  $\exists H, (D, H) \in \Phi$ , return  $H$ ;
   $H := \text{false}$ ;
  if  $x$  is 0, for  $i := 1$  to  $n$ ,
     $H := H \cup (x - x', (\alpha_i, \text{rm\_ineq}(\text{rec\_DRC\_left}(D_i), \text{LEFT} - x', \beta + \alpha_i)))$ ;
  else  $H := (x - x', (\alpha_i, \text{rec\_DRC\_left}(D_i))_{1 \leq i \leq n})$ ;
   $\Phi := \Phi \cup \{(D, H)\}$ ; return  $H$ ;
}

rec_DRC_right( $D$ ) with  $D = (x - x', (\alpha_i, D_i)_{1 \leq i \leq m})$  {
  if  $D$  is true, return  $D$ ;
  else if  $\exists H, (D, H) \in \Phi$ , return  $H$ ;
   $H := \text{false}$ ;
  if  $x'$  is 0, for  $i := 1$  to  $n$ ,
     $H := H \cup (x - x', (\alpha_i, \text{rm\_ineq}(\text{rec\_DRC\_right}(D_i), x - \text{RIGHT}, \beta + \alpha_i)))$ ;
  else  $H := (x - x', (\alpha_i, \text{rec\_DRC\_right}(D_i))_{1 \leq i \leq n})$ ;
   $\Phi := \Phi \cup \{(D, H)\}$ ; return  $H$ ;
}

```

8 Evaluation ordering in CRD+BDD

The manipulation efficiency of BDD-like data structures is strongly related to decision atom evaluation ordering. Traditional wisdom says that we should place two strongly related decision atoms close to each other in the ordering. We consider the following three possibilities for evaluation ordering to test how CRD+BDD reacts to evaluation orderings.

- **B**, with no interleaving between BDD decision atoms and CRD decision atoms in the ordering.
- **H**, only with interleaving between BDD decision atoms and CRD magnitude decision atoms.
- **F**, with full interleaving between BDD decisions and CRD decision atoms.

When interleaving is implemented, we consider the precedence of process identifiers and system variable declaration ordering to define the interleaving ordering. Our experiments show performance data that are very compatible with the traditional wisdom. That is, **F** is more efficient than **H**, which in turn is more efficient than **B**. The definition of the three orderings is quite tedious. For the sake of presentation flow, we have left the details of the three ordering definitions to the appendix.

9 Safety analysis with CRD

We need two basic procedures, one for the computation of weakest precondition of transitions and the other for that of backward time progression. These two procedures are important in the symbolic construction of backward reachable state space representations. Various presentations of the two procedures can be found in [11, 16–19, 21]. Given a state space representation η and a transition e , the first procedure, `xtion_bck`(η, e), computes the weakest precondition

- in which every state satisfies the invariance condition imposed by $\mu()$ and
- from which we can (forwardly) transit to states in η through e .

The second procedure, `time_bck`(η), computes the space representation of states

- from which we can go to states in η simply by time passage and
- every state in the time passage that also satisfies the invariance condition imposed by $\mu()$.

With the two basic procedures, we can construct a symbolic backward reachability procedure as in [11, 16–19, 21]. Computationally, this backward reachable state space from the goal state η can be defined as the solution Y to the following least fixpoint equation:

$$Y = \eta \cup \text{time_bck} \left(\bigcup_{e \in T} \text{xtion_bck}(Y, e) \right),$$

i.e., $\text{lfp}Y. (\eta \cup \text{time_bck}(\bigcup_{e \in T} \text{xtion_bck}(Y, e)))$. Then the safety analysis problem $\text{SA}(A, \eta)$ can be answered with the unsatisfiability of

$$I \wedge \text{lfp}Y. \left((\neg \eta) \cup \text{time_bck} \left(\bigcup_{e \in T} \text{xtion_bck}(Y, e) \right) \right).$$

Recall that I is the initial condition of the TA in Definition 1. In the following discussion, we shall discuss the algorithm for implementing these two procedures with CRDs.

9.1 `xtion_bck()` for weakest precondition of discrete transitions

Given a discrete transition e from mode q to mode q' such that $\pi(e)$ can be written as a sequence of clock-reset operations to clocks x_1, \dots, x_n , the operation of the transition can be visualized as the following successive steps:

- (1) Invariance condition $\mu(q)$ of the source mode and triggering condition $\tau(e)$ are checked.
- (2) Clocks x_1, \dots, x_n are reset successively.
- (3) Invariance condition $\mu(q')$ of the destination mode is then checked.

To compute the weakest precondition on a state predicate η through this transition, we have to operate on

these steps in a backward fashion. Steps (1) and (3) can be performed with $\eta \wedge \mu(q) \wedge \tau(e)$ and $\eta \wedge \mu(q')$, respectively, and we already know how to perform a \wedge -operation on CRD+BDD.

Step (2) can be performed with a sequence of weakest precondition analysis for the individual reset operations. For reset operation $x := 0$, the weakest precondition algorithm follows.

```
reset_clock(D, x)
{ return  rm_clock(xtive(D ∧ x = 0, x), x); }
```

$D \wedge x = 0$ is for the enforcing of the postcondition after the reset operation. Procedure `rm_clock()` removes every inequality related to clock x from its argument state predicate since there is no restriction on the value of clock x before the reset operation just by looking at this individual reset operation. However, before the removal of all inequalities related to clock x , we must make sure that all other constraints deducible from constraints on x will be preserved with procedure `xtive`(D, x).

According to the explanation above, we now need to design algorithms for procedure `rm_clock()`. Geometrically, given a state predicate η , `rm_clock`(η, x_i) is the projection of η on the space of $X - \{x_i\}$. The algorithm for the procedure with CRDs is as follows.

```
set Ψ;
rm_clock(D, x_i)
{ Ψ := ∅; MID := x_i; return rec_rm_clock(D); }
rec_rm_clock(D) with D = (x - x', (α_i, D_i)_{1 ≤ i ≤ m}) {
  if D is true or false, return D;
  else if ∃H, (D, H) ∈ Ψ, return H;
  else if either x or x' is MID,
    H := ⋃_{1 ≤ i ≤ n} rec_rm_clock(D_i);
  else H := (x - x', (α_i, rec_rm_clock(D_i))_{1 ≤ i ≤ n});
  Ψ := Ψ ∪ {(D, H)}; return H;
}
```

9.2 `time_bck()` for weakest precondition of time progress

The important technique presented in [11] for the calculation of timed weakest precondition utilizes an existentially quantified dense variable $\delta \geq 0$ to reason with the effect of time progress. For example, we may have $\eta = x_1 \leq 3 \wedge x_2 > 4 \wedge x_3 > 1 \wedge x_1 - x_2 \leq 6$. The weakest timed precondition from η characterizes the state space in which a state can progress to a state in η without discrete transitions. Such states can be characterized by the following formulas:

$$x_1 \geq 0 \wedge x_2 \geq 0 \wedge x_3 \geq 0 \\ \wedge \exists \delta \left(\delta \geq 0 \wedge x_1 + \delta \leq 3 \wedge -(x_2 + \delta) < -4 \right. \\ \left. \wedge -(x_3 + \delta) < -1 \wedge (x_1 + \delta) - (x_2 + \delta) \leq 6 \right),$$

which is equivalent to

$$x_1 \geq 0 \wedge x_2 \geq 0 \wedge x_3 \geq 0 \\ \wedge \exists \delta \left(\begin{array}{l} 0 \leq \delta \wedge \delta \leq 3 - x_1 \wedge -x_2 + 4 < \delta \\ \wedge -x_3 + 1 < \delta \wedge x_1 - x_2 \leq 6 \end{array} \right).$$

δ is introduced into the formulas by replacing every clock, say, x_1 , with $x_1 + \delta$.

