Symbolic Parametric Safety Analysis of Linear Hybrid Systems with BDD-Like Data-Structures

Farn Wang, Member, IEEE Computer Society

Abstract—We introduce a new BDD-like data structure called *Hybrid-Restriction Diagrams (HRDs)* for the representation and manipulation of linear hybrid automata (LHA) state-spaces and present algorithms for weakest precondition calculations. This permits us to reason about the valuations of parameters that make safety properties satisfied. Advantages of our approach include the ability to represent discrete state information and concave polyhedra in a unified scheme, as well as to save both memory consumptions and manipulation times when processing the same substructures in state-space representations. Our experimental results document its efficiency in practice.

Index Terms—Data-structures, BDD, hybrid automata, verification, model-checking.

1 Introduction

Linear hybrid automata (LHA) are state-transition systems equipped with continuous variables that can change values with different rates [7]. They are important to computing and society because of their extensive modeling capability. In practice, such models are usually presented with symbolic constants, called *parameters*, whose values may engender different behaviors of the models. Setting and calibrating these parameters is a crucial task for the engineers developing hybrid systems. *Parametric safety analysis (PSA)*[1] of LHA can generate a symbolic characterization of the parameters' valuations, called *solutions*, that make a model satisfy a *safety property*. For example, we may be interested at the constraint that relates the parameter for time-out values and the parameter for number of message retransmissions in a telecommunication protocol. We may also be interested to know the constraint that relates the sampling period and error propagation in an air traffic control system. Such symbolic characterizations, once constructed, shed important feedback information to engineers and can be used repeatedly in the synthesis and verification of implementations of the corresponding models.

Although the emptiness problem of the PSA solution spaces is undecidable in general [8], people have constructed experimental tools for the verification of LHA [1], [3], [4], [7] based on the following three reasons: First, the verification problem is by itself interesting and challenging. Second, it may happen that there exist techniques and strategies which can solve many typical examples. Third, there is still an urgent need in industry for such tools to help engineers with fast prototyping of their complex designs.

Because LHA necessarily involves continuous variables, it is helpful and more efficient to represent and manipulate their state-space symbolically. Others have developed separate representation schemes for the discrete part and continuous part of LHA [1], [3], [4], [7]. According to previous experiences [10], [12], [23], [24], [25], it can be more efficient to have a unified data-structure for both the discrete part and the continuous part. Moreover, the schemes used in [1], [3], [4], [7] only represent convex polyhedra in LHA state-space (concave polyhedra have to be represented as sets of convex ones) and suffer from the limitation to share common representation structures in different convex polyhedra.

In this work, we extend *BDD* [10], [12] for the representation and manipulation of LHA state-spaces. A BDD [12] is an acyclic graph with a single source and two sinks for $true$ and $false$, respectively. Each internal node is labeled with a Boolean variable and has a $true$-child and a $false$-child. A BDD works as a *decision diagram* for state-space membership. Any sequence of variables labeled on a path in a BDD must follow a predefined total ordering.

The term *BDD-like data-structure* is coined in [25] to call the many BDD extensions in the literature. BDD-like data-structures have the advantage of data-sharing in both representation and manipulation and have shown great success in VLSI verification industry. The same structure following two decision path prefixes will only be represented once. Also, a pattern of operand substructures will only be manipulated once. The manipulation result will be saved and returned immediately the next time when the same pattern is encountered. These features of BDD-technology lead to not only savings in memory consumptions, but also speed-up in manipulations.

One of the major difficulties to use BDD-like data-structures to analyze LHAs comes from the unboundedness of the dense variable value ranges and the unboundedness of linear constraints. To explain one of the major contribution of this work, we need to discuss the following issue

1. PSA here means the process of deriving the constraints on the parameters in a design for the fulfillment of the specification.

• *The author is with the Department of Electrical Engineering, National Taiwan University, 1, Sec. 4, Roosevelt Rd., Taipei, Taiwan 106, ROC. E-mail: farn@cc.ee.ntu.edu.tw.*

Manuscript received 11 June 2004; revised 22 Nov. 2004; accepted 3 Jan. 2005; published online 29 Jan. 2005.
Recommended for acceptance by R. Lutz.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number TSE-0111-0604.

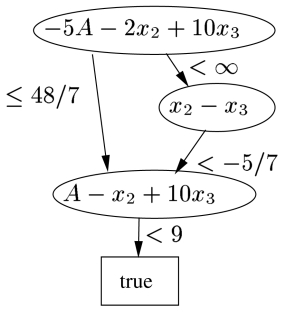

Fig. 1. An example of HRD.

first. In the research of BDD-like data-structures, there are two classes of variables: *system variables* and *decision atoms* [25]. System variables are those used in the input behavior descriptions. Decision atoms are those labeled on each BDD node. For discrete systems, these two classes are the same. But for dense-time systems, decision atoms can be different from state variables. For example, in CDD (Clock-Difference Diagram) [11] and CRD (Clock-Restriction Diagram) [25], decision atoms are of the form $x - x'$ where x and x' are system variables of type clock. Previous work on BDD-like data-structures are based on the assumption that decision atom domains are of finite sizes. Thus, we need new techniques to extend BDD-like data-structures to represent and manipulate state-spaces of LHAs. Our innovations include using expressions, like $-5A - 2x_2 + 10x_3$ (where A, x_2, x_3 are dense system variables), as the decision atoms and using total dense orderings among these atoms. Fig. 1 is an HRD example for the concave space of

$$\begin{aligned} & (x_2 - x_3 \leq -5/7 \vee -5A - 2x_2 + 10x_3 \leq 48/7) \\ \wedge \quad & A - x_2 + 10x_3 < 9 \end{aligned}$$

assuming that $-5A - 2x_2 + 10x_3$ *precedes* $x_2 - x_3$ (in symbols $-5A - 2x_2 + 10x_3 \prec x_2 - x_3$) and $x_2 - x_3$ *precedes* $A - x_2 + 10x_3$ in the given evaluation ordering. In this example, the system variables are A, x_2, x_3 while the decision atoms are $x_2 - x_3$, $-5A - 2x_2 + 10x_3$, and $A - x_2 + 10x_3$. A node label $\sum_i a_i x_i$ with a corresponding outgoing arc label $\sim c$ constitute the constraint of $\sum_i a_i x_i \sim c$. A source-to-sink path in an HRD thus represents the conjunction of constituent constraints along the path. An HRD represents the union of the convex state-spaces of the respective source-to-sink paths. Or, equivalently, we may view an HRD as the set of convex polyhedra of those respective source-to-sink paths. In this way, we devise *HRD (Hybrid-Restriction Diagram)* and successfully extend BDD-technology to models with unbounded domains of decision atoms.

In total, we defined three total dense orderings for HRD decision atoms (Section 6). We also present algorithms for set-oriented operations (Section 7) and symbolic weakest precondition calculation (Section 9), and a procedure for symbolic parametric safety analysis (Section 3).

HRD is for the symbolic representation and manipulation of embedded systems with dense variables. In real-world designs, it is likely that discrete variables, like operation modes, message retransmission counts, are also used. Such discrete variables can be represented and manipulated more efficiently with BDD than with HRD. We follow the approach in [25] to combine BDD and HRD into a new data structure called *HRD+BDD* so that we can use both discrete variables and dense constraints in the same BDD-like data-structure. Details will be given in Section 8.

Like CDD and CRD, HRD itself does not enforce a canonical form in representing state-spaces. But, the calculation of canonical representation of state-spaces for LHA is very expensive. In this work, we have carefully adopted a compromise that will normalize the representations of concave polyhedra in an HRD, but will not overdo the normalization to blow up the resource consumptions. Our normalization comes in two ways. First, we eliminate some redundancies along each HRD path that represents a convex polyhedron. Second, we adapt the *straightforward containment requirement* in [25] to eliminate those convex polyhedra that are contained by peers in the same HRD. In a sense, the first approach tries to contain the depths of HRDs while the second does the widths. Details can be found in Section 10.

We have also developed a technique to eliminate state-space exploration irrelevant to the answer of parametric safety analysis. Desirably, this technique enhances the performance of our tool by orders of magnitude (Section 11) and does not sacrifice the precision of parametric safety analysis. To our knowledge, nobody else has come up with a similar technique. Finally, we have implemented our ideas in our tool **RED** 5.3 and reported our experiments to see how the three dense-orderings perform and how our implementation performs in comparison with HyTech 2.4.5 [15] and TReX 1.3 [1], [3].

2 Parametric Safety Analysis of Linear Hybrid Automata

A *linear hybrid automaton (LHA)* [7] is a finite-state automaton equipped with a finite set of dense system variables which can hold real-values. At any moment, the LHA 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 LHA instantaneously transits from one mode to another and sets some dense variables to values in certain ranges. In between transitions, all dense variables increase their readings at rates in fixed intervals determined by the current mode.

In Fig. 2, we have drawn a version of the Fischer's mutual exclusion algorithm for a process. There are two parameters α and β that control the behavior of the processes. In each mode, local clock x increases its reading according to a rate in $[4/5, 1]$ or $[1, 11/10]$. The rate intervals in different modes can be different.

For convenience, given a set Q of modes and a set X of dense variables, we use $P(Q, X)$ to denote the set of all Boolean combinations of atoms of the forms q and $\sum a_i x_i \sim c$, where $q \in Q$, a_i are integer constants, $x_i \in X$, "$\sim$" is one of $\leq, <, =, >, \geq$, and c is a rational constant. For convenience, an element in $P(Q, X)$ is called a *state predicate*.

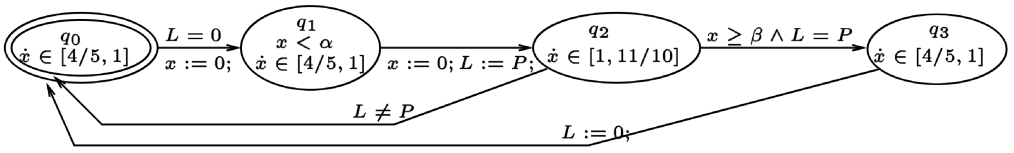

Fig. 2. Fischer's timed mutual exclusion algorithm in LHA.

We let Θ be the set of rational intervals like $\langle d, d' \rangle$ where "$\langle$" is either "[" or "("; "$\rangle$" is either "]" or ")"; and d, d' are $-\infty, \infty$ or rational numbers. Given a set $X = \{x_1, \ldots, x_n\}$ of dense system variables, an *LH-expression (linear hybrid expression)* is an expression like $a_1 x_1 + \ldots + a_n x_n$ where $a_1, \ldots, a_n$ are integer constants. We let $\Lambda(X)$ be the set of all LH-expressions of X.

Definition 1: Linear Hybrid Automata (LHA). *An LHA A is a tuple $\langle X, Q, I, \mu, \gamma, E, \tau, \pi \rangle$ with the following restrictions: X is the set of dense variables. 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. $\gamma : (Q \times X) \mapsto \Theta$ defines the rate intervals of dense variables at each mode. $E \subseteq Q \times Q$ is the set of transitions. $\tau : E \mapsto P(\emptyset, X)$ defines the triggering condition of transitions. To accommodate the various features of LHA in the literature, we allow for two types of assignment statements defined with π, which is a partial function from $E \times X$ to $\Theta \cup \Lambda(X)$. If $\pi(e, x)$ is undefined, x remains unchanged in transition e. If $\pi(e, x) \in \Theta$, x is nondeterministically assigned a finite value in $\pi(e, x)$ in transition e. If $\pi(e, x) \in \Lambda(X)$, x is assigned the value of $\pi(e, x)$ in transition e.*

In Table 1, we list the components in the tuple for the LHA in Fig. 2.

A *valuation* of a set is a mapping from the set to another set. In other words, a valuation of a set is a function whose domain is the set and whose codomain is another. For example, we may have a valuation ν that maps clocks x_1, x_2 in X to reals 0.35 and $11/3$, respectively.

Definition 2 (Satisfaction of state predicate). *Given an $\eta \in P(Q, X)$ and a valuation ν of X, we say ν* satisfies *η, in symbols $\nu \models \eta$, iff η evaluates to true under substitution ν. Formally speaking, a state ν satisfies state predicate η iff*

- $\nu \not\models false$;
- $\nu \models \sum a_i x_i \sim c$ *iff* $\sum a_i \nu(x_i) \sim c$;
- $\nu \models \eta_1 \vee \eta_2$ *iff* $\nu \models \eta_1$ *or* $\nu \models \eta_2$; *and*
- $\nu \models \neg \eta_1$ *iff* $\nu \not\models \eta_1$.

For example, given a valuation ν such that $\nu(x_1) = 0.35$ and $\nu(x_2) = 11/3$, $\nu \models x_1 < 1 \wedge 3x_1 + x_2 > 4$ while $\nu \not\models x_1 < 1 \wedge 3x_1 - x_2 > 4$.

Definition 3: States. *A* state *ν of $A = \langle X, Q, I, \mu, \gamma, E, \tau, \pi \rangle$ is a valuation of $X \cup Q$ such that*

- *there is a unique $q \in Q$ where $\nu(q) = true$ and for all $q' \neq q, \nu(q') = false$;*
- *for each $x \in X$, $\nu(x) \in \mathcal{R}$ (the set of reals) and $\forall q \in Q, \nu(q) \Rightarrow \nu \models \mu(q)$.*

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

For any $t \in \mathcal{R}^+$ (the set of nonnegative reals), $\nu \overset{t}{\rightsquigarrow} \nu'$ iff we can go from ν to ν' merely by the passage of t time units. Formally speaking, $\nu \overset{t}{\rightsquigarrow} \nu'$ is true iff ν' is a state identical to ν except that, for every $x \in X$ with $\gamma(\nu^Q, x) = \langle d, d' \rangle$, $\nu'(x) \in \langle \nu(x) + t \cdot d, \nu(x) + t \cdot d' \rangle$.

For a transition $e \in E$, $\nu \overset{e}{\rightsquigarrow} \nu'$ iff we can go from ν to ν' with discrete transition $e = (q, q')$. Formally speaking, $\nu \overset{e}{\rightsquigarrow} \nu'$ is true iff $\nu^Q = q$, $\nu \models \mu(q) \wedge \tau(e)$, and ν' is identical to ν except that

- $\nu'^Q = q'$ and $\nu' \models \mu(q')$; and
- for each $x \in X$, if $\pi(e, x) \in \Theta$, $\nu'(x) \in \pi(e, x)$; if $\pi(e, x) = \sum_i a_i x_i \in \Lambda(X)$, $\nu'(x) = \sum_i a_i \nu(x_i)$; otherwise, $\nu'(x) = \nu(x)$.

Definition 4: Runs. *Given an LHA $A = \langle X, Q, I, \mu, \gamma, E, \tau, \pi \rangle$, a* run *is an infinite sequence of pairs $(\phi_0, t_0)(\phi_1, t_1) \ldots (\phi_k, t_k) \ldots\ldots$ such that $t_0 t_1 \ldots t_k \ldots\ldots$ is a monotonically increasing real-number (time) divergent sequence and for all $k \geq 0$,*

- *ϕ_k is a mapping from $[t_k, t_{k+1}]$ to states, and*
- time-progress is continuous: *that is, for each $t_k \leq t \leq t' \leq t_{k+1}$, $\phi_k \overset{t'-t}{\rightsquigarrow} \phi_k(t')$; and*
- invariance conditions are preserved in each interval: *that is, for all $t_k \leq t \leq t_{k+1}$, $\phi_k(t) \models \mu(\phi_k(t)^Q)$; and*

TABLE 1
LHA Description for Fischer's Timed Mutual Exclusion Algorithm

X	clocks	$\{x\}$
Q	modes	$\{q_0, q_1, q_2, q_3\}$
I	initial condition	q_0
μ	invariance conditions	$\mu(q_0) : true; \mu(q_1) : x < \alpha; \mu(q_2) : true; \mu(q_3) : true$
γ	rates	$\gamma(q_0, x) \in [4/5, 1]; \gamma(q_1, x) \in [4/5, 1]; \gamma(q_2, x) \in [1, 11/10]; \gamma(q_3, x) \in [4/5, 1]$
E	transitions	$\{(q_0, q_1), (q_1, q_2), (q_2, q_0), (q_2, q_3), (q_3, q_0)\}$
τ	triggering conditions	$\tau(q_0, q_1) : L = 0; \tau(q_1, q_2) : true; \tau(q_2, q_0) : L \neq P; \tau(q_2, q_3) : x \geq \beta \wedge L = P; \tau(q_3, q_0) : true;$
π	reset clocks	$\pi(q_0, q_1) : \{x := 0;\}; \pi(q_1, q_2) : \{x := 0; L := P;\}; \pi(q_2, q_0) : \emptyset; \pi(q_2, q_3) : \emptyset; \pi(q_3, q_0) : \{L := 0;\};$

- *either* no transition happens at time t_{k+1}, *that is,* $\phi_k(t_{k+1}) = \phi_{k+1}(t_{k+1})$, *or* a transition e happens at t_{k+1}, *that is,* $\phi_k(t_{k+1}) \overset{e}{\sim} \phi_{k+1}(t_{k+1})$.

A run $\rho = (\phi_0, t_0)(\phi_1, t_1)\ldots(\phi_k, t_k)\ldots\ldots$ is *safe* with regard to a safety state-predicate η, in symbols $\rho \models \eta$, iff for all $k \geq 0$ and $t \in [t_k, t_{k+1}]$, $\phi_k(t) \models \eta$. A dense variable x in an LHA is a *static parameter* iff its rate is always zero in all modes. Suppose H is the set of static parameters in X of LHA A. A *static parameter valuation* $\mathcal{H}$ of a run $(\phi_0, t_0)(\phi_1, t_1)\ldots(\phi_k, t_k)\ldots\ldots$ is a mapping from H to reals such that $\mathcal{H}$ is consistent with every state along ρ, i.e., $\forall k \geq 0 \forall t_k \leq t \leq t_{k+1}(\phi_k(t_k) \models \bigwedge_{x \in H} x = \mathcal{H}(x))$. $\mathcal{H}$ is a *parametric solution* to A and η iff for all runs ρ with static parameter valuation $\mathcal{H}$, $\rho \models \eta$.

Our verification framework is called *parametric safety analysis (PSA) problem*. Such a problem instance, denoted $\mathrm{PSA}(A, \eta)$, consists of an LHA A and a safety state-predicate η and asks for a symbolic characterization of all parametric solutions to A and η. The general parametric safety analysis problem can be proved incomputable with a straightforward adaptation of the undecidability proof in [8].

3 Symbolic PSA of LHA

In the following, we present a symbolic procedure for the parametric safety analysis of LHA. The procedure is symbolic in that each of its statements or subprocedures constructs a logic predicate from some argument predicates. The remainder sections of the manuscript then serve as puzzle pieces to fit into the symbolic procedure. Moreover, in this section, we do not discuss how to implement the logic predicates. In Section 4, we examine the geometry of the logic predicates (i.e., Boolean combinations of linear constraints of LHA dense variables). In Section 5, we then discuss how to use HRD to implement such logic predicates.

Suppose we are given an LHA A. There are two basic procedures in this analysis procedure. The first, $\mathtt{xtion}(D, e)$, computes the weakest precondition from state-space represented by HRD D through discrete transition e from mode q to q'. The second basic procedure, $\mathtt{time}(D, q)$, computes the weakest precondition from D through time passage in mode q. Our implementation of the two basic procedures is explained in Section 9.

Given $\mathrm{PSA}(A, \eta)$, assume that the transition set is E and the unsafe states are in mode q_f. With the two basic procedures, then the backward reachable state-space from the unsafe states in $\neg\eta$ (represented as an HRD) can be characterized by

$$\mathtt{lfp}Z.\left(\mathtt{time}(\neg\eta, q_f) \cup \bigcup_{e=(q,q') \in E} \mathtt{time}(\mathtt{xtion}(Z, e), q)\right).$$

Here, $\mathtt{lfp}Z.F(Z)$ is the least fixpoint of function $F()$ and is very commonly used in the reachable state-space representation of discrete and dense-time systems. After the fixpoint is successfully constructed, we conjunct it with the initial condition and then make existential quantifications to filter out all variables except those static parameters. Geometrically speaking, the quantifications project the reachable state-space to the dimensions of the static parameters. Suppose the set of static parameters is H and $X - H = \{y_1, \ldots, y_n\}$. The characterization of unsafe parameter valuations is thus

$$\exists y_1 \ldots \exists y_n \left(I \wedge \mathtt{lfp}Z.\left(\begin{array}{l}\mathtt{time}(\neg\eta, q_f) \\ \cup\ \bigcup_{e=(q,q') \in E} \mathtt{time}(\mathtt{xtion}(Z, e), q)\end{array}\right)\right).$$

The solution space is characterized by the complement of this final result. Our parametric safety analysis procedure is as follows:

```
PSA(A, η) {
  D̄ := time(¬η, q_f); D := false;
  while D̄ ≠ false, do {
    D := D ∪ D̄; D̄ := ⋃_{e=(q,q')∈E} time(xtion(D̄, e), q);
    D̄ := D̄ ∧ (¬D);
  }
  return ¬∃y_1 ... ∃y_n(I ∧ D);
}
```

In this general framework, we iteratively construct a logic predicate characterizing the backward reachable state-space from the logic predicate for the risk condition and through the weakest precondition calculation procedures, respectively, for time-passage and discrete transitions.

4 Convex Polyhedra

An LH-expression $\sum_i a_i x_i$ of X is *normalized* iff the gcd (greatest common divisor) of nonzero coefficients in $\{a_1, \ldots, a_n\}$ is 1, i.e., $\gcd\{a_i \mid 1 \leq i \leq n; a_i \neq 0\} = 1$. From now on, we shall assume that all given LH-expressions are normalized.

For efficiency of manipulation, we have the following requirements on constraints used to represent a *convex polyhedron*: Given a constraint like $\sum_i a_i x_i \sim c$, 1) $\sum_i a_i x_i$ is a normalized LH-expression, 2) $\sim \in \{''\leq'', ''<''\}$, and 3) c is either a rational number or ∞ such that when $c = \infty$, $\sim = ''<''$.

An *LH-upperbound* is either $(<, \infty)$ or a pair like $(\sim, c)$, where $\sim \in \{''<'', ''\leq''\}$ and c is a rational number. For any two $(\sim, c)$ and $(\sim', c')$, $(\sim, c)$ is *stricter than* $(\sim', c')$, denoted $(\sim, c) \sqsubset (\sim', c')$, iff $c < c'$ or $(c = c' \wedge \sim = ''<'' \wedge \sim' = ''\leq'')$. $(\sim, c) \sqsubseteq (\sim', c')$ if either $(\sim, c) \sqsubset (\sim', c')$ or $(\sim, c) = (\sim', c')$. We can also define the addition and constant multiplication of LH-upperbounds. Specifically, $(\sim, c) + (\sim', c') = (\leq, c + c')$ when both $\sim$ and $\sim'$ are $''\leq''$ and $(<, c + c')$ otherwise. For any positive integer k, $k(\sim, c) = (\sim, kc)$.

An *LH-constraint* is a pair of an LH-expression and an LH-upperbound. Given an LH-expression $\sum_i a_i x_i$ and an LH-upperbound $(\sim, c)$, we shall naturally write the corresponding LH-constraint as $\sum_i a_i x_i \sim c$.

Formally, a convex polyhedron can be defined as a mapping from the set of LH-expressions to the set of LH-upperbounds. Alternatively, we may also represent a convex polyhedron ζ as the set $\{\sum_i a_i x_i \sim c \mid \zeta(\sum_i a_i x_i) = (\sim, c)\}$. We shall use the two equivalent notations flexibly as we see fit.

For a given X, the set of all LH-expressions and the set of convex polyhedra are both infinite.

5 HRD (HYBRID-RESTRICTION DIAGRAM)

To construct BDD-like data-structures, three fundamental issues have to be solved. The first is the domain of the decision atoms, the second is the range of the arc labels from BDD nodes, and the third is the evaluation ordering among the decision atoms. For modularity of presentation, we shall leave the discussion of the evaluation orderings to Section 6. In this section, we shall assume that we are given a decision atom evaluation ordering.

We decide to follow an approach similar to the one adopted in [25]. That is, the decision atoms of HRD are LH-expressions and the arcs are labeled with LH-upperbounds. A node label $\sum_i a_i x_i$ with a corresponding outgoing arc label $(\sim, c)$ constitute the constraint of $\sum_i a_i x_i \sim c$. A source-to-sink path in an HRD thus represents the conjunction of constituent constraints along the path. An HRD represents the union of the convex state-spaces of the respective source-to-sink paths. Or, equivalently, we may view an HRD as the set of convex polyhedra of those respective source-to-sink paths.

Definition 5: ***HRD (Hybrid-Restriction Diagram).*** *Given a dense variable set $X = \{x_1, \ldots, x_n\}$ and an evaluation ordering $\prec$ among normalized LH-expressions of X, an* HRD *is either true or a tuple $(\epsilon, (\beta_1, D_1), \ldots, (\beta_m, D_m))$ such that*

- *ϵ is a normalized LH-expression;*
- *for each $1 \leq i \leq m$, β_i is an LH-upperbound such that $(<, \infty) \neq \beta_1 \sqsubset \beta_2 \sqsubset \ldots \sqsubset \beta_m$; and*
- *for each $1 \leq i \leq m$, D_i is an HRD such that if $D_i = (\epsilon_i, \ldots)$, then $\epsilon \prec \epsilon_i$.*

For completeness, we use "()" to represent the HRD for $false$.

In our algorithms, $false$ does not participate in comparison of evaluation orderings among decision atoms. Also, note that in Fig. 1, for each arc label $(\sim, c)$, we simply put down $\sim c$ for convenience.

6 THREE DENSE ORDERINGS AMONG DECISION ATOMS

In the definition of a dense-ordering among decision atoms (i.e., LH-expressions), special care must be taken to facilitate efficient manipulation of HRDs. Here, we use the experience reported in [25] and present three heuristics in designing the orderings among LH-expressions.

The three heuristics are presented in the order of their relative importance.

6.1 HEURISTICS I

It is desirable to place a pair of converse LH-expressions next to one another so that simple inconsistencies can be easily detected. That is, LH-expressions $\sum_i a_i x_i$ and $\sum_i -a_i x_i$ are better placed next to one another in the ordering. With this arrangement, inconsistencies like $-x_1 + 3x_2 \leq -5 \wedge x_1 - 3x_2 < 0$ can be checked by comparing adjacent nodes in HRD paths. Such an inconsistency very often happens after transitivity deductions from constraints of a convex polyhedron. Without the heuristics, we may have to traverse long in an HRD path to detect such inconsistencies.

To fulfill this requirement, when comparing the precedence between LH-expressions in a given ordering, we shall first toggle the signs of coefficients of an LH-expression if its first nonzero coefficient is positive. If two LH-expressions are identical after the necessary toggling, then we compare the signs of their first nonzero coefficients to decide the precedence between the two.

For rigorosity of presentation, we need the following definitions: Given a nonnull LH-expression $\epsilon = \sum_i a_i x_i$, the *leading coefficient* of ϵ is the first nonzero a_i in $a_1, \ldots, a_n$. The *leading sign* of ϵ, in symbols $\mathtt{sign}(\epsilon)$, is the sign of the leading coefficient. Then, the *norm* of ϵ, in symbols $|\epsilon|$, is $\sum_i (-a_i \cdot \mathtt{sign}(\epsilon)) x_i$.

With this heuristics, from now on, we shall only focus on the orderings among the norms of LH-expressions. Only when the norms of two LH-expressions are identical, we resort to the signs of the leading coefficients to decide the ordering between the two LH-expressions.

6.2 HEURISTICS II

According to the literature [14], [17], [19], it is important to place strongly correlated LH-expressions close together in the evaluation orderings. Usually, instead of a single global LHA, we are given a set of communicating LHAs, each representing a process. Thus, it is desirable to place LH-expressions for the same process close to each other in the orderings. Our second heuristics respects this experience. Given a system with m processes with respective local dense variables, we shall partition the LH-expressions into $m+1$ groups: $G_0, G_1, \ldots, G_m$. G_0 contains all LH-expressions without local variables (i.e., coefficients for local variables are all zero). For each $p > 0$, G_p contains all LH-expressions with a nonzero coefficient for a local variable of process p and only zero coefficients for local variables of processes $p+1, \ldots, m$. Then, our second heuristics requires that for all $0 \leq p < m$, LH-expressions in G_p precede those in $G_{p+1}, \ldots, G_m$. For rigorosity of discussion, for all $\epsilon \in G_p$, we denote $G(\epsilon) = p$.

6.3 HEURISTICS III

If the precedence between two LH-expressions cannot be determined with heuristics I and II, then the following third heuristic comes into play. The design of this heuristic is a challenge since each of $G_0, \ldots, G_m$ can be of infinite size. Traditionally, BDD-like data-structures have been used with finite decision atom domains. Carefully examining the BDD manipulation procedures, we found that the sizes of the domains do not matter. What really matters is the ordering among all decision atoms so that we know which atom is to be decided next along a decision path. With a generic procedure $\mathtt{Heuristic}_3()$ to decide the ordering between two LH-expressions in the same group, the procedure to decide the ordering between two LH-expressions is in the following: The procedure $\prec(\epsilon_1, \epsilon_2, \mathtt{Heuristic}_3)$ returns -1 if $\epsilon_1 \prec \epsilon_2$, returns 0 if $\epsilon_1 = \epsilon_2$, and returns 1 if $\epsilon_1 \succ \epsilon_2$.

```
≺ (ε1, ε2, Heuristic3) {
  if G(|ε1|) < G(|ε2|), return −1;
  else if G(|ε1|) > G(|ε2|), return 1;
```

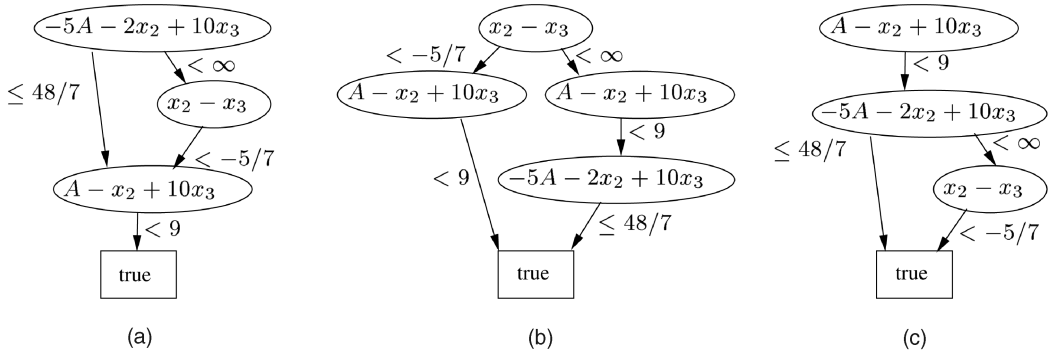

Fig. 3. Examples of HRD evaluation orderings. (a) HRD with coefficient ordering. (b) HRD with magnitude ordering. (c) HRD with dictionary ordering.

$\texttt{comp} := \texttt{Heuristic}_3(|\epsilon_1|, |\epsilon_2|)$;
if $\texttt{comp} \neq 0$, return comp;
else if $\texttt{sign}(\epsilon_1) < \texttt{sign}(\epsilon_2)$, return -1;
else if $\texttt{sign}(\epsilon_1) > \texttt{sign}(\epsilon_2)$, return 1;
else return 0;
}

Thus, we invent the following three dense-orderings among LH-expressions for our use. In Section 13, we shall report experiments with these orderings.

6.3.1 Dictionary Ordering

We can represent each LH-expression as a string, assuming that the ordering among $x_1, \ldots, x_n$ is fixed and no blanks are used in the string. Then, we can use dictionary (i.e., lexicographic) ordering and ASCII ordering to decide the precedence among LH-expressions. Given a character string s, $|s|$ is the length of string s. For each $0 \leq i < |s|$, $s[i]$ is the $i+1$st character in the string. Suppose s_1 and s_2 are the strings for LH-expressions ϵ_1 and ϵ_2, respectively. Then, $\epsilon_1 \prec \epsilon_2$ in the dictionary ordering if there is a $0 \leq i < \min(|s_1|, |s_2|)$ such that $s_1[i] < s_2[i]$ and for all $0 \leq j < i$, $s_1[j] = s_2[j]$. For the LH-expressions in Fig. 1, we then have $-5A - 2x_2 + 10x_3 \prec A - x_2 + 10x_3 \prec x_2 - x_3$ since "$-$" precedes "A" and "A" precedes "x" in ASCII. The HRD for

$$(x_2 - x_3 \leq -5/7 \vee -5A - 2x_2 + 10x_3 \leq 48/7) \wedge$$
$$A - x_2 + 10x_3 < 9$$

in dictionary ordering is in Fig. 3c. One interesting feature of this ordering is that it has the potential to be extended to nonlinear hybrid constraints. For example, we may say $\cos(x_1) + x_2^3 \prec x_2^2 - x_2 x_3$ in dictionary ordering since "c" precedes "x" in ASCII.

6.3.2 Coefficient Ordering

Assume that the ordering of the dense variables is fixed as $x_1, \ldots, x_m$. In this ordering, the precedence between two LH-expressions is determined by iteratively comparing the coefficients of dense variables $x_1, \ldots, x_n$ in sequence. Formally, $\sum_{1 \leq i \leq n} a_i x_i$ precedes $\sum_{1 \leq i \leq n} b_i x_i$ in the coefficient ordering if there is a $1 \leq i \leq n$ such that $a_i < b_i$ and for all $1 \leq j < i$, $a_j = b_j$. For the LH-expressions in Fig. 1, we then have $-5A - 2x_2 + 10x_3 \prec x_2 - x_3 \prec A - x_2 + 10x_3$. An example of HRD in this ordering is in Fig. 3a.

6.3.3 Magnitude Ordering

This ordering is similar to the last one. Instead of comparing coefficients, we compare the absolute values of coefficients. We iteratively

- first compare the absolute values of coefficients of x_i and
- if they are equal, then compare the signs of coefficients of x_i.

Formally, $\sum_{1 \leq i \leq n} a_i x_i$ precedes $\sum_{1 \leq i \leq n} b_i x_i$ in the magnitude ordering if there is a $1 \leq i \leq n$ such that $|a_i| < |b_i| \vee (|a_i| = |b_i| \wedge a_i < b_i)$ and for all $1 \leq j < i$, $a_j = b_j$. For the LH-expressions in Fig. 1, $x_2 - x_3 \prec A - x_2 + 10x_3 \prec -5A - 2x_2 + 10x_3$ in this magnitude ordering. The corresponding HRD is in Fig. 3b.

7 Set-Oriented Operations

Please be reminded that an HRD records a set of convex polyhedra. For convenience of discussion, given an HRD, we may just represent it as the set of convex polyhedra recorded in it. Definitions of set-union ($\cup$), set-intersection ($\cap$), and set-exclusion ($-$) of two convex polyhedra sets respectively represented by two HRDs are straightforward. For example, given HRDs $D_1 : \{\zeta_1, \zeta_2\}$ and $D_2 : \{\zeta_2, \zeta_3\}$, $D_1 \cap D_2$ is the HRD for $\{\zeta_2\}$, $D_1 \cup D_2$ is for $\{\zeta_1, \zeta_2, \zeta_3\}$, and $D_1 - D_2$ is for $\{\zeta_1\}$.

Given an evaluation ordering, we can write HRD-manipulation algorithms pretty much as usual [10], [12], [21], [25]. For convenience of presentation, we may represent an HRD $(u, (\beta_1, B_1), \ldots, (\beta_n, B_n))$ symbolically as $(u, (\beta_i, B_i)_{1 \leq i \leq n})$. A union operation $\texttt{union}(B, D)$ can then be implemented as follows:

set Ψ; /* database for the recording of already-processed cases */
$\texttt{union}(B, D)$ {
if $B = false$, return D; else if $D = false$, return B;
$\Psi := \emptyset$; return $\texttt{rec\_union}(B, D)$;
}
$\texttt{rec\_union}(B, D)$ where $B = (\epsilon, (\beta_i, B_i)_{1 \leq i \leq n})$, $D = (\epsilon', (\alpha_j, D_j)_{1 \leq j \leq m})$ {
if B is $true$ or D is $true$, return $true$;
else if $\exists F, (B, D, F) \in \Psi$, return F; ..... (a)
else if $\epsilon \prec \epsilon'$, construct

$F := (\epsilon, (\beta_i, \texttt{rec\_union}(B_i, D))_{1\le i\le n})$;
else if $\epsilon' \prec \epsilon$, construct
$F := (\epsilon', (\alpha_j, \texttt{rec\_union}(B, D_j))_{1\le j\le m})$;
else {
$i := n; j := m; F := false$;
while $i \ge 1$ and $j \ge 1$, do {
if $\beta_i = \alpha_j$, { $F := F \triangleright (\epsilon, (\beta_i, \texttt{rec\_union}(B_i, D_j)))$;
$i := i - 1; j := j - 1;$ }
else if $\beta_i \sqsubset \alpha_j$, { $F := F \triangleright (\epsilon, (\alpha_j, D_j)); j := j - 1;$}
else if $\alpha_j \sqsubset \beta_i$, { $F := F \triangleright (\epsilon, (\beta_i, B_i)); i := i - 1;$}
}
if $i \ge 1$, $F := F \triangleright (\epsilon, (\beta_h, B_h)_{1\le h\le i})$;
if $j \ge 1$, $F := F \triangleright (\epsilon, (\alpha_k, D_k)_{1\le k\le j})$;
}
$\Psi := \Psi \cup \{(B, D, F)\}$; return F; ..... (b)
}

Here, we conveniently use operator $\triangleright$ for the following construction:

- $false \triangleright D = D$
- $(\epsilon, (\beta_i, B_i)_{h<i\le n}) \triangleright (\epsilon, (\beta_i, B_i)_{1\le i\le h}) = (\epsilon, (\beta_i, B_i)_{1\le i\le n})$ for each $1 \le h < n$.

Note that, in Statement (a), we take advantage of the data-sharing capability of HRDs so that we do not process the same substructure twice. Set Ψ is maintained in Statement (b) to record the processing result the first time the substructure is encountered. The complexities of the three manipulations are all $O(|D_1| \cdot |D_2|)$. The algorithms for $\cap$ and $-$ are pretty much the same. For simplicity of presentation, we have left the algorithm for set subtraction to Appendix A.

Given two convex polyhedra ζ_1 and ζ_2, $\zeta_1 \wedge \zeta_2$ is a new convex polyhedron representing the space-intersection of ζ_1 and ζ_2. Formally speaking, for decision atom $\sum_i a_i x_i$, $\zeta_1 \wedge \zeta_2(\sum_i a_i x_i) = \zeta_1(\sum_i a_i x_i)$ if $\zeta_1(\sum_i a_i x_i) \sqsubset \zeta_2(\sum_i a_i x_i)$; or $\zeta_2(\sum_i a_i x_i)$ otherwise. Space-intersection ($\wedge$) of two HRDs D_1 and D_2, in symbols $D_1 \wedge D_2$, is a new HRD for $\{\zeta_1 \wedge \zeta_2 \mid \zeta_1 \in D_1; \zeta_2 \in D_2\}$. For simplicity of presentation, we have left the algorithm for space intersection to Appendix B. Our current algorithm of the manipulation has complexity $O(|D_1|^2 |D_2|^2)$.

8 HRD+BDD

It is possible to combine HRD and BDD into one data-structure for fully symbolic manipulation. Since HRD only has one sink node, *true*, it is more compatible with BDD without FALSE terminal node, which is more space-efficient than ordinary BDD. There are two things we need to take care of in this combination. The first is about the interpretation of default values of 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 HRD, when we find a decision atom $\sum_i a_i x_i$ is missing along a path, the decision atom is interpreted as $\sum_i a_i x_i < \infty$.

The second is about the interpretation of HRD 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 HRD+BDD a combination structure of HRD and BDD.

9 Weakest Preconditon Calculation and Symbolic PSA

Our tool **RED** 5.3 runs a backward reachability analysis by default. Suppose we are given an LHA A. There are two basic procedures in this analysis procedure. The first, $\texttt{xtion}(D, e)$, computes the weakest precondition from state-space represented by HRD D through discrete transition e from mode q to q'. Assume that the variables that get assigned in e are $y_1, \ldots, y_k$ and there is no variable that gets assigned twice in e. The characterization of $\texttt{xtion}(D, e)$ is

$$\mu(q) \wedge \tau(e) \wedge \exists y_1 \ldots \exists y_k \begin{pmatrix} & D \\ \wedge & \wedge_{1\le i\le k; \pi(e,y_i)\in\Theta}\ y_i \in \pi(e, y_i) \\ \wedge & \wedge_{1\le i\le k; \pi(e,y_i)\in\Lambda(X)}\ y_i = \pi(e, y_i) \end{pmatrix}.$$

For convenience of discussion, here we assume that, in the same transition, a variable does not occur in both the left-hand-side and right-hand-side of the assignments. We assume that there is a special discrete variable "mode" whose value is changed from q to q' in the transition. Also, we use the notations for membership constraints in intervals: $y \in [d, d'] \equiv d \le y \le d'$; $y \in (d, d'] \equiv d < y \le d'$; $y \in [d, d') \equiv d \le y < d'$; and $y \in (d, d') \equiv d < y < d'$.

Assume that $\texttt{delta\_exp}(D)$ is the same as D, except that all dense variables x are replaced by $x + \delta_x$, respectively. Here, δ_x represents the value-change of variable x in time-passage. For example, $\texttt{delta\_exp}(2x_1 - 3x_2 \le 3/5) = ''2x_1 + 2\delta_{x_1} - 3x_2 - 3\delta_{x_2} \le 3/5.''$ Intuitively, when x represents the value of variable x in the precondition of time passage, then $x + \delta_x$ is the value of x in the post-condition of the time-passage.

Please be reminded that for each dense variable x, $\gamma(q, x)$ specifies the rate interval of x in q. The second basic procedure, $\texttt{time}(D, q)$, computes the weakest precondition from D through time passage in mode q. It is characterized as

$$\mu(q) \wedge \exists \delta_{x_1} \exists \delta_{x_2} \ldots \exists \delta_{x_n} \exists \delta \begin{pmatrix} & \delta \ge 0 \wedge \texttt{delta\_exp}(D) \\ \wedge & \wedge_{1\le i\le n; \gamma(q,x_i)=\langle d_i, d_i'\rangle}\ \delta_{x_i} \in \langle d_i\delta, d_i'\delta\rangle \end{pmatrix}.$$

One basic building block of both $\texttt{xtion}()$ and $\texttt{time}()$ is for the evaluation of $\exists x(D(x))$ and is implemented as a Fourier-Motzkin elimination of x.

$$\exists x(D(x)) \equiv \texttt{var\_del}(\texttt{xtivity}(D, x), \{x\}).$$

Procedure $\texttt{var\_del}(D, X')$ eliminates all constraints in D involving variables in set X'. Procedure $\texttt{xtivity}(D, x)$ adds to a path every constraint that can be transitively deduced from two peer constraints involving x in the same path in D. The algorithm of $\texttt{xtivity}()$ is as follows:

set Ψ, Φ;
$\texttt{xtivity}(D, x)$ { $R := \emptyset$; return $\texttt{rec\_xtivity}(D, x)$; }
$\texttt{rec\_xtivity}(D, x)$ {

if D is $true$ or $false$, return D;
else if $\exists(D, D') \in \Psi$, return D';
else /* assume $D = (ax + \epsilon, (\beta_1, D_1), \ldots, (\beta_m, D_m))$ */ {
$D' :=$ `rec_xtivity`(D_i, x)
$\Phi := \emptyset; D' := \bigcup_{1 \le i \le m} ax + \epsilon\beta_i \wedge$
`rec_xtivity_given`$(D', ax + \epsilon, \beta_i)$;
$\Psi := \Psi \cup \{(D, D')\}$; return D';
}
}
`rec_xtivity_given`$(D, ax + \epsilon, \beta)$ {
if D is $true$ or $false$, return D;
else if $\exists(D, D') \in \Phi$, return D'; ..... (c)
else /* assume $D = (bx + \epsilon', (\beta_1, D_1), \ldots, (\beta_m, D_m))$ */ {
if $ab < 0$,

$$D' := \bigcup_{1 \le i \le m} \begin{pmatrix} bx+\epsilon'\beta_i \wedge \text{rec\_xtivity\_given}(D_i, ax+\epsilon, \beta) \\ \wedge \quad |b|\epsilon'/\gcd(a,b) + |a|\epsilon'/\gcd(a,b)((|b|\beta + |a|\beta_i)/\gcd(a,b)) \end{pmatrix};$$

else $D' := \bigcup_{1 \le i \le m} bx + \epsilon'\beta_i \wedge$
`rec_xtivity_given`$(D_i, ax + \epsilon, \beta)$;
$\Phi := \Phi \cup \{(D, D')\}$; return D'; ..... (d)
}
} /* $(|b|\beta + |a|\beta_i)/\gcd(a, b)$ is a shorthand for the new upperbound obtained from the transitivity of $ax + \epsilon\beta$ and $bx + \epsilon'\beta_i$. */

Thus, we preserve all constraints transitively deducible from a dense variable before it is eliminated from a predicate. This guarantees that no information will be unintentionally lost after the variable elimination.

Again, in our algorithm, we do not enumerate all paths in HRD to carry out this least fixpoint evaluation. Instead, in Statement (c), our algorithm follows the traditional BDD programming style which takes advantage of the data-sharing capability of BDD-like data-structures. Thus, our algorithm does not explode due to the combinatorial complexity of path counts in HRD. This can be justified by the performance of our implementation reported in Section 13.

10 Normalization

There can be more than one set of LH-constraints representing the same convex polyhedron. Moreover, convex polyhedra may contain one another. Thus, it is important for efficiency not to waste computing power and memory space in recording the same state-spaces again and again. We rely on two techniques to counter the challenge. First, we normalize the representation of the convex polyhedron for each path in HRD. Second, we use a simple technique to eliminate some convex polyhedra contained by others. Discussion on the two techniques follows:

10.1 Normalization of Polyhedron Representation

A *normal form* is a representation that satisfies some requirement. Normal forms are usually defined according to the geometric structure of convex polyhedra. For example, an LH-constraint set is *minimal* if the removal of any LH-constraint from the set will change the shape of the convex polyhedron it represents. We can use minimal LH-constraint sets as normal forms for the corresponding convex polyhedron. Depending on the requirement, a convex polyhedron may have one or more normal forms. For example, the following two sets are both minimal and characterize the same space.

$$\{A + 3x \le 5, -2x + y \le -8, -2A - 3y \le 3\}$$
$$\{A + 3x \le 5, -A - 3x \le -5, 2A + 3y \le -3, -2A - 3y \le 3\}$$

If we use strict requirements so that each convex polyhedron has exactly one normal form, then we call this normal form the *canonical form*. For timed automata, there is a natural canonical form called *all-pair-shortest-path form* (or tight form) [13] in which each constraint is tight. In comparison, there is no such natural canonical form for convex polyhedra. The reason is that, by repetitively adding an equality to the two sides of an LH-constraint, we still get a tight LH-constraint. For example, each LH-constraint in set $\{A + 3x \le 5, -2x + y \le -8, 2x - y \le 8\}$ is tight. However, the set implies $-2x + y = -8$. Thus, $A + 3x - 2kx + ky \le 5 - 8k$, for any integer k, is still in a tight form representation.

As can be seen from the above, there are more issues to take care of in implementing normal forms of convex polyhedra than zones. In our experience, normalizing convex polyhedron representations in HRDs may consume most of the resources in computation. The implementation of canonical forms will be very costly. Thus, we have decided to sacrifice the precision of normalization for computation efficiency. Given an LH-constraint set, we shall only try to eliminate some LH-constraints which are redundant. An LH-constraint is *redundant* in a set if it is transitively deducible from other constraints in the set. It is *2-redundant* if it is deducible from exactly two other LH-constraints in the set. Formally speaking, given three LH-constraints $\kappa_a : \sum_i a_i x_i \alpha$, $\kappa_b : \sum_i b_i x_i \beta$, and $\kappa_c : \sum_i c_i x_i \gamma$, κ_c is 2-redundant with respect to κ_a and κ_b iff there exist positive integers h, j, k such that $h \sum_i a_i x_i + j \sum_i b_i x_i = k \sum_i c_i x_i$ and $h\alpha + j\beta \sqsubseteq k\gamma$. (Please be reminded that the addition and constant multiplication of LH-upperbounds were defined in Section 4.) The solutions for h, j, k can be obtained with standard Gaussian elimination on inequality systems. In the following, we assume that we have a procedure `solvable`$(\epsilon_a, \epsilon_b, \epsilon_c, \&h, \&j, \&k)$, which returns $true$ when $h\epsilon_a + j\epsilon_b = k\epsilon_c$ has an integer solution with $h > 0, j > 0, k > 0$ and returns $false$ otherwise. When it returns $true$, the actual parameters h, j, k are returned with one such solution.

Given an evaluation ordering $\prec$, an LH-constraint $\epsilon\beta$ is *downward redundant* in a set if it is transitively deducible from peer LH-constraints $\epsilon_1\alpha_1, \ldots, \epsilon_n\alpha_n$ in the same set and for all $1 \le i \le n$, $\epsilon_i \prec \epsilon$. Downward redundancy elimination is easy to implement with HRD since it respects the ordering of evaluation.

At this moment, our normalization only supports the elimination of *downward 2-redundant LH-constraints*. The algorithm is called `D2R_del()` and uses two variables to hold the LH-expressions ϵ_a, ϵ_b and two variables to hold the LH-upperbounds α, β.

LH-expression ϵ_a, ϵ_b;
LH-upperbound α, β;
set Ψ, Φ, Γ;
`D2R_del`(D)\{ $\Psi := \emptyset$; return `rec_D2R_del`(D); \}

$\texttt{rec\_D2R\_del}(D)$ with $D = (\epsilon, (\alpha_i, D_i)_{1\le i\le m})$ {
 if D is $true$ or $false$, return D;
 else if $\exists F, (D, F) \in \Psi$, return F;
 $F := false$;
 for $i := 1$ to m, {
  $D' := \texttt{rec\_D2R\_del}(D_i)$; $\epsilon_a := \epsilon$; $\alpha := \alpha_i$; $\Phi := \emptyset$;
  $F := F \cup (\epsilon, (\alpha_i, \texttt{rec\_D2R\_del\_given\_a}(D')))$;
 }
 $\Psi := \Psi \cup \{(D, F)\}$; return F;
}
$\texttt{rec\_D2R\_del\_given\_a}(D)$ with $D = (\epsilon, (\alpha_i, D_i)_{1\le i\le m})$ {
 if D is $true$, return D; else if $\exists F, (D, F) \in \Phi$, return F;
 $F := false$;
 for $i := 1$ to m, {
  $D' := \texttt{rec\_D2R\_del\_given\_a}(D_i)$; $\epsilon_b := \epsilon$; $\beta := \alpha_i$; $\Gamma := \emptyset$;
  $F := F \cup (\epsilon, (\alpha_i, \texttt{rec\_D2R\_del\_given\_ab}(D')))$;
 }
 $\Phi := \Phi \cup \{(D, F)\}$; return F;
}
$\texttt{rec\_D2R\_del\_given\_ab}(D)$ with
  $D = (\epsilon, (\alpha_i, D_i)_{1\le i\le m})$ {
 if D is $true$, return D; else if $\exists F, (D, F) \in \Gamma$, return F;
 $F := false$;
 if $\texttt{solvable}(\epsilon_a, \epsilon_b, \epsilon, \&h, \&j, \&k)$, {
  $F := (\epsilon, (\alpha_i, \texttt{rec\_D2R\_del\_given\_ab}(D_i))_{1\le i\le m; k\alpha_i \sqsubseteq h\alpha + j\beta})$;
  $F := F \cup \bigcup_{1\le i\le m; h\alpha+j\beta \sqsubset k\alpha_i} \texttt{rec\_D2R\_del\_given\_ab}(D_i)$;
 }
 else $F := (\epsilon, (\alpha_i, \texttt{rec\_D2R\_del\_given\_ab}(D_i))_{1\le i\le m})$;
 $\Gamma := \Gamma \cup \{(D, F)\}$; return F;
}

10.2 Polyhedron Elimination by Straightforward Containment Checking

Polyhedron containment checking in an HRD is also important in controlling the size of the HRD. Specifically, we want to check if a path in the HRD specifies a polyhedron that is subsumed by another path's polyhedron. In [25], a sufficient condition called *SCR (Straightforward Containment Requirement)* was presented for the containment checking of zones. We have adapted SCR for HRDs. Intuitively, given $D = (v, (\alpha_j, D_j)_{1\le j\le m})$, D and $(v, (\alpha_j, D_j - \bigcup_{j<k\le m} D_k)_{1\le j\le m})$ should characterize the same state-space. In the following, we first prove its correctness and then present our algorithm:

Lemma 1. *Given any* $D = \bigcup_{1\le j\le m} \epsilon\alpha_j \wedge D_j$ *such that* $\alpha_1 \sqsubset \alpha_2 \sqsubset \ldots \sqsubset \alpha_m$, D *and* $\bigcup_{1\le j\le m} \epsilon\alpha_j \wedge (D_j - \bigcup_{j<k\le m} D_k)$ *characterize the same state-space.*

Proof. We prove the lemma by contradiction. We assume that D and $\bigcup_{1\le j\le m} \epsilon\alpha_j \wedge (D_j - \bigcup_{j<k\le m} D_k)$ do not characterize the same real space. Let us examine the largest index i so that $\bigcup_{i\le j\le m} \epsilon\alpha_j \wedge D_j$ and $\bigcup_{i\le j\le m} \epsilon\alpha_j \wedge (D_j - \bigcup_{j<k\le m} D_k)$ do not characterize the same state-space. This means that D_i and $\bigcup_{i<j\le m} D_j$ have a nonempty intersection. Assume one convex polyhedron in the intersection is ζ. Assume $k = \max\{j \mid \zeta \in D_j; i < j \le m\}$. Then, the ζ that is subtracted from D_i actually means $\zeta \wedge \epsilon\alpha_i$, which is subtracted. On the other hand, the $\zeta \in D_k$ means $\zeta \wedge \alpha_k$, which remains in $\bigcup_{i\le j\le m} \epsilon\alpha_j \wedge (D_j - \bigcup_{j<k\le m} D_k)$. Since $\alpha_i \sqsubset \alpha_k$, we know that the removal of ζ from D_i does not make $\bigcup_{i\le j\le m} \epsilon\alpha_j \wedge D_j$ different from $\bigcup_{i\le j\le m} \epsilon\alpha_j \wedge (D_j - \bigcup_{j<k\le m} D_k)$. Since this argument is applicable to all is and all such ζs, we infer that D cannot be different from $\bigcup_{1\le j\le m} \epsilon\alpha_j \wedge (D_j - \bigcup_{j<k\le m} D_k)$. This is a contradiction. □

We have implemented a procedure $\texttt{SCR}(D)$ (Straightforward Containment Reduction) to eliminate all convex polyhedra contained by other convex polyhedra in D.

set Ψ;
$\texttt{SCR}(D)$ { $\Psi := \emptyset$; return $\texttt{rec\_SCR}(D)$; }
$\texttt{rec\_SCR}(D)$ with $D = (\epsilon, (\alpha_i, D_i)_{1\le i\le m})$ {
 if D is $true$ or $false$, return D; .....(e)
 else if $\exists F, (D, F) \in \Psi$, return F;
 else {
  $i := m$; $F := false$; $\dot{D} := false$;
  for $i := m$ to 1, do .....(f)
   {$F := F \cup (\epsilon, (\alpha_i, \texttt{rec\_SCR}(D_i) - \dot{D}))$; $\dot{D} := \dot{D} \cup D_i$;}..(g)
 }
 $\Psi := \Psi \cup \{(D, F)\}$; return F;
}

11 Pruning Strategy-Based on Parameter Space Construction (PSPSC)

We have also experimented with techniques to improve the efficiency of parametric safety analysis. One such technique, called *PSPSC*, is avoiding new state-space exploration if the exploration does not contribute to new parametric solutions. A constraint is *static* iff all its variables are static parameters. Static constraints do not change their truth values. Once a static constraint is derived in a convex polyhedron, its truth value will be honored in all weakest preconditions derived from this convex polyhedron. All states backwardly reachable from a convex polyhedron must also satisfy the static constraints required in the polyhedron. Thus, if we know that static parameter valuation $\mathcal{H}$ is already in the parametric solution space, then we really do not need to explore those states whose parameter valuations fall in $\mathcal{H}$.

We use the LHA in Fig. 4, with two static parameters α and β, to explain why PSPSC works. We want to deduce the constraint on α and β that makes the risk mode reachable from the initial mode q_0. In the LHA, modes q_0, q_1, q_2 constitute a loop whose execution engenders the static parametric constraint of $\alpha < \beta$. Dense variable x is reset once in each execution of the loop. y is used to accumulate the time when the LHA stays in q_1 at each execution. The state-space of the LHA is divergent. In Table 2, we report the weakest precondition calculations of the first few iterations. After we have constructed the constraints for the first execution through the loop, we have already deduced that the constraint of $\beta > 0 \wedge \alpha < \beta$ is sufficient for the reachability of the risk mode from q_0. Then, at the beginning of the second loop execution, when calculating the weakest precondition from q_0 to q_2, this static constraint of $\beta > 0 \wedge \alpha < \beta$ will be carried over to all preconditions henceforth deduced. Thus, after the first loop execution, any explora-

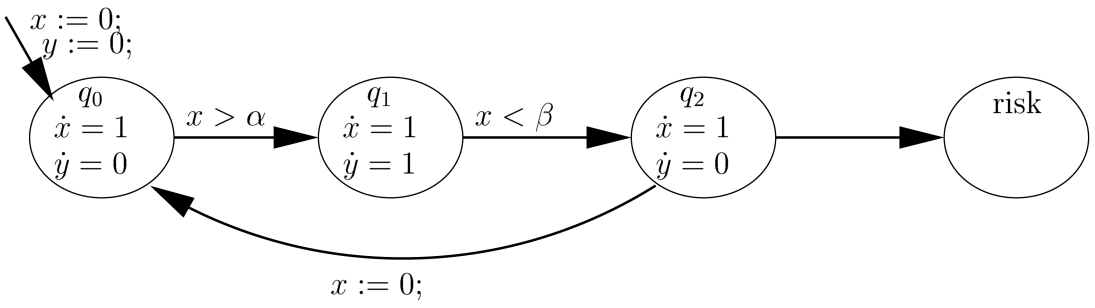

Fig. 4. A simple LHA to explain PSPSC.

tion of the weakest precondition along this backward path will not add any new parametric solutions for the reachability of the risk condition. For parametric safety analysis, PSPSC says that we do not have to make further exploration from mode q_0 after the first loop execution.

With PSPSC, our new parametric safety analysis procedure is as follows:

```
PSA_with_PSPSC(A, η) {
  D̄ := time(¬η, q_f); D := false; P := var_del(D, X − H);
  while D̄ ≠ false, do {
    D := D ∪ D̄; D̄ := ⋃_{e=(q,q')∈E} time(xtion(D̄, e), q);
    D̄ := D̄ ∧ (¬P) ∧ (¬D); ..............................(h)
    P := P ∪ var_del(I ∧ D̄, X − H);
  }
  return ¬P;
}
```

In the procedure, we use variable P to symbolically accumulate the parametric evaluations leading to the unsafe states in the least fixpoint iterations. In Statement (h), we check and eliminate in $\bar{D}$ those state descriptions which cannot possibly contribute to new parametric evaluations by conjuncting $\bar{D}$ with $\neg P$.

One nice feature of PSPSC is that it does not sacrifice the precision of our parametric safety analysis.

Lemma 2. *$\mathcal{H}$ is a* parametric solution *to A and η iff $\mathcal{H}$ satisfies the return result of* PSA_with_PSPSC(A, η).

Proof. Note that the intersection at Line (h) in procedure PSA_with_PSPSC() only stops the further exploration of those states that do not contribute to new parameter-spaces. Those parameter-spaces pruned in Line (h) do not contribute because they are already contained in the known parameter constraints P and, along each exploration path, the parameter constraints only get stricter. □

PSPSC can help in pruning the space of exploration in big chunks. But, in the worst case, PSPSC does not guarantee the exploration will terminate. Note that PSPSC works if we can quickly construct a symbolic path from the initial polyhedra to a risk polyhedra. Once such a path is constructed, we can use the static parameter constraints of that path to prune the state space yet to be explored. One case that could prevent PSPSC from termination is as follows: Suppose we have a state subspace that does not need any static parameter constraints for their reachability. When this subspace is not representable by a finite set of convex polyhedra, the state-space exploration algorithm will not terminate its exploration of this subspace. Moreover, since the exploration of this subspace does not engender any static parameter constraints, its exploration cannot be pruned by PSPSC.

In Section 13, we shall report the performance of this technique.

12 Related Work

Many modern model-checkers [20], [25], [31] for timed automata [6] are built around symbolic manipulation procedures [16], [7] of *zones*. A zone is characterized by a conjunction of constraints like $x_1 - x_2 \sim c$ or $x_1 \sim c$, where x_1, x_2 are two clocks, "$\sim$" is one of $\leq, <, =, >, \geq$, and c is an integer constant. Geometrically, it means a convex state

TABLE 2
Weakest Precondition Calculations to Explain PSPSC

loop executions	0		1						2
WPC iterations	0		1		2		3		4
WP	risk	$\rightarrow$	q_2	$\rightarrow$	q_1 $\wedge$ $\beta > 0$ $\wedge$ $x > 0$ $\wedge$ $\beta > x$	$\rightarrow$	q_0 $\wedge$ $\beta > 0$ $\wedge$ $\alpha < \beta$ $\wedge$ $x > 0$ $\wedge$ $\beta > x$	$\rightarrow$	q_2 $\wedge$ $\beta > 0$ $\wedge$ $\alpha < \beta$

WP: Weakest precondition. WPC: Weakest precondition calculation.

space of timed automata. The most popular data-structure for zones is DBM [13], which is a two dimensional matrix recording differences between pairs of clocks and nothing BDD-like.

People have used convex subspaces, called *convex polyhedra*, as a basic unit for symbolic manipulation. A convex polyhedron characterizes a state-space of an LHA and can be symbolically represented by a set (or conjunction) of constraints like $a_1x_1 + \ldots + a_nx_n \sim c$ [4], [7], [5], where $a_1, \ldots, a_n$ are integer constants, $x_1, \ldots, x_n$ are system variables and c is a rational number. Two commonly used representations for convex polyhedra in HyTech are polyhedra and frames in dense state-space [15]. These two representations are neither BDD-like nor can represent concave state-spaces. Data-sharing among convex polyhedra is difficult.

In 1993 [29], Wang et al. discussed how to use BDD with decision atoms like $x_i + c \leq x_j + d$ to model-check timed automata. In the last several years, people have explored this approach in the hope of duplicating the success of BDD techniques [10], [12] in hardware verification for the verification of timed automata [2], [9], [11], [18], [21], [22], [23], [24], [25]. Our HRD can be seen as a variation of *CDD (Clock-Difference Diagram)* [11] and an extension of *CRD (Clock-Restriction Diagram)* [23], [24] for timed systems. The decision atoms in a CRD are all clock differences like $x - x'$. Thus, CRDs are incapable of representing the state-space of LHAs. In [11], CDD only served as a recording device in that reachable state-space representations were analyzed using DBM [13] and then converted for recording to CDD. In [25], [27], [26], a full set of verification algorithms (including forward/backward reachability analysis, normalization, and full TCTL model-checking procedures) for CRD were reported.

For parametric safety analysis, Annichini et al. have extended DBM [13] to PDBM for parametric safety analysis of timed automata [1], [3] and implemented a tool called *TReX*, which also supports verification with lossy channels.

13 Implementation and Experiments

We have implemented our ideas in our tool **RED**, version 5.3, which has been previously reported in [21], [22], [23], [24], [25] for the verification of timed automata. **RED** 5.3 supports full TCTL model-checking/simulation with graphical user-interface. Techniques for coverage estimation of dense-time state-spaces and fast greatest fixpoint evaluation have been reported [27], [28]. In addition, we have also compared with HyTech 2.4.5 [15] and TReX 1.3 [1], [3] against several benchmarks. The first three benchmark series are adapted from HyTech benchmark repository. The last, CSMA/CD, is adapted from [31].

- *Fischer's mutual exclusion algorithm.* This is one of the classic benchmarks. There are two static parameters A and B, m processes, and one local clock for each process. The first process has a local clock with rate in $[4/5, 1]$ while all other processes have local clocks with rates in $[1, 11/10]$. The algorithm may violate the mutual exclusion property when $-A \leq 0 \wedge -11A + 8B \leq 0$.
- *General railroad crossing benchmarks.* There is a static parameter `CUTOFF`, a gate-process, a controller-process, and m train-processes. The local dense variable of the gate-process models the angle of the gate and has rates in $[0, 0]$, $[-10, -9]$, and $[9, 10]$ depending on which modes the gate-process is in. The controller process does not use clocks. Each train-process uses a local clock with rate in $[1, 1]$. The system may not lower the gate in time for a crossing train when $20 \leq$ `CUTOFF` ≤ 40.
- *Nuclear reactor controller.* There are m rod-processes and one controller process. Each process has a clock with rate in $[1, 1]$. A rod just-moved out of the heavy water must stay out of water for at least T (a static parameter) time units. The timing constants used in the benchmarks are $58/10, 59/10, 16$, and $161/10$. The controller may miss the timing-constraints for the rods if $-T \leq -(109m - 29)/5$.
- *CSMA/CD.* The two timing constants A and B, set to 26 and 52, respectively, are now treated as static parameters to be analyzed. We do require that $B \geq 52$. Basically, this is the Ethernet bus arbitration protocol with the idea of collision-and-retry. The biggest timing constant used is 808. We want to verify that mutual exclusion after bus-contending period can be violated if

$$A > 0 \wedge B \geq 52 \wedge B \leq 808 \wedge B < 2A.$$

Experiments are conducted on a Pentium 4M 1.6GHz/256MB running LINUX.

13.1 Comparison with HyTech 2.4.5

We have also carried out experiments to compare various ideas mentioned in this work. In addition, we have also compared with HyTech 2.4.5 [15], which is the best known and most popular tool for the verification of LHA due to its pioneering importance. We compare performance in both forward and backward reachability analyses. The performance data of HyTech 2.4.5 and **RED** 5.3 with dictionary ordering (no PSPSC), coefficient ordering (no PSPSC), magnitude ordering (PSPSC), and coefficient ordering with PSPSC is reported in Table 3.

The experiment, although not extensive, does show signs that HRD-technology (with or without PSPSC) can compete with the technology used in HyTech 2.4.5. For all the benchmarks, HRD-technology demonstrates better scalability with regard to concurrency complexity.

For the forward analysis with dictionary ordering and without PSPSC, the GRC benchmark with three trains incurs an LH-upperbound overflow while eliminating redundancies in an HRD. This happens because we have implemented rational upperbounds with 32-bit integer numerators and denominators. When we use two or more LH-constraints to transitively deduce a new LH-constraint, the numerator and denominator of this new LH-constraint is of the same order as the lcm of the numerators and denominators of those former LH-constraints and may not be representable by 32-bit integers. Our tool terminates when it detects the possibility of such representation overflow.

TABLE 3
Comparison with HyTech with Regard to Number of Processes

direction	benchmarks	rates used	m	HyTech 2.4.5	**RED** 5.3 (backward)			
					dictionary	coefficient	magnitude	coefficient
					no PSPSC			PSPSC
backward	Fischer's mutual exclusion (m processes)	$[4/5, 1]$ $[1, 11/10]$	2	0.23s	0.10s/17k	0.11s/17k	0.11s/17k	0.07s/16k
			3	2.40s	1.83s/81k	1.75s/74k	1.23s/59k	0.70s/44k
			4	28.04s	20.29s/320k	23.85s/269k	12.38s/215k	5.14s/163k
			5	O/M	278.8s/1420k	354.1s/1149k	162.0s/1034k	31.36s/474k
			6	O/M	2846s/5848k	9923s/8796k	1485s/4000k	168.6s/1170k
	GRC (gate +controller +m trains)	$[0, 0], [1, 1]$ $[9, 10]$ $[-10, -9]$	2	O/M	0.79s/103k	0.68s/101k	0.68s/101k	0.76s/94k
			3	O/M	11.48s/806k	8.85s/616k	8.84s/616k	11.48s/530k
			4	O/M	248.5s/6046k	184.9s/4249k	186.1s/4249k	252.5s/2820k
			5	O/M	6095s/37093k	4883s/25841k	4900s/25841k	6527s/19234k
	reactor (controller +m rods)	$[1, 1]$	2	0.056s	0.08s/19k	0.07s/19k	0.06s/19k	0.05s/15k
			3	0.33s	0.41s/51k	0.38s/52k	0.37s/52k	0.22s/41k
			4	2.61s	3.10s/187k	2.69s/186k	2.71s/186k	1.42s/155k
			5	31.29s	41.47s/1042k	37.03s/1039k	36.89s/1039k	18.67s/884k
			6	647.8s	951.5s/8228k	866.9s/8191k	839.3s/8191k	461.8s/6941k
	CSMA/CD (bus +m senders)	$[1, 1]$	2	O/M	0.98s/42k*	1.47s/125k	0.57s/34k	0.56s/33k
			3	O/M	O/M	5076s/2407k*	121.5s/807k	0.66s/105k
			4	O/M	O/M	O/M	O/M	2.47s/378k
			5	O/M	O/M	O/M	O/M	9.77s/1192k
			6	O/M	O/M	O/M	O/M	40.58s/3513k
forward	Fischer's mutual	same as above	2	0.34s	0.10s/20k	0.10s/20k	0.10s/19k	0.08s/18k
			3	37.89s	O/M	22.10s/561k	19.18s/654k	5.59s/538k
	GRC	same as above	2	3.29s	2.29s/192k	1.41s/95k	1.43s/95k	0.44s/84k
			3	O/M	O/F	274.5s/2165k	272.5s/2165k	6.35s/418k
	reactor	$[1, 1]$	2	O/M	O/M	O/M	O/M	O/M
	CSMA/CD	$[1, 1]$	2	0.19s	0.19s/29k	0.17s/29k	0.17s/29k	0.25s/33k
			3	2.63s	1.81s/102k	1.64s/101k	1.62s/101k	2.61s/106k
			4	68.75s	20.07s/370k	17.49s/378k	17.52s/378k	27.03s/378k
			5	O/M	268.0s/1905k	240.3s/1906k	242.2s/1906k	331.9s/1910k
			6	O/M	3889s/11725k	3123s/11525k	3155s/11525k	4163s/11552k

s: seconds; k: kilobytes of memory in data-structure;
O/M: Out of memory; O/F: LH-upperbound overflow.

Finally, PSPSC cuts down the time and memory needed for parametric safety analysis. This shows very good promise of this technique.

13.2 Comparison with TReX 1.3

Another famous tool for the verification of hybrid systems is TReX [1], [3], which supports the verification of systems with clocks, parameters, and lossy channels. As mentioned in Section 12, the time-progress weakest precondition (or strongest postcondition in forward analysis) calculation algorithm in **RED** 5.3 is more complex than the one in TReX. And TReX now mainly runs in forward analysis. And, TReX also may have tuned its performance for systems with lossy channels. Thus, it can be difficult to compare the performance of TReX with **RED** 5.3 directly. Anyway, we still tried hard and used one week to learn the input language of TReX and to analyze two benchmarks. Since TReX 1.3 only supports the verification of systems with clocks, parameters, and lossy channels, we choose the following two benchmarks. The first is Fischer's protocol with all clocks in the uniform rate of 1. The second is the Nuclear Reactor Controller. The performance data is shown in Table 4 for both forward and backward analysis.

Two additional options of **RED** 5.3 were chosen: coefficient evaluation ordering with PSPSC and magnitude evaluation ordering without. We marked N/A (not available) with higher concurrencies when we feel that too much time (like more than 1 hour) or too much memory (20MB) has been consumed in early fixpoint iterations. Although the data set is still small and incomplete, but we feel that the HRD-technology shows promise in the table.

14 Summary

This work is a first step toward using BDD-technology for the verification of LHAs. Although the initial experiment data shows good promise, we feel that there are still many issues worthy of further research to check the pros and cons of HRD-technology. Especially, we intend to research techniques for fast redundant constraints elimination. Although our current normalization approach in Section 10 does perform well against the benchmarks, