Region Encoding Diagram for Fully Symbolic Verification of Real-Time Systems*

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

Abstract

RED (Region Encoding Diagram), first reported in [14], is a BDD-like data-structure for fully symbolic verification of symmetric real-time systems with single clock per process. We propose to extend RED for asymmetric real-time systems with unrestricted number of global or local clocks. Unlike in DBM which records differences between pairs of clock readings, we records the ordering among fractional parts of clock readings into integer sequences encoded in RED's. Like BDD, the new RED is also a minimal canonical form for its target system state-space representations. The number of variables used in RED is $O(|X|\log|X|)$ when X is the clock set in the input system description. Experiment has been carried out to show the possible verification efficiency through the intense data-sharing nature of RED.

1 Introduction

Fully symbolic verification technologies, like BDD[4, 8], can be efficient in both space and time complexities with its intense data-sharing in the manipulation of state space representations. Recently, Wang has proposed a new BDD-like data-structure called *Region-Encoding Diagram (RED)* for symbolic verification of symmetric systems with single local clock per process[14]. In this manuscript, we extend RED to asymmetric systems which may have unrestricted number of local or global clocks. The ordering among fractional parts of clock readings is explicitly recorded as an integer sequence encoded in RED. To keep record of such sequences, we add one auxiliary $O(\log|X|)$-bit variable per clock with X as the set of clocks, global or local, in the systems. Like BDD[8], RED is also a minimum canonical form with respect to a given variable ordering. It is also efficient for representing unions of zones. Experiments have been carried out to investigate its potential for verification efficiency.

Compared to the classic DBM [6, 10, 11, 12, 13, 15], RED provides data-sharing capability of fully symbolic manipulation. In a DBM-based model-checker, since matrices and BDD are two different types of data-structure, we are forced to use a pair of BDD and matrix to represent a region. As a result, identical interaction pattern between BDD and matrix may replicate in different nodes of the region graph representation. Such replication not only increases space complexity but also incurs duplicate effort in region processing. Moreover, to get region canonical representations, DBM-technology usually resorts to the processing of convex hulls which are equivalent to conjunctions of clock inequalities. Thus it may be necessary to break a big disjunction down to exponentially many conjunctions. Such breaking-down can be a source of inefficiency. But since the present RED algorithms derive time step next-state RED's at very tiny steps, DBM may have better verification performance with systems with big timing constants.

Newer technologies of NDD[1] and CDD[7] use binary inequalities of the form $x - y \leq c$. NDD uses binary encoding for the possible values of c while CDD uses multiple value-ranges to record them. However, the number of variables in their decision diagram is likely to be quadratic to the number of clocks used in the systems. The number of variables used in our RED technology, on the other hand, is $O(|X|\log|X|)$ with X as the clock set.

*The work is partially supported by NSC, Taiwan, ROC under grant NSC 89-2213-E-001-002.

A previous version of RED was reported in [14] which takes advantage of symmetry of multiprocess systems to use only $O(|X|)$ binary variables in RED. But its restriction in [14] is that no global clocks can be used and for each process, only one local clock is allowed. Although it is more restricted, the technique reported in [14] indeed can prevail in performance in its target systems. In the future, we are looking forward to incorporating that technique[14] in our future RED tools.

Here is our presentation plan. Section 2 briefly defines timed automata as our model for discussion. Section 3 formally presents our data-structure scheme. Section 4 shows how to derive next-state RED's with symbolic manipulations on RED's. Section 5 experiments with Fischer's protocol[3], CSMA/CD[16], and FDDI[5, 11] protocols to examine the potential of RED.

2 Timed automata

We use the widely accepted model of *timed automata*[2] to explain idea. We assume familiarity with this model and will not go into much detail due to the page-limit.

A *timed automaton* is a finite-state automaton equipped with a finite set of clocks which can hold nonnegative real-values. It is structured as a directed graph whose nodes are *modes (control locations)* and whose arcs are *transitions*. The modes are labeled with *invariance conditions* while the transitions are labeled with *triggering conditions* and a set of clocks to be reset during the transitions. The invariance conditions and triggering conditions are Boolean combinations of inequalities comparing a clock with an integer. At any moment, the timed automaton can stay in only one mode. 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 clocks in the corresponding transition clock set label to zero. In between transitions, all clocks increase their readings at a uniform rate.

For convenience, given a set X of clocks, we use $B(X)$ as the set of all Boolean combinations of inequalities of the form $x \sim c$ or $x - y \sim c$ where $x, y \in X$, "$\sim$" is one of $\leq, <, =, >, \geq$, and c is an integer constant.

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

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

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

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

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

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

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

A safety requirement on timed automaton A can be written as a Boolean combination of clock constraints in $B(X)$ and mode restrictions in the form of $\texttt{mode} = q$ meaning that A is currently in mode $q \in Q$. A run

$\rho = (\nu_0, t_0)(\nu_1, t_1) \ldots (\nu_k, t_k) \ldots\ldots$ of A satisfies *safety* requirement η, in symbols $\rho \models \eta$, iff for all $k \geq 0$ and $t_k \leq t \leq t_{k+1}$, $\nu_k + t \models \eta$. We say $A \models \eta$ iff for all ν-runs ρ, $\nu \models I$ implies $\rho \models \eta$. The *safety analysis problem* of A and η is to answer whether $A \models \eta$.

3 Region-Encoding Diagram

Notationally we let C_A be the biggest integer constant used to compare with clocks in the invariance conditions, initial condition, triggering conditions, and safety requirement of a safety analysis problem instance regarding timed automaton A. According to Alur et al's region graph construction [2], the state-equivalence relation for timed automaton model-checking [9] is determined by the following three factors:

- the discrete information of each state,
- the integer parts of clock readings $\leq C_A$, and
- the ordering among the fractional parts of clock readings $\leq C_A$.

Here we shall use an integer sequence to record the ordering among the fractional parts of clock readings in a state. Given an automaton A with clock set X, the variable set in the RED's is

$$VARS_A = \{\texttt{mode}\} \cup X \cup \{\kappa\langle x \rangle \mid x \in X\}$$

The integer sequence is kept record in the special auxiliary variables $\kappa\langle\rangle$'s for all clocks. Given a state ν, its *region encoding*, in symbols $[\nu]$, is a partial mapping from $VARS_A$ such that $[\nu](\texttt{mode}) = \nu(\texttt{mode})$ and for each clock $x \in X$, the following restrictions are satisfied.

- If $\nu(x) > C_A$, then $[\nu](x) = \infty$.
- If $\nu(x) \leq C_A$, then $[\nu](x) = \lfloor \nu(x) \rfloor$. Furthermore, the following restrictions are satisfied on $[\nu](\kappa\langle x \rangle)$. For convenience, let $frac(d) = d - \lfloor d \rfloor$ be a notation for the fractional part of any $d \in \mathcal{R}^+$.
  - If $\nu(x)$ is an integer, then $[\nu](\kappa\langle x \rangle) = 0$.
  - If $\nu(x)$ is fractional and $[\nu](\kappa\langle x \rangle) = d \in \{1, \ldots, |X|\}$, then $frac(\nu(x))$ is the d'th smallest among all fractional parts of such clocks. Formally speaking, if the set $\{frac(\nu(y)) \mid y \in X; \nu(y) \leq C_A; frac(\nu(y)) \neq 0\}$ is equal to $\{r_1, r_2, \ldots, r_k\}$ with $r_1 < r_2 < \ldots < r_k$ and $frac(\nu(x)) = r_h$ with $1 \leq h \leq k$, then $[\nu](\kappa\langle x \rangle) = h$.

For example, we may have $X = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8\}$ in a state ν, with $\nu(x_1) = 1.456, \nu(x_2) = 3, \nu(x_3) = 38, \nu(x_4) = 1.3, \nu(x_5) = 9.456, \nu(x_6) = 20.7, \nu(x_7) = 0, \nu(x_8) = 10\pi$ while $C_A = 13$. The readings of clocks and values of $\kappa\langle x_i \rangle$'s are shown in the following.

i	1	2	3	4	5	6	7	8
$\nu(x_i)$	1.456	3	38	1.3	9.456	20.7	0	10π
$[\nu](x_i)$	1.456	3	∞	1.3	9.456	∞	0	∞
$[\nu](\kappa\langle x_i \rangle)$	2	0	–	1	2	–	0	–

Note that $[\nu](\kappa\langle x_1 \rangle) = [\nu](\kappa\langle x_5 \rangle) = 2$ while $[\nu](\kappa\langle x_2 \rangle) = [\nu](\kappa\langle x_7 \rangle) = [\nu](\kappa\langle x_8 \rangle) = 0$. "–" means "don't care" or "no restriction." Given any ν, $[\nu]$ keeps the minimal information necessary for model-checking timed automata. RED's can be viewed as decision diagrams for the $[\nu]$'s instead of ν's. In the implementation aspect, it resembles CDD[7] and each node in RED has the structure shown in figure 1(a). Such a node is used to evaluate the truth value of a formulus from variable $v \in VARS_A$. The outgoing arcs are labeled with lower and upper bounds of **integer parts** (note, only integer parts) of values of variable v and direct to the RED's for the subformulae true in the corresponding ranges of v's values. In figure 1(b), we have depicted the RED with $X = \{x_1, x_2\}$ for

$$(\texttt{mode} = 0 \wedge x_1 = 0 \wedge 0 < x_2 < 1) \vee (\texttt{mode} = 1 \wedge 0 < x_2 \wedge x_2 < x_1 \wedge x_1 < 1)$$

Given a timed automaton, each of the $\kappa\langle\rangle$ variables needs $\log(|X| + 1)$ bits. Thus the variables in our RED together may need $O(\log(|Q|) + |X|(\log(|X| + 1) + \log(|C_A| + 2))$ bits.

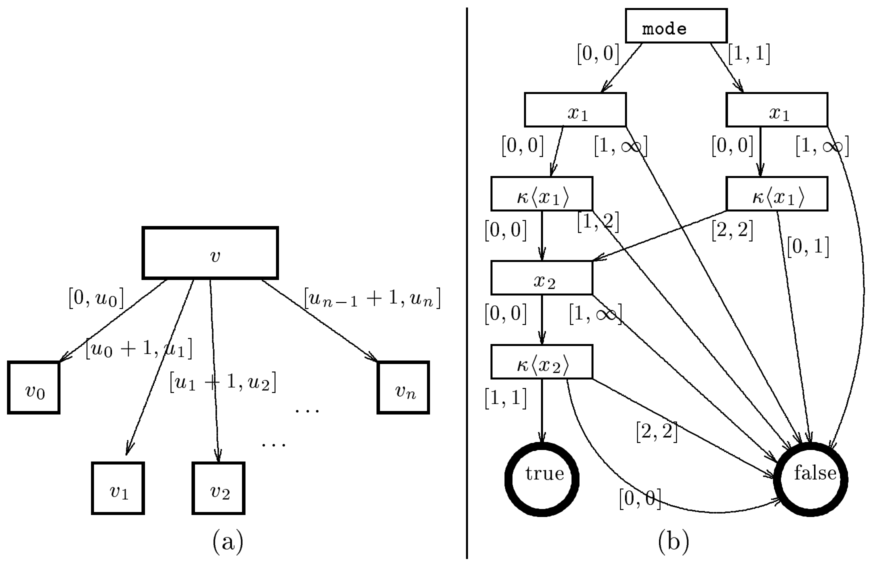

Figure 1: Data structure implementation of a node in RED

4 Manipulations on RED

4.1 Boolean operations

The Boolean operations on RED's follow the same style in Bryant's BDD manipulations [4, 7, 8]. We present procedure $\texttt{RedOp}(\mathbf{OP}, D_1, D_2)$ in table 1 to illustrate the idea in the implementation of such operations. For convenience, we use $[l,u]D$ to denote an outgoing arc whose lower bound is l, upper bound is u, and the subformulus RED is D. Also, $D.v$ denotes the index of D's variable in the variable-ordering of RED's.

We should point out that the algorithm shown in table 1 is for simplicity and clarity of presentation and is not for efficiency. Our implementation is more efficient in that it records which pairs of D_1, D_2 have already been processed. If a pair of D_1, D_2 has already been processed with procedure $\texttt{RedOp}()$ before, then we simply return the result recorded in the first time when such a pair was processed.

4.2 Manipulations for state-transitions

We use Boolean operations on RED's to derive next-state RED's with time steps and discrete transitions. The details of manipulations vary according to properties of the regions. The technique we developed is called *precise slicing* and *slice processing*. Conceptually, with *precise slicing*, we use various Boolean conjunctions to slice a target Boolean formulus into many slices each with different properties suitable for simple RED processing. After the individual slice processing, we then disjunct the processed slices together to form the next-state RED.

Due to page-limit, we shall only discuss the manipulations to derive time-step next-state RED's. There are two kind of time steps for a state ν in a region graph. We make a case analysis in the following. Suppose the next state is ν'.

1. If $\nu(x) > C_A$ (i.e., $[\nu](x) = \infty$) for all clocks x, then $[\nu'] = [\nu]$.
2. If there is a clock x such that $\nu(x)$ is an integer $\leq C_A$ (i.e., $[\nu](x) \in [0, C_A] \wedge [\nu](\kappa\langle x\rangle) = 0$), then the next time step will advance all such clocks to become fractional according to the following rules.
   (a) In case there are some clocks x with integer reading $\nu(x) < C_A$ (i.e., $[\nu](x) \in [0, C_A - 1] \wedge [\nu](\kappa\langle x\rangle) = 0$), such clocks will have their $\kappa\langle x\rangle$ advance from zero to one and push every other clocks' $\kappa\langle\rangle$ value to increment by one in ν'.

```
RedOp($\mathbf{OP}, D_1, D_2$){
  if $\mathbf{OP}=$ $\mathbf{AND}$,
    if $D_1$ is *true*, return $D_2$;
    else if $D_2$ is *true*, return $D_1$;
    else if either $D_1$ or $D_2$ is *false*, return *false*;
  else if $\mathbf{OP}=$ $\mathbf{OR}$,
    if either $D_1$ or $D_2$ is *true*, return *true*;
    else if $D_1$ is *false*, return $D_2$;
    else if $D_2$ is *false*, return $D_1$;
  else {
    Construct a new RED node $D$ with $D.v = \min(D_1.v, D_2.v)$;
    if $D_1.v < D_2.v$, then for each outgoing arc $[l,u]D'_1$ of $D_1$,
      add an outgoing arc $[l,u]$RedOp($\mathbf{OP}, D'_1, D_2$) to $D$.
    else if $D_1.v > D_2.v$, then for each outgoing arc $[l,u]D'_2$ of $D_2$,
      add an outgoing arc $[l,u]$RedOp($\mathbf{OP}, D_1, D'_2$) to $D$.
    else for each outgoing arc $[l_1,u_1]D'_1$ of $D_1$ and outgoing arc $[l_2,u_2]D'_2$ of
$D_2$,
      if $[\max(l_1,l_2), \min(u_1,u_2)]$ is nonempty, add an outgoing arc
        $[\max(l_1,l_2), \min(u_1,u_2)]$RedOp(OP, $D'_1, D'_2$) to $D$.
    Merge any two outgoing arcs $[l,u]D', [u+1,u']D'$ of $D$ into one $[l,u']D'$
      until no more merge can be done.
    if $D$ has more than one outgoing arcs, return $D$,
      else return the sole subformulus of $D$;
  }
} /* RedOp() */
```

Table 1: Algorithm for computing $D_1 \mathbf{OP}\ D_2$

   i. All clocks x with $[\nu](x) = C_A$ will have $[\nu'](x) = \infty$ with $[\nu'](\kappa\langle x\rangle)$ unrestricted.
   ii. All clocks x with $[\nu](x) < C_A$ will have $[\nu'](x) = [\nu](x)$ and $[\nu'](\kappa\langle x\rangle) = [\nu](\kappa\langle x\rangle) + 1$.

 (b) In case that all clocks x with integer readings $\leq C_A$ also satisfy $[\nu](x) = C_A$, then
   i. $[\nu'](x) = \infty$ for all such clocks x and
   ii. the valuations regarding all other clocks and their $\kappa\langle\rangle$ values in $[\nu']$ remain identical to the ones in $[\nu]$.

3. Suppose there is no clock with integer reading $\leq C_A$. For all those clocks x such that $[\nu](x) \leq C_A$ and $[\nu](\kappa\langle x\rangle) = \max\{[\nu](\kappa\langle y\rangle) \mid [\nu](y) \leq C_A\}$, $[\nu'](x) = [\nu](x) + 1$ and $[\nu'](\kappa\langle x\rangle) = 0$. For all other clocks x, the valuations regarding x and their $\kappa\langle\rangle$'s in $[\nu']$ remain identical to those in $[\nu]$.

In table 2, we show the pseudo code for our procedure to derive the RED by advancing regions with integer clock readings $\leq C_A$ according to case 2. Here FILTER_EXISTS_SMALLER_THAN_C_A, FILTER_ONLY_C_A, and RED_INVARIANCE are the RED's we constructed in the initialization phase of the program which are used to slice out the proper RED's for the corresponding region subsets. Due to page-limit, we can only elaborate on how procedure NextFracExistsSmallerThanC_A() properly processes RED slices in table 3. But other procedures are constructed on similar techniques. As can be seen in table 3, we iteratively process for each clock $x \in X$. For each clock x, we break it down to three cases. For each case, we then make necessary filter for precise slicing of RED's and then use proper slice processing procedures. Here we use shorthands like RedOp($\mathbf{AND}, D, x = C_A$) to represent the RED obtained by conjuncting RED D and the RED of $x = C_A$ together. RedVariableEliminate(D, x) is an RED-slice processing procedure which remove every recording about variable x from D.

```
ToFrac($D$) {
  /* case 2(a): precisely slice out the RED with integer reading clocks $< C_A$
*/
  $S$ := RedOp(AND, $D$, FILTER_EXISTS_SMALLER_THAN_$C_A$);

  /* process the slice to advance the $\kappa\langle x\rangle$'s of all clocks $x$ with reading $< C_A$.
*/
  /* process the slice to advance all clocks $x$, with readings $= C_A$, to $O_A$. */
  $R$ := NextFracExistsSmallerThan$C_A$($S$);

  /* case 2(b): precisely slice out the RED with all integer clocks $= C_A$ */
  $S$ := RedOp(AND, $D$, FILTER_ONLY_$C_A$);

  /* process the slice to advance all clocks $x$, with readings $= C_A$, to $O_A$. */
  $S$ := NextFracOnly$C_A$($S$);

  /* union both processed slices. */
  $R$ := RedOp(OR, $R$, $S$);

  /* only the slice satisfying invariance condition is needed. */
  $R$ := RedOp(AND, RED_INVARIANCE, $R$);

  return $R$;
} /* ToFrac() */
```

Table 2: **procedures for calculating the RED after time step from integer clock reading regions**

5 Implementation and experiments

We have implemented the idea in a software called `red` which supports the verification of real-time systems with multiprocesses, pointer data-structures, and synchronizations (synchronous send and receive) from one process to another. The tools will soon be available at:

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

after the submission. The current version of `red` tool was modified from our previous verification tool announced in [14]. Each process can use *global* and *local* variables of type *clock*, *discrete*, and *pointer*. Pointer variables either contain value NULL or the identifiers of processes. Thus in our representation, we allow complicate dynamic networks to be constructed with pointers.

We have tested RED with three previously published benchmarks: CSMA/CD [16], FDDI [11, 5], and Fischer's timed mutual exclusion protocol[3]. The performance is listed in the following table.

	# processes	2	3	4	5	6	7	8	9	10	11	12
CSMA/CD	time	1.25	11.6	51.9	220	600	1302	2451	4421	11037	16356	N/A
	space	55	296	822	2378	5258	9426	14802	22241	38208	58612	
FDDI	time	8.71	75	446	1903	6683	20249	54675	140101	N/A	N/A	N/A
	space	168	583	1900	5039	11551	23783	44921	79192			
Fischer's	time	0.8	7.96	70.9	582	4092	23338	N/A	N/A	N/A	N/A	N/A
	space	23	125	689	3733	16985	68306					
Fischer's	time	0.73	5.11	24.5	89.7	281	775	1911	4354	9213	18817	36204
(w. symm.)	space	21	83	251	643	1401	2716	4837	7994	12730	20731	33152

The CSMA/CD and FDDI benchmarks are executed on a Pentium II 366 MHz IBM notebook with 256 MB memory (real plus swap) running Linux while the Fischer's protocol is executed on a 296 MHz Sun Ultrasparc-II with 1 GB memory running Sun OS with single-thread. The CPU time is measured in seconds. The space is measured in kilobytes and only includes those for RED's and 2-3 trees. The running time can be reduced to about half for smaller concurrencies if we disable some of the garbage-collection function implemented in our tool.

```
NextFracExistsSmallerThan$C_A(D)$ {
    /* iteratively process $d$ for each clock $x$ */
    for every $x \in X$, do {
        /* Case one, $x = C_A$. */
        $R :=$ RedOp($\mathbf{AND}, D, x = C_A$);
        $R :=$ RedVariableEliminate(RedVariableEliminate($R, x$), $\kappa\langle x\rangle$);

        /* Case two, clock $x$ is an integer $< C_A$ or a frac $< C_A$. */
        /* Process with iterating for each possible value of $\kappa\langle x\rangle$ */
        for every $0 \leq i < |X|$, do {
            $S :=$ RedOp($\mathbf{AND}, D, 0 \leq x < C_A \wedge \kappa\langle x\rangle = i$);
            $S :=$ RedOp($\mathbf{AND}$, RedVariableEliminate($S, \kappa\langle x\rangle$), $\kappa\langle x\rangle = i+1$);
            $S :=$ RedOp($\mathbf{OR}, R, S$);
        }

        /* Case three, clock $x$ is overbound. */
        $S :=$ RedOp($\mathbf{OR}, D, x > C_A$);

        $D :=$ RedOp($\mathbf{OR}, R, S$);
    }
    return $D$;
} /* tt NextFracExistsSmallerThan$C_A$() */
```

Table 3: Algorithm to derive time-step next state RED when there is some integer clock readings $< C_A$

CSMA/CD[16] is a mutual-exclusion protocol used in broadcast network based on collision-detection capability of processes. Each process uses only one local clock and there is no global variables of any types. There are a network process and many sender processes. Processes talk to the network process through synchronizers. All processes have three modes. The input file for CSMA/CD with two sender processes is given in Appendix A. The property to be verified is that at any moment, no two processes can both be in transmission mode with local clock reading ≥ 1 (meaning passing the collision-detection period).

FDDI [11, 5] is the token-passing protocol in a ring network. We need one process to model the network and the other processes to model the stations. The input for FDDI with two station processes is given in appendix B. The number of modes of the network process is two times the number of station processes. The number of modes of each station process is three. For each station process, two local clocks are needed. Thus FDDI is not possible to be naturally verified by our previous version of RED tool [14]. Each station process can use the token to transmit message in mandatory synchronous mode and 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 second local clock y is used to accumulate the time used by all processes in one token-passing cycle of the network.

We choose to run the Balarin's version of Fischer's timed mutual exclusion protocol benchmark [3, 13, 14, 15]. The input of two processes to `red` is in appendix C. The protocol relies on a global lock to control access to the critical section. The property to be safety analyzed is at any moment, no more than two processes are in the critical section. We collected two sets of data. One is with plain RED technology (the 3rd congregate row) while the other is with a simple symmetry technique which sorts processes in a region according to clock readings and their $\kappa\langle\rangle$ values (the 4th congregate row). As can be seen from the table, the symmetry technique very much alleviate the factorial explosion in the many different ordering among the $\kappa\langle\rangle$ values. Actually with the vast memory space of 1 GB, we expect that `red` passes the benchmark with concurrency size beyond 16. However, the version of the symmetry technique is still not applicable to protocols with synchronizers.

For all the benchmarks, we observe that the exponential base for the growing rate of both time and space complexities with respect to concurrency sizes decreases. For example, for the CSMA/CD benchmark, when we go from five senders to six senders, the space complexity increases to 2.21 times. But when we go from ten processes to eleven processes, the complexities only increases to 1.53 times. Together with the experiments in [14], we believe such pattern is typical in BDD-style fully symbolic manipulation and may imply that RED technology is very suitable for high-concurrency verification task.

6 Conclusion

We propose to extend the RED technology reported in [14] to verify systems with global clocks and multiple local clocks. Preliminary experiment data shows promise with the intense data-sharing capability of RED technology. We see much work can be done to enhance the performance of RED. Some possibilities follow.

- The way to encode the ordering among clock reading fractional parts in [14] is not possible in the version of `red` in this manuscript. As can be seen, the total number of bits of variables in [14] is $O(|X|)$ while it is $O(|X|\log|X|)$ with in the RED version in this paper. Thus for single local clock systems, the technique reported in [14] should prevail in performance. In the future, we hope to combine both result in the same tool.
- For FDDI benchmarks, we are using smaller timing constants because RED-technology in its current stage is very sensitive to the magnitude of timing constants in performance. This is due to that our time step algorithm now advance the regions in very tiny steps. We feel it possible to develop better algorithms to make `red` stride in much larger time steps.

References

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

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

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

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

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

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

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

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

[9] E. Clarke and E.A. Emerson. Design and Synthesis of Synchronization Skeletons using Branching-Time Temporal Logic, Proceedings of Workshop on Logic of Programs, Lecture Notes in Computer Science 131, Springer-Verlag, 1981.

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

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

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

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

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

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

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

A CSMA/CD with two sender processes

```
process count = 3; /* 1 is for bus, the others for stations. */
local clock x;
global synchronizer
begin1, end1, cd1, busy1,
begin2, end2, cd2, busy2;

mode bus_idle true {
when ?begin1 true  may x:= 0; goto bus_active;
when ?begin2 true  may x:= 0; goto bus_active;
}

mode bus_active true {
when ?end1 true  may x:= 0; goto bus_idle;
when !busy1 x => 1 may ;
when ?begin1 x < 1  may x:= 0; goto bus_collision;

when ?end2 true  may x:= 0; goto bus_idle;
when !busy2 x => 1 may ;
when ?begin2 x < 1  may x:= 0; goto bus_collision;
}

mode bus_collision x < 1 {
when !cd1 !cd2 x < 1  may x:= 0; goto bus_idle;
}


mode sender_wait1 true {
when !begin1 true may x:= 0; goto sender_transm1;
when ?cd1 true may x:= 0;
when ?cd1 true may x:= 0; goto sender_retry1;
when ?busy1 true may x:= 0; goto sender_retry1;
}

mode sender_transm1 x =< 1 {
when !end1 x=1 may x:= 0; goto sender_wait1;
when ?cd1 x<1 may x:= 0; goto sender_retry1;
}

mode sender_retry1 x < 2 {
when !begin1 x < 2 may x:= 0; goto sender_transm1;
when ?cd1 x < 2 may x:= 0;
}


mode sender_wait2 true {
when !begin2 true may x:= 0; goto sender_transm2;
when ?cd2 true may x:= 0;
when ?cd2 true may x:= 0; goto sender_retry2;
when ?busy2 true may x:= 0; goto sender_retry2;
}
```

```
mode sender_transm2 x =< 1 {
when !end2 x=1 may x:= 0; goto sender_wait2;
when ?cd2 x<1 may x:= 0; goto sender_retry2;
}

mode sender_retry2 x < 2 {
when !begin2 x < 2 may x:= 0; goto sender_transm2;
when ?cd2 x < 2 may x:= 0;
}



initially
    bus_idle[1] and x[1] = 0
and sender_wait1[2] and x[2] = 0
and sender_wait2[3] and x[3] = 0;


risk
  sender_transm1[2] and x[2] => 1
and  sender_transm2[3] and x[3] => 1;
```

B FDDI with two station processes

```
process count = 3; /* 3 is for ring, the others for stations. */
local clock x, y;
global synchronizer tt1, tt2, rt1, rt2;

mode ring_to_1 x =< 0 {
when !tt1 x = 0  may goto ring_1;
}

mode ring_1 true {
when ?rt1 true  may x:= 0; goto ring_to_2;
}

mode ring_to_2 x =< 0 {
when !tt2 x = 0  may goto ring_2;
}

mode ring_2 true {
when ?rt2 true  may x:= 0; goto ring_to_1;
}

mode station_1_idle true {
when ?tt1 true may y:= x; x:= 0; goto station_1_sync;
}

mode station_1_sync x < 1 {
when !rt1 x < 1 and y => 1 may goto station_1_idle;
when x < 1 and y < 1 may goto station_1_async;
}

mode station_1_async y =< 1 {
when !rt1 true may goto station_1_idle;
}

mode station_2_idle true {
when ?tt2 true may y:= x; x:= 0; goto station_2_sync;
}

mode station_2_sync x < 1 {
when !rt2 x < 1 and y => 1 may goto station_2_idle;
when x < 1 and y < 1 may goto station_2_async;
}

mode station_2_async y =< 1 {
when !rt2 true may goto station_2_idle;
}

initially
    station_1_idle[1] and x[1] = 0 and y[1] => 1
and station_2_idle[2] and x[2] = 0 and y[2] => 1
and ring_to_1[3] and x[3] = 0 and y[3] => 1;

risk
  (station_1_sync[1] or station_1_sync[1])
and  (station_2_sync[2] or station_2_sync[2]);
```

C Fischer's timed mutual exclusion protocol

```
/* Fischer's protocol with 2 processes */
process count = 2;

global pointer lock;

local clock x;

mode idle true {
  when lock = null may x:= 0; goto ready;
}

mode ready true {
  when x < 1 may x:= 0; lock:= P; goto waiting;
}

mode waiting true {
  when (x > 1 and lock = P) may goto critical;
  when lock != P may goto idle;
}

mode critical true {
  when true may lock := null; goto idle;
}

initially lock = null and idle[1] and x[1] = 0 and idle[2] and x[2] = 0;

risk critical[1] and critical[2];
```