To get rid of the δ , we pretty much use the same technique for the weakest precondition of an assignment statement like $x_i := 0$ in the last subsection, except that now the role of x_i is replaced by that of δ . For simplicity, we assume that only \leq appears. In general, we are given a constraint like

$$\exists \delta \geq 0 \left(\eta \wedge \bigwedge_{1 \leq i \leq n} E_i + c_i \leq \delta \wedge \bigwedge_{1 \leq j \leq m} \delta \leq E'_j + c'_j \right),$$

where η is independent of δ . With the inequality transitivity of real numbers, the constraint is equivalent to

$$\eta \wedge \bigwedge_{1 \leq i \leq n, 1 \leq j \leq m} E_i + c_i \leq E'_j + c'_j.$$

The equivalence relies on the dense nature of real numbers. Thus the timed weakest precondition for η is

$$x_1 \geq 0 \wedge x_2 \geq 0 \wedge x_3 \geq 0 \wedge 0 \leq 3 - x_1 \\ \wedge -x_2 + 4 < 3 - x_1 \wedge -x_3 + 1 < 3 - x_1 \wedge x_1 - x_2 \leq 6,$$

which is equivalent to

$$x_1 \geq 0 \wedge x_2 \geq 0 \wedge x_3 \geq 0 \wedge x_1 \leq 3 \\ \wedge x_1 - x_2 < -1 \wedge x_1 - x_3 < 2 \wedge x_1 - x_2 \leq 6.$$

The derivation just mentioned is okay with DBM and has resulted in efficient implementations with DBM. But for BDD-like data structures, the method is likely to incur extra decision atoms involving δ , e.g., $0 - \delta, x_1 + \delta, -x_2 - \delta$, etc. and greatly increase the CPU time and memory consumption. In the following, we shall present a simplified derivation of the timed weakest precondition.

The above-mentioned equivalence relation, that is,

$$\exists \delta \geq 0 \left(\eta \wedge \bigwedge_{1 \leq i \leq n} E_i + c_i \leq \delta \wedge \bigwedge_{1 \leq j \leq m} \delta \leq E'_j + c'_j \right) \\ \text{iff } \eta \wedge \bigwedge_{1 \leq i \leq n, 1 \leq j \leq m} E_i + c_i \leq E'_j + c'_j,$$

can be rewritten as

$$\exists \delta \geq 0 \left(\eta \wedge \bigwedge_{1 \leq i \leq n} E_i - \delta \leq -c_i \right. \\ \left. \wedge \bigwedge_{1 \leq j \leq m} -E'_j + \delta \leq c'_j \right) \\ \text{iff } \eta \wedge \bigwedge_{1 \leq i \leq n, 1 \leq j \leq m} E_i - E'_j \leq c'_j - c_i.$$

Two advantages of this new formulation are:

1. Only \leq and $<$ are now used. This makes the representation more compatible with our CRDs with only upperbounds.
2. Now δ s are cancelled by adding a positive δ with another negative δ .

But where do those positive δ s come from? They only come from constraints like $x \leq c$. Where do those negative δ s come from? They only come from constraints like $-x' \leq c$ and $0 - \delta \leq 0$. For inequalities like $x - x' \leq c$, the positive and negative δ s cancel each other out and never really appear. Thus our key observation is that to cancel the δ s, we only have to perform the following two steps:

- Pairwisely match inequalities like $x \leq c$ and $-x' \leq c'$ and add them together to yield $x - x' \leq c + c'$.
Reasoning: $x \leq c$ will give rise to $x + \delta \leq c$ while $-x' \leq c$ will give rise to $-x' - \delta \leq c'$. Thus the addition of the two will give rise to $x - x' \leq c + c'$ and cancel the δ s.
- Remove any constraints of the form $-x' \leq c'$ from the zone.
Reasoning: $x \leq c$ will give rise to $x + \delta \leq c$ and later be transformed back to $x \leq c$ with the addition of $-\delta \leq 0$. But $-x' \leq c'$, which gives rise to $-x' - \delta \leq c'$, will be gone with the quantification.

Thus `time_bck(D)` can actually be implemented with

`rm_lbs(xtive(D, 0))`.

Here we can reuse procedure `preserve_xtive()` because the procedure can exactly match every pair of $x - 0\alpha$ and $0 - x'\alpha'$ and deduce $x - x'(\alpha + \alpha')$.

Procedure `rm_lbs()` only removes inequalities like $-x' \leq c'$. It is implemented as follows.

```

set  $\Psi$ ;
rm_lbs(D) {  $\Psi := \emptyset$ ; return rec_rm_lbs(D); }
rec_rm_lbs(D) with  $D = (x - x', (\alpha_i, D_i)_{1 \leq i \leq n})$ , {
  if D is true or false, return D;
  else if  $\exists H, (D, H) \in \Psi$ , return H;
  else if x is 0,  $H := \bigcup_{1 \leq i \leq n} \text{rec\_rm\_lbs}(D_i)$ ;
  else  $H := (x - x', (\alpha_i, \text{rec\_rm\_lbs}(D_i))_{1 \leq i \leq n})$ ;
   $\Psi := \Psi \cup \{(D, H)\}$ ; return H;
}

```

10 Implementation and experiments

Our newest implementation of CRD technology is in version 4.2 of the **Red** tool. **Red** 4.2 now supports TCTL model checking/simulation of real-time systems with multiprocesses, counterexample generation, forward/backward reachability analysis, and deadlock detection. The recent technical achievement of our implementation,

Red, includes numerical coverage estimation in symbolic simulation of dense-time systems [24] and speedup techniques for greatest fixpoint evaluation [23]. The new version, together with benchmarks, is available at

<http://cc.ee.ntu.edu.tw/~val/red>.

It supports options for forward/backward reachability analysis, deadlock detection, and counterexample generation. Reduction by elimination of inactive read-write system variables [21] is always executed.¹

We have carried out six experiments to see in reality how well various strategies for CRD perform. We use the following shorthand notations when referring to these strategies: *Fw* (forward reachability analysis), *Bk* (backward reachability analysis, by default), *NC* (without contained zone elimination, see Sect. 6, the default is to run with contained zone elimination), and evaluation orderings **B**, **H**, **F** (as defined in Sect. 8). We use six benchmarks to compare the performance:

- *Fischer’s timed mutual exclusion algorithm* [3, 16, 21]: 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* [25]: 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 transmission mode for ≥ 52 time units.
- *FDDI token-ring mutual exclusion protocol* [5, 10]: 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 messages in the mandatory synchronous mode and the optional asynchronous mode. The asynchronous mode is optional because a station process can do it only when, first, it has finished with synchronous mode transmission and, second, it detects that in the last cycle of token passing, all processes together have not used much network time. The biggest timing constant used is $50 \cdot m + 20$, where m is the number of stations. We want to verify that, at any moment, at most one station is holding the token.
- *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. The largest timing constant used is equal to the number of processes. 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. Each process initially comes 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 the smallest identifier will become the parent of the other process in the requester-responder pair. The largest timing constant used is 2. We want to verify that, at any moment, at least one process has no parent.
- *Bounded liveness of a leader-election algorithm*: This includes 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 two 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.

In Sect. 10.1, we report the performance of **Red** 4.2 with the three canonical forms in both forward and backward reachability analyses. In Sect. 10.2, we report the effect of contained-zone elimination. In Sect. 10.3, we report the performances of Kronos, UPPAAL2k, and **Red** 4.2 with respect to timing constant magnitudes. In Sect. 10.4, we report our experiment with the effect of representation fragmentation. In Sect. 10.5, we report the experiment with different evaluation orderings. In Sect. 10.6, we compare **Red** 4.2 with Kronos and UPPAAL2k.

Data are collected on a Pentium IV 1.7 GHz with 256 MB memory running LINUX. Execution times are collected for Kronos and UPPAAL2k, while times and memory (for data structures) are collected for **Red**. “s” and “m”, respectively, mean seconds and minutes of CPU time, “k” means kilobytes of memory space, “O/M” means “out-of-memory”, while “N/A” means “not available”.

10.1 Comparison of forward/backward analyses and the three canonical forms

The first experiment compares the performance, with respect to number of clocks, of forward/backward analyses and the three canonical forms; evaluation ordering **F** is used throughout all runs in this experiment. The performance data in Table 2 give rise to two observations.

- Backward analysis performs better than forward analysis. We have examined the CRDs generated in the iterations of the least fixpoint evaluation in Sect. 9. We found that, in general, backward analysis results in less enumeration of the ordering among clock readings than forward analysis. With fewer constraints for those total ordering among clock readings generated, more data sharing is now possible with BDD-like data structures.
- DRC in general performs better than the other two canonical forms. By examining the CRDs, we found

¹ A system 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 system variables can be omitted from state information without affecting the computations.

Table 2. Performance data of scalability with respect to various strategies

Benchmarks	m #clocks		red 4.2					
			Fw+Reduced	Fw+Closure	Fw+DRC	Bk+Reduced	Bk+Closure	Bk+DRC
Fischer's	3	3	0.91s/25k	0.19s/25k	0.18s/21k	0.51s/18k	0.16s/16k	0.15s/15k
mutual	4	4	9.99s/80k	1.02s/111k	0.94s/86k	3.19s/48k	0.83s/41k	0.75s/32k
exclusion	5	5	125.51s/338k	8.42s/581k	6.63s/416k	12.86s/111k	3.23s/87k	2.62s/70k
(m	6	6	2699.89s/1813k	114.51s/3214k	87.70s/2116k	43.45s/246k	10.31s/184k	7.38s/126k
processes	7	7	49 133.37s/15 003k	1255.91s/21 247k	947.16s/13 459k	136.36s/572k	30.49s/390k	18.87s/206k
)	8	8	O/M	O/M	11 736.05s/97 037k	460.32s/1363k	86.76s/816k	42.81s/312k
	9	9	O/M	O/M	O/M	1507.53s/3194k	230.94s/1709k	95.51s/445k
	10	10	O/M	O/M	O/M	4950.02s/7415k	577.42s/3755k	194.32s/610k
	11	11	O/M	O/M	O/M	15 983.33s/16 984k	1435.55s/8780k	372.00s/1171k
	12	12	O/M	O/M	O/M	52 912.10s/38 514k	3626.67s/20 395k	717.62s/2440k
	13	13	O/M	O/M	O/M	O/M	9584.88s/40 991k	1339.85s/5186k
CSMA/CD	3	4	20.32s/70k	0.31s/77k	0.29s/74k	0.45s/49k	0.08s/49k	0.06s/49k
(1 bus+	4	5	427.08s/379k	2.47s/371k	2.15s/344k	1.54s/83k	0.18s/82k	0.17s/82k
m senders	5	6	12 519.37s/2313k	32.75s/2402k	26.10s/1861k	4.82s/166k	0.42s/166k	0.41s/166k
)	6	7	over 2313m	410.79s/16 729k	328.73s/10 865k	17.40s/348k	1.00s/310k	0.92s/310k
	7	8	N/A	6757.75s/124 750k	4962.52s/68 209k	62.43s/745k	2.40s/617k	2.05s/617k
	8	9	N/A	O/M	O/M	239.17s/1574k	5.91s/1424k	4.59s/1424k
	9	10	N/A	O/M	O/M	849.55s/3318k	14.71s/3280k	11.06s/3280k
	10	11	N/A	O/M	O/M	2960.76s/6991k	37.66s/7485k	26.46s/7485k
	11	12	N/A	O/M	O/M	9443.45s/14 743k	100.95s/16 973k	67.76s/16 973k
	12	13	N/A	O/M	O/M	28 543.34s/31 142k	288.92s/38 197k	204.11s/38 197k
FDDI	11	23	0.36s/136k	0.37s/136k	0.38s/136k	0.17s/84k	0.17s/84k	0.17s/84k
token-ring	12	25	0.53s/139k	0.52s/139k	0.54s/139k	0.31s/139k	0.30s/139k	0.30s/139k
passing	20	41	2.03s/503k	2.00s/503k	2.03s/503k	0.67s/258k	0.67s/258k	0.68s/258k
(1 ring+	30	61	7.17s/1317k	7.17s/1317k	7.18s/1317k	1.98s/618k	1.96s/618k	1.96s/618k
m	40	81	17.95s/1746k	18.34s/1746k	18.34s/1746k	16.29s/1746k	16.02s/1746k	15.67s/1746k
stations	50	101	36.83s/2934k	36.88s/2934k	36.59s/2934k	38.63s/2934k	38.63s/2934k	38.36s/2934k
)	60	121	65.95s/4539k	65.32s/4539k	67.82s/4539k	81.68s/4539k	80.51s/4539k	81.57s/4539k
	70	141	106.29s/6599k	105.95s/6599k	106.41s/6599k	151.49s/6599k	151.80s/6599k	150.71s/6599k
pathos	3	6	5.14s/54k	0.28s/62k	0.28s/60k	0.23s/28k	0.04s/17k	0.04s/17k
(m	4	8	173.32s/434k	5.69s/462k	5.10s/450k	1.87s/55k	0.18s/36k	0.15s/36k
processes	5	10	18 333.51s/10 069k	180.24s/5287k	163.61s/5150k	13.83s/101k	0.66s/75k	0.53s/70k
)	6	12	over 2436m	6061.00s/83 289k	5515.89s/81 757k	100.20s/222k	2.94s/224k	1.97s/133k
	7	14	N/A	O/M	O/M	1029.59s/486k	16.86s/712k	9.88s/375k
	8	16	N/A	O/M	O/M	9980.55s/1198k	97.05s/2309k	51.86s/1112k
	9	18	N/A	O/M	O/M	80 303.34s/3291k	503.76s/74 722k	265.04s/3361k
	10	20	N/A	O/M	O/M	O/M	2550.62s/23 899k	1291.95s/10 238k
	11	22	N/A	O/M	O/M	O/M	13 425.09s/75 897k	6251.12s/31 204k
leader	3	3	0.04s/22k	0.04s/22k	0.03s/22k	0.05s/32k	0.05s/32k	0.05s/32k
(m	4	4	0.09s/37k	0.09s/37k	0.09s/37k	0.15s/72k	0.16s/72k	0.16s/72k
processes	5	5	0.21s/55k	0.21s/55k	0.21s/55k	0.51s/139k	0.49s/139k	0.52s/139k
)	6	6	0.41s/73k	0.41s/73k	0.42s/73k	1.68s/243k	1.63s/243k	1.66s/243k
	7	7	0.76s/93k	0.74s/93k	0.75s/93k	5.57s/395k	5.39s/395k	5.61s/395k
	8	8	1.43s/123k	1.41s/123k	1.43s/123k	19.92s/606k	19.90s/606k	19.26s/606k
	9	9	2.60s/163k	2.39s/163k	2.30s/163k	52.02s/891k	52.73s/891k	51.73s/891k
	10	10	3.61s/212k	3.65s/212k	3.60s/212k	122.84s/1261k	123.00s/1261k	123.80s/1261k
	11	11	5.41s/270k	5.37s/270k	5.47s/270k	265.20s/1758k	267.27s/1758k	260.82s/1758k
	12	12	7.92s/340k	7.92s/340k	7.79s/340k	533.19s/2816k	531.53s/2816k	532.21s/2816k
	13	13	11.09s/420k	11.20s/420k	11.39s/420k	1017.94s/4732k	1015.71s/4732k	1028.42s/4732k
	14	14	15.98s/513k	15.79s/513k	16.03s/513k	1893.71s/7927k	1889.36s/7927k	1893.06s/7927k
lbound	3	4	1.03s/104k	0.18s/104k	0.20s/104k	1.05s/40k	0.18s/45k	0.12s/39k
(m	4	5	14.26s/172k	1.50s/161k	1.43s/157k	16.40s/112k	1.47s/151k	0.98s/102k
processes	5	6	137.67s/547k	13.86s/541k	11.07s/465k	250.41s/371k	16.80s/563k	6.79s/308k
)	6	7	875.61s/1339k	103.44s/1935k	80.43s/1452k	3077.09s/1271k	149.52s/1909k	44.95s/934k
	7	8	3928.86s/2930k	634.77s/7142k	477.76s/5305k	26 926.49s/4237k	965.46s/5982k	228.41s/2714k
	8	9	14 734.04s/5877k	4357.20s/28 948k	3053.32s/17 741k	190 045.81s/11 840k	5166.10s/17 174k	1017.41s/7593k
	9	10	48 262.21s/10 934k	29 586.74s/123 103k	19 793.96s/72 137k	over3756m	25 536.59s/47 760k	4271.08s/20 493k

that DRC results in better balance between CRD width and depth than closure and reduced forms.

Between the closure form and the reduced form, the former performs better than the latter. This is because our algorithm for reduced form construction is quite complicated. In general, to compute the reduced form, closure form has to be constructed in some way before-

hand. This suggests that the reduced form by definition cannot perform better than the closure form.

We found that backward analysis in combination with DRC performs better than the rest. So in the following discussion, we shall choose this combination to make more comparisons.

10.2 Performance with respect
to zone-containment operations

In the second experiment, we want to check if our effort in eliminating contained zones really pays off. We choose

to run **Red** 4.2 against the six benchmarks with forward and backward analysis. In our implementation, we use the flexible approach described in Sect. 6 such that only when CRD sizes are made smaller, the contained zones will be eliminated. The performance data for both with

Table 3. Performance data of scalability with respect to zone-containment operations

Benchmarks	m	#clocks	red 4.2			
			Fw+NC	Fw	Bk+NC	Bk
Fischer's	3	3	0.17s/24k	0.18s/21k	0.15s/14k	0.15s/15k
mutual	4	4	0.93s/114k	0.94s/86k	0.73s/33k	0.75s/32k
exclusion	5	5	10.68s/609k	6.63s/416k	2.61s/70k	2.62s/70k
(m	6	6	196.49s/4810k	87.70s/2116k	7.63s/126k	7.38s/126k
processes	7	7	3240.72s/45584k	947.16s/13459k	17.94s/206k	18.87s/206k
)	8	8	O/M	11736.05s/97037k	42.38s/312k	42.81s/312k
	9	9	O/M	O/M	92.29s/445k	95.51s/445k
	10	10	O/M	O/M	185.33s/610k	194.32s/610k
	11	11	O/M	O/M	375.91s/1171k	372.00s/1171k
	12	12	O/M	O/M	725.00s/2445k	717.62s/2440k
	13	13	O/M	O/M	1349.26s/5169k	1339.85s/5186k
CSMA/CD	3	4	0.28s/76k	0.29s/74k	0.09s/49k	0.06s/49k
(1 bus+	4	5	2.16s/414k	2.15s/344k	0.16s/82k	0.17s/82k
m senders	5	6	28.09s/2391k	26.10s/1861k	0.41s/166k	0.41s/166k
)	6	7	366.70s/14539k	328.73s/10865k	0.89s/310k	0.92s/310k
	7	8	6361.45s/93419k	4962.52s/68209k	2.09s/617k	2.05s/617k
	8	9	O/M	O/M	5.04s/1427k	4.59s/1424k
	9	10	O/M	O/M	10.65s/3283k	11.06s/3280k
	10	11	O/M	O/M	26.35s/7495k	26.46s/7485k
	11	12	O/M	O/M	69.48s/16986k	67.76s/16973k
	12	13	O/M	O/M	208.30s/38176k	204.11s/38197k
FDDI	11	23	0.37s/136k	0.38s/136k	0.17s/84k	0.17s/84k
token-ring	12	25	0.53s/139k	0.54s/139k	0.31s/139k	0.30s/139k
passing	20	41	1.90s/503k	2.03s/503k	0.67s/258k	0.68s/258k
(1 ring+	30	61	6.67s/1317k	7.18s/1317k	1.98s/618k	1.96s/618k
m	40	81	18.44s/1746k	18.34s/1746k	16.29s/1746k	15.67s/1746k
stations	50	101	35.91s/2934k	36.59s/2934k	38.63s/2934k	38.36s/2934k
)	60	121	65.69s/4539k	67.82s/4539k	81.68s/4539k	81.57s/4539k
	70	141	105.75s/6599k	106.41s/6599k	151.49s/6599k	150.71s/6599k
pathos	3	6	0.29s/68k	0.28s/60k	0.05s/16k	0.04s/17k
(m	4	8	11.40s/1102k	5.10s/450k	0.15s/35k	0.15s/36k
processes	5	10	723.46s/29201k	163.61s/5150k	0.52s/70k	0.53s/70k
)	6	12	O/M	5515.89s/81757k	2.08s/183k	1.97s/133k
	7	14	O/M	O/M	12.29s/701k	9.88s/375k
	8	16	O/M	O/M	89.01s/3083k	51.86s/1112k
	9	18	O/M	O/M	646.28s/15176k	265.04s/3361k
	10	20	O/M	O/M	5678.75s/83174k	1291.95s/10238k
	11	22	O/M	O/M	O/M	6251.12s/31204k
leader	3	3	0.03s/22k	0.03s/22k	0.04s/32k	0.05s/32k
(m	4	4	0.09s/37k	0.09s/37k	0.14s/72k	0.16s/72k
processes	5	5	0.21s/55k	0.21s/55k	0.47s/139k	0.52s/139k
)	6	6	0.42s/73k	0.42s/73k	1.60s/243k	1.66s/243k
	7	7	0.75s/93k	0.75s/93k	5.23s/395k	5.61s/395k
	8	8	1.43s/123k	1.43s/123k	18.68s/606k	19.26s/606k
	9	9	2.51s/163k	2.30s/163k	52.84s/891k	51.73s/891k
	10	10	3.62s/212k	3.60s/212k	123.29s/1261k	123.80s/1261k
	11	11	5.37s/270k	5.47s/270k	265.14s/1758k	260.82s/1758k
	12	12	8.40s/340k	7.79s/340k	529.94s/2814k	532.21s/2816k
	13	13	11.04s/420k	11.39s/420k	1012.28s/4741k	1028.42s/4732k
	14	14	16.01s/513k	16.03s/513k	1885.42s/7924k	1893.06s/7927k
lbound	3	4	0.19s/105k	0.20s/104k	0.12s/38k	0.12s/39k
(m	4	5	1.41s/157k	1.43s/157k	0.94s/102k	0.98s/102k
processes	5	6	10.65s/465k	11.07s/465k	6.18s/309k	6.79s/308k
)	6	7	78.51s/1453k	80.43s/1452k	50.06s/1029k	44.95s/934k
	7	8	460.35s/5304k	477.76s/5305k	256.45s/3086k	228.41s/2714k
	8	9	2864.78s/17743k	3053.32s/17741k	1143.94s/8705k	1017.41s/7593k
	9	10	18351.06s/72132k	19793.96s/72137k	4889.95s/23583k	4271.08s/20493k

Table 4. Performance data of scalability with respect to timing-constant magnitude

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
UPPAAL2k	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
red 4.2	3	0.16s/14k	0.18s/14k	0.19s/15k	0.17s/14k	0.17s/14k	0.17s/14k
	4	0.76s/33k	0.76s/33k	0.76s/33k	0.76s/33k	0.77s/33k	0.78s/33k

Table 5. Representation complexity of reachable state spaces in CDD and CRD

Benchmarks	Concurrency	CDD			CRD			Size Ratio CDD/CRD	
		#nodes	#arcs	size	#nodes	#arcs	size		
Fischer's mutual exclusion	3 processes	53	103	156	65	110	175	0.891	
	4 processes	103	217	320	125	228	353	0.906	
	5 processes	174	382	556	206	389	595	0.934	
	6 processes	276	620	896	309	595	904	0.991	
	7 processes	429	977	1406	436	850	1286	1.093	
	8 processes	673	1545	2218	589	1158	1747	1.269	
	9 processes	1088	2508	3596	770	1523	2293	1.568	
	10 processes	1834	4234	6068	981	1949	2930	2.070	
	CSMA/CD	bus+3 senders	38	52	90	46	53	99	0.909
		bus+4 senders	48	67	115	54	63	117	0.982
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	1.137	
bus+9 senders		98	142	240	94	113	207	1.159	
FDDI token-ring passing	3 stations	37	45	82	46	54	100	0.820	
	4 stations	57	68	125	69	80	149	0.838	
	5 stations	81	105	186	92	106	198	0.939	
	6 stations	109	141	250	122	139	261	0.957	
	7 stations	141	182	323	156	176	332	0.972	
	8 stations	177	228	405	194	217	411	0.985	
Pathos schedulibility	3 processes	25	33	58	27	32	59	0.983	
	4 processes	89	135	224	90	113	203	1.103	
	5 processes	288	484	772	258	328	586	1.317	
	6 processes	983	1811	2794	733	934	1667	1.676	
	7 processes	3662	7296	10958	2108	2684	4792	2.286	
	8 processes	14872	31583	46455	6129	7791	13920	3.337	
leader- election safeness	3 processes	93	116	209	118	141	259	0.806	
	4 processes	198	253	451	255	310	565	0.798	
	5 processes	352	454	806	457	559	1016	0.793	
	6 processes	564	731	1295	736	903	1639	0.790	
	7 processes	843	1096	1939	1104	1357	2461	0.787	
	8 processes	1410	1775	3185	1785	2150	3935	0.809	
	9 processes	1949	2451	4400	2466	2968	5434	0.809	
	leader- election bounded liveness	3 processes	125	148	273	152	174	326	0.837
		4 processes	379	493	872	448	542	990	0.880
5 processes		1464	1986	3450	1444	1780	3224	1.070	
6 processes		3353	4755	8108	3216	4119	7335	1.105	
7 processes		6451	9433	15884	5835	7642	13477	1.178	
8 processes		10836	16135	26971	9408	12484	21892	1.232	
9 processes	27609	42554	70163	16662	21692	38354	1.829		

Table 6. Performance data with respect to evaluation ordering

Benchmarks	Concurrency	B	H	F	
Fischer's mutual exclusion	3 processes	0.15s/16k	0.16s/15k	0.15s/15k	
	4 processes	0.95s/40k	0.84s/32k	0.75s/32k	
	5 processes	3.62s/111k	2.77s/70k	2.62s/70k	
	6 processes	15.37s/288k	7.88s/126k	7.38s/126k	
	7 processes	68.76s/718k	20.51s/243k	18.87s/206k	
	8 processes	296.48s/1744k	49.84s/522k	42.81s/312k	
	9 processes	1073.42s/4216k	115.63s/1127k	95.51s/445k	
	10 processes	3810.61s/10196k	261.54s/2448k	194.32s/610k	
	11 processes	12843.44s/24958k	556.14s/5318k	372.00s/1171k	
	12 processes	45568.55s/62739k	1235.56s/11496k	717.62s/2440k	
	CSMA/CD	bus+3 senders	0.09s/53k	0.09s/49k	0.06s/49k
		bus+4 senders	0.24s/155k	0.21s/94k	0.17s/82k
bus+5 senders		0.79s/444k	0.55s/267k	0.41s/166k	
bus+6 senders		2.86s/1176k	1.50s/767k	0.92s/310k	
bus+7 senders		10.22s/3039k	4.60s/2169k	2.05s/617k	
bus+8 senders		33.48s/7699k	13.53s/5990k	4.59s/1424k	
bus+9 senders		105.26s/19063k	42.63s/16165k	11.06s/3280k	
bus+10 senders		312.38s/41437k	156.66s/42685k	26.46s/7485k	
bus+11 senders		O/M	O/M	67.76s/16973k	
bus+12 senders		O/M	O/M	204.11s/38197k	
FDDI token-ring passing		11 stations	27.53s/33008k	0.18s/84k	0.17s/84k
		12 stations	73.37s/52396k	0.30s/139k	0.30s/139k
	20 stations	O/M	0.67s/258k	0.68s/258k	
	30 stations	O/M	1.89s/618k	1.96s/618k	
	40 stations	O/M	15.53s/1746k	15.67s/1746k	
	50 stations	O/M	37.70s/2934k	38.36s/2934k	
Pathos	60 stations	O/M	80.55s/4539k	81.57s/4539k	
	3 processes	0.04s/17k	0.04s/17k	0.04s/17k	
	4 processes	0.14s/37k	0.16s/36k	0.15s/36k	
	5 processes	0.52s/96k	0.58s/70k	0.53s/70k	
	6 processes	2.30s/215k	2.34s/179k	1.97s/133k	
	7 processes	13.84s/516k	13.32s/575k	9.88s/375k	
	8 processes	80.83s/1362k	82.84s/2034k	51.86s/1112k	
	9 processes	418.05s/4355k	480.17s/7428k	265.04s/3361k	
	10 processes	2016.30s/14168k	2766.90s/26920k	1291.95s/10238k	
	11 processes	10288.20s/46753k	20373.96s/98237k	6251.12s/31204k	
	Leader	3 processes	0.05s/34k	0.05s/32k	0.05s/32k
4 processes		0.16s/81k	0.16s/72k	0.16s/72k	
5 processes		0.70s/171k	0.49s/139k	0.52s/139k	
6 processes		4.07s/376k	1.89s/243k	1.66s/243k	
7 processes		21.87s/934k	5.79s/395k	5.61s/395k	
8 processes		99.76s/2373k	19.44s/606k	19.26s/606k	
9 processes		400.35s/5949k	52.00s/891k	51.73s/891k	
10 processes		1521.33s/14871k	123.05s/1261k	123.80s/1261k	
11 processes		5755.83s/37751k	264.64s/1758k	260.82s/1758k	
12 processes		23301.86s/97280k	530.15s/2816k	532.21s/2816k	
13 processes		O/M	1025.77s/4732k	1028.42s/4732k	
14 processes		O/M	1896.40s/7927k	1893.06s/7927k	
lbound		3 processes	0.11s/37k	0.15s/42k	0.12s/39k
		4 processes	0.91s/116k	1.13s/134k	0.98s/102k
	5 processes	8.22s/402k	8.84s/552k	6.79s/308k	
	6 processes	65.87s/1381k	71.18s/2341k	44.95s/934k	
	7 processes	378.36s/4523k	411.38s/8944k	228.41s/2714k	
	8 processes	1862.98s/14099k	2286.83s/31547k	1017.41s/7593k	
	9 processes	8736.78s/42117k	15956.90s/104330k	4271.08s/20493k	

and without contained-zone elimination are in Table 3. In the 12 cases, 5 (Fischer's with Fw, CSMA/CD with Fw, Pathos with Fw, Pathos with Bk, and lbound with Bk) show that the contained-zone elimination option gives us

a significant performance boost. In all the other cases, the option either does not improve much or charges us a little for the overhead (case lbound with Fw). This set of data suggests that it is worthwhile to use the option always.

10.3 Performance with respect to timing constant magnitude complexity

The performance of some previous technologies, e.g., NDD and RED, does not scale very well to the magnitude

of the timing constant. The data in Table 4 are collected by running Kronos, UPPAAL2k, and **Red** 4.2 with strategy Bk+DRC+F. Against Fischer’s mutual exclusion algorithm with various timing constant magnitudes. The performance data are in Table 4. The table suggests

Table 7. Performance comparison with other tools with respect to number of processes

Benchmarks	m	#clocks	Kronos	UPPAAL	red 4.2	
			2.4.5	3.2.4	Fw+DRC	Bk+DRC
Fischer’s	3	3	0.02s	0.01s	0.18s/21k	0.15s/15k
mutual	4	4	0.15s	0.09s	0.94s/86k	0.75s/32k
exclusion	5	5	0.95s	2.97s	6.63s/416k	2.62s/70k
(m	6	6	O/M	292.56s	87.70s/2116k	7.38s/126k
processes	7	7	O/M	O/M	947.16s/13459k	18.87s/206k
)	8	8	O/M	O/M	11736.05s/97037k	42.81s/312k
	9	9	O/M	O/M	O/M	95.51s/445k
	10	10	O/M	O/M	O/M	194.32s/610k
	11	11	O/M	O/M	O/M	372.00s/1171k
	12	12	O/M	O/M	O/M	717.62s/2440k
	13	13	O/M	O/M	O/M	1339.85s/5186k
CSMA/CD	3	4	0.01s	0.01s	0.29s/74k	0.06s/49k
(1 bus+	4	5	0.04s	0.04s	2.15s/344k	0.17s/82k
m senders	5	6	0.26s	0.46s	26.10s/1861k	0.41s/166k
)	6	7	1.91s	13.87s	328.73s/10865k	0.92s/310k
	7	8	O/M	752.42s	4962.52s/68209k	2.05s/617k
	8	9	O/M	O/M	O/M	4.59s/1424k
	9	10	O/M	O/M	O/M	11.06s/3280k
	10	11	O/M	O/M	O/M	26.46s/7485k
	11	12	O/M	O/M	O/M	67.76s/16973k
	12	13	O/M	O/M	O/M	204.11s/38197k
FDDI	11	23	72.61s	30.96s	0.38s/136k	0.17s/84k
token-ring	12	25	O/M	118.35s	0.54s/139k	0.30s/139k
passing	20	41	O/M	O/M	2.03s/503k	0.68s/258k
(1 ring+	30	61	O/M	O/M	7.18s/1317k	1.96s/618k
m	40	81	O/M	O/M	18.34s/1746k	15.67s/1746k
stations	50	101	O/M	O/M	36.59s/2934k	38.36s/2934k
)	60	121	O/M	O/M	67.82s/4539k	81.57s/4539k
	70	141	O/M	O/M	106.41s/6599k	150.71s/6599k
Pathos	3	6	0.0s	0.01s	0.28s/60k	0.04s/17k
(m	4	8	O/M	0.11s	5.10s/450k	0.15s/36k
processes	5	10	O/M	8.02s	163.61s/5150k	0.53s/70k
)	6	12	O/M	O/M	5515.89s/81757k	1.97s/133k
	7	14	O/M	O/M	O/M	9.88s/375k
	8	16	O/M	O/M	O/M	51.86s/1112k
	9	18	O/M	O/M	O/M	265.04s/3361k
	10	20	O/M	O/M	O/M	1291.95s/10238k
	11	22	O/M	O/M	O/M	6251.12s/31204k
Leader	3	3	0.13s	0.00s	0.03s/22k	0.05s/32k
(m	4	4	4.88s	0.01s	0.09s/37k	0.16s/72k
processes	5	5	O/M	0.09s	0.21s/55k	0.52s/139k
)	6	6	O/M	0.74s	0.42s/73k	1.66s/243k
	7	7	O/M	3.34s	0.75s/93k	5.61s/395k
	8	8	O/M	O/M	1.43s/123k	19.26s/606k
	9	9	O/M	O/M	2.30s/163k	51.73s/891k
	10	10	O/M	O/M	3.60s/212k	123.80s/1261k
	11	11	O/M	O/M	5.47s/270k	260.82s/1758k
	12	12	O/M	O/M	7.79s/340k	532.21s/2816k
	13	13	O/M	O/M	11.39s/420k	1028.42s/4732k
	14	14	O/M	O/M	16.03s/513k	1893.06s/7927k
lbound	3	4	0.13s	0.00s	0.20s/104k	0.12s/39k
(m	4	5	4.88s	0.02s	1.43s/157k	0.98s/102k
processes	5	6	O/M	0.08s	11.07s/465k	6.79s/308k
)	6	7	O/M	0.65s	80.43s/1452k	44.95s/934k
	7	8	O/M	3.14s	477.76s/5305k	228.41s/2714k
	8	9	O/M	O/M	3053.32s/17741k	1017.41s/7593k
	9	10	O/M	O/M	19793.96s/7213k	4271.08s/20493k

that CRD is at least as good as DBM technology as far as performance scalability with respect to timing constant complexity is concerned.

10.4 Performance with representation fragmentation

To observe the effect of the representation fragmentation phenomenon of CDD, we have endeavored to implement some CDD manipulation routines. This enables us to collect data, shown in Table 5, of sizes of reachable state space representations in CDD and CRD for the six benchmarks. The state space representation of benchmark leader-election safeness and liveness are very similar except that one more global clock is used for the liveness benchmark. From the table we observe that CDD demonstrates exponential blowup for two of the benchmarks (Fischer’s, pathos) in comparison with CRD. For the FDDI benchmark, it seems a blowup is ongoing. 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 decision atom, while in CRD two separate decision atoms have to be used. For instance, the constraint of $3 < x_1 - x_2 < 5$ can be specified with one decision atom in CDD. But in CRD, it is specified with two decision atoms as $x_1 - x_2 < 5 \wedge x_2 - x_1 < -3$. This explains why, when CDD performs better, it does so by a constant factor.

10.5 Performance with respect to evaluation ordering

We compare the performance of **Red** 4.2 with respect to the evaluation orderings discussed in Sect. 8. The performance data in Table 6 are very compatible with the traditional experience with BDD-like data structures. That is, decision atoms with a strong relation should be placed near each other.

10.6 Comparison with other tools

We choose to compare the performance of **Red** with *Kronos* [5, 10, 25] (version 2.4, release 5) and *UPPAAL2k* [6, 15] (version 3.2.4), which are perhaps the two best-known model checkers for real-time systems with DBM technology. Comparison with these two tools also gives us some rough feeling about how well CRD technology performs against DBM technology with both forward analysis (used in UPPAAL2k) and backward analysis (default in Kronos). The performance data are shown in Table 7. Except for the FDDI benchmark, UPPAAL2k runs with options “-aSOWD”. For the FDDI benchmark, UPPAAL2k runs with options “-SOTDda”. Kronos is invoked with backward analysis.

The performance data show that CRD technology is more space efficient and scales better with respect to the

number of clocks than the DBM implementation used in Kronos and UPPAAL2k. For time complexities, CRD with backward analysis and DRC are outperformed only by *DBM technologies with a small number of clocks*. This shows that DBM technology generally has good time complexities for small systems with its cubic complexity all-pair shortest-path algorithm on matrices. But for systems with high concurrency, **Red** 4.2 with backward analysis and DRC normal form outperform the other targets. Thus the reasoning behind the design of CRD seems justified.

Also, for forward analysis, in general, CRD technology shows greater space efficiency than DBM technology with forward analysis as represented by UPPAAL2k. But in time complexity, DBM technology is better. In our experience, BDD-like data structures can create a lot of intermediate data structures only to be garbage-collected. For example, in our CSMA/CD benchmark, we usually observed that the maximum memory consumption in the model checking process is 100 times the memory requirement for the final reachable state space representation. This may imply that with better garbage-collection techniques, we can further improve the performance of CRD technology. In comparison, DBM manipulation usually does not incur any intermediate space consumption. You basically just work on the same two-dimensional matrix.

11 Conclusion

Efficient model-checking of TAs requires a deep understanding of the subtle interaction between data structures and algorithms. We believe that earlier 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 TA state spaces. We have carried out extensive experiments to justify our arguments. We have also developed techniques for the DRC form. We believe the experience reported in this paper will be very valuable in the implementation of industry-usable model checkers for real-time systems.

In the future, it will be interesting to see if the CRD technology can be further improved. We suggest the following two approaches.

- *Better garbage-collection mechanism.* Garbage-collecting intermediate structures is not only complicated but also very time consuming. At this moment, we use a stack to keep those CRDs that are temporarily used. The garbage-collection procedure is invoked between major steps of the weakest precondition calculation. We believe that space usage can be more efficient if a more complicated and proactive garbage-collector, specially tailored for CRDs, is designed and used.
- *More canonical form choices.* Although we have tried six different choices of canonical forms in the last

2 years, we believe that more canonical forms and their construction algorithms can still be investigated. More experiments with new choices may lead to an even better balance between the depth and width of BDD-like data structures.

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

References

1. Asarin E, Bozga M, Kerbrat A, Maler O, Pnueli A, Rasse A (1997) Data-structures for the verification of timed automata. In: Proceedings of HART'97 (International Workshop on Hybrid And Real-Time Systems) 1997. Lecture notes in computer science, vol 1201. Springer, Berlin Heidelberg New York
2. Alur R, Courcoubetis C, Dill DL (1990) Model checking for real-time systems. IEEE LICS (5th International Symposium on Logics in Computer Science), June 1990, Philadelphia, USA, IEEE Computer Society, pp 414–425
3. Balarin F (1996) Approximate reachability analysis of timed automata. IEEE RTSS (17th IEEE Real-Time Systems Symposium), December 1996, Washington D.C., USA, IEEE Computer Society, pp 52–61
4. Burch JR, Clarke EM, McMillan KL, Dill DL, Hwang LJ (1990) Symbolic model checking: 10^{20} states and beyond. IEEE LICS (5th International Symposium on Logics in Computer Science), June 1990, Philadelphia, USA, IEEE Computer Society, pp 428–439
5. Bozga M, Daws C, Maler O (1998) Kronos: a model-checking tool for real-time systems. In: Proceedings of the 10th conference on computer-aided verification (CAV'98), June/July 1998. Lecture notes in computer science, vol 1427. Springer, Berlin Heidelberg New York
6. Bengtsson J, Larsen K, Larsson F, Pettersson P, Wang Y (1996) UPPAAL – a tool suite for automatic verification of real-time systems. In: Proceedings of the hybrid control system symposium, 1996. Lecture notes in computer science. Springer, Berlin Heidelberg New York
7. Behrmann G, Larsen KG, Pearson J, Weise C, Wang Y (1999) Efficient timed reachability analysis using clock difference diagrams. In: Proceedings of the conference on computer-aided verification (CAV'99), Trento, Italy, July 1999. Lecture notes in computer science, vol 1633. Springer, Berlin Heidelberg New York
8. Bryant RE (1986) Graph-based algorithms for Boolean function manipulation. IEEE Trans Comput C-35(8)
9. Dill DL (1989) Timing assumptions and verification of finite-state concurrent systems. In: Proceedings of the conference on computer-aided verification (CAV'89), 1989. Lecture notes in computer science, vol 407. Springer, Berlin Heidelberg New York
10. Daws C, Olivero A, Tripakis S, Yovine S (1996) The tool KRONOS. In: Proceedings of the 3rd conference on hybrid systems, 1996. Lecture notes in computer science, vol 1066. Springer, Berlin Heidelberg New York
11. Henzinger TA, Nicollin X, Sifakis J, Yovine S (1992) Symbolic model checking for real-time systems. IEEE LICS (7th International Symposium on Logics in Computer Science), Santa Cruz, USA, IEEE Computer Society, June 1992, pp 394–406
12. Larsen KG, Larsson F, Pettersson P, Wang Y (1998) Efficient verification of real-time systems: compact data-structure and state-space reduction. IEEE RTSS (19th IEEE Real-Time Systems Symposium), Madrid, Spain, IEEE Computer Society, December 1998
13. Moller J, Lichtenberg J, Andersen HR, Hulgaard H (1999) Difference decision diagrams. In: Proceedings of the annual conference of the European Association for Computer Science Logic (CSL), September 1999, Madrid, Spain
14. Moller J, Lichtenberg J, Andersen HR, Hulgaard H (1999) Fully symbolic model-checking of timed systems using difference decision diagrams. In: Proceedings of the workshop on symbolic model-checking (SMC), July 1999, Trento, Italy
15. Pettersson P, Larsen KG (2000) UPPAAL2k. Bull Eur Assoc Theor Comput Sci 70:40–44
16. Wang F (2000a) Efficient data-structure for fully symbolic verification of real-time software systems. In: Proceedings of the conference on tools and algorithms for the construction and analysis of systems (TACAS 2000), March 2000, Berlin. Lecture notes in computer science, vol 1785. Springer, Berlin Heidelberg New York
17. Wang F (2000b) Region encoding diagram for fully symbolic verification of real-time systems. In: Proceedings of the the 24th COMPSAC (24th Computer Software and Applications Conference), October 2000, Taipei, Taiwan, ROC. IEEE Computer Society, IEEE Press, New York, pp 509–515
18. Wang F (2001a) RED: Model-checker for timed automata with clock-restriction diagram. In: Proceedings of the workshop on real-time tools, August 2001. Technical Report 2001-014, ISSN 1404-3203, Department of Information Technology, Uppsala University, Uppsala, Sweden
19. Wang F (2001b) Symbolic verification of complex real-time systems with clock-restriction diagram. In: Proceedings of FORTE'2001 (21st IFIP International Conference on Formal Techniques for Networked and Distributed Systems), August 2001, Cheju Island, Korea. Kluwer, Dordrecht, pp 235–250
20. Wang F (2003) Symbolic parametric analysis of linear hybrid systems with BDD-like data-structures. ACM CoRR (Computing Research Repository, <http://www.acm.org/corr/>), user:cs.DS/0306113
21. Wang F, Hsiung P-A (2002) Efficient and user-friendly verification. IEEE Trans Comput
22. Wang F, Mok A, Emerson EA (1993) Symbolic model-checking for distributed real-time systems. In: Proceedings of the FME (1st International Symposium of Formal Methods Europe), April 1993, Odense, Denmark. Lecture notes in computer science, vol 670. Springer, Berlin Heidelberg New York, pp 632–651
23. Wang F, Hwang G-D, Yu F (2003a) TCTL inevitability analysis of dense-time systems. In: Proceedings of the 8th conference on implementation and application of automata (CIAA), July 2003, Santa Barbara, CA. Lecture notes in computer science, vol 2759. Springer, Berlin Heidelberg New York
24. Wang F, Hwang G-D, Yu F (2003b) Numerical coverage estimation for the symbolic simulation of real-time systems. To appear in the proceedings of FORTE'2003 (23rd IFIP International Conference on Formal Techniques for Networked and Distributed Systems), September–October 2003, Berlin. Lecture notes in computer science, vol 2767. Springer, Berlin Heidelberg New York, pp 160–176
25. Yovine S (1997) Kronos: a verification tool for real-time systems. Int J Softw Tools Technol Transfer 1(1/2)

Appendix : Definitions of the three evaluation orderings

Here we consider the evaluation ordering of decision atoms of systems with concurrent processes with local and global system variables. For nonclock system variables, the same variable names are used as the BDD decision atom names in the CRD+BDD. For clock system variables $x, x' \in \{0\} \cup X$, the relevant decision atoms in the CRD+BDD are *CRD decision atoms* with names like $x - x'$. Given a TA A , we let Δ_A be the set of BDD decision atoms and Γ_A the set of CRD decision atoms.

The input language of our implementation allows the declaration of processes with corresponding local system variables. The processes are labeled with identifier

1 . . . m . For each local system variable u , we let $\text{proc}(u)$ be the identifier of the process to which u (or x) is local. For each global variable u , we let $\text{proc}(u) = 0$. Also, $\text{proc}(0) = 0$. Given a CRD decision atom $x - x'$ in a CRD+BDD, we let $\text{proc}(x - x') = \max(\text{proc}(x), \text{proc}(x'))$. We here extend the meaning of evaluation index $\Omega()$ in Sect. 4 to both CRD decision atoms and BDD decision atoms. For any two decision atoms $w, w' \in \Delta_A \cup \Gamma_A$, we write $w \prec_\Omega w'$ to denote that w precedes w' in evaluation ordering Ω .

According to the ordering in which a system variable is declared in the input, we also define attribute $\text{offset}()$. Given two system variables u, u' , $\text{offset}(u) < \text{offset}(u')$ iff u is declared before u' in the input file. Given a CRD decision atom $x - x'$, we let $\text{offset}(x - x') = \max(\text{offset}(x), \text{offset}(x'))$.

We have general rules and special rules for the decision of precedence relations among decision atoms. There are two general rules applied to all three of these orderings.

- For every two distinct clocks x, x' , we put a CRD decision atom like $x' - x$ immediately next to $x - x'$ in the evaluation ordering. This arrangement allows us to efficiently check for some trivial negative cycles.
- Given two decision atoms w, w' , if the evaluation ordering cannot be determined by the following special rules, then $w \prec_\Omega w'$ iff $\text{offset}(w) < \text{offset}(w')$ for all $\Omega \in \{\mathbf{B}, \mathbf{H}, \mathbf{F}\}$.

Given a CRD decision atom $x - x'$ in a CRD+BDD, we let $\text{proc}_{\min}(x - x') = \min(\text{proc}(x), \text{proc}(x'))$. Also, $\text{offset}_{\min}(x - x') = \min(\text{offset}(x), \text{offset}(x'))$. For BDD decision atom w , $\text{proc}_{\min}(w) = \text{proc}(w)$ and $\text{offset}_{\min}(w) = \text{offset}(w)$. Given two decision atoms v and v' , we write $v \sqsubset v'$ iff one of the following four conditions is true: (1) if $\text{proc}(v) < \text{proc}(v')$, (2) else if $\text{proc}_{\min}(v) < \text{proc}_{\min}(v')$, (3) else if $\text{offset}(v) < \text{offset}(v')$, or (4) else if $\text{offset}_{\min}(v) < \text{offset}_{\min}(v')$. Intuitively, $v \sqsubset v'$ means that v precedes v' according to the process ordering and the declaration ordering.

The special rules are as follows.

- B:** (1) $\forall u \in \Delta_A \forall x - x' \in \Gamma_A (u \prec_{\mathbf{B}} x - x')$, i.e., BDD decision atoms precede CRD decision atoms.
(2) $\forall x - x', y - y' \in \Gamma_A (x - x' \prec_{\mathbf{B}} y - y' \text{ iff } x - x' \sqsubset y - y')$.
- H:** (1) $\forall u \in \Delta_A \forall x - x' \in \Gamma_A$ s.t. $x = 0$ or $x' = 0$ ($u \prec_{\mathbf{H}} x - x'$ iff $\text{proc}(u) \leq \text{proc}(x - x')$).
(2) $\forall x - 0, 0 - x, y - y' \in \Gamma_A$ s.t. $0 \notin \{y, y'\}$ ($x - 0 \prec_{\mathbf{H}} y - y' \wedge 0 - x \prec_{\mathbf{H}} y - y'$).
(3) $\forall x - x', y - y' \in \Gamma_A$ s.t. $0 \notin \{x, x', y, y'\}$ ($x - x' \prec_{\mathbf{H}} y - y'$ iff $x - x' \sqsubset y - y'$).
- F:** (1) $\forall u \in \Delta_A \forall x - x' \in \Gamma_A (u \prec_{\mathbf{F}} x - x'$ iff $\text{proc}(u) \leq \text{proc}(x - x')$).
(2) $\forall x - x', y - y' \in \Gamma_A (x - x' \prec_{\mathbf{F}} y - y'$ iff $x - x' \sqsubset y - y'$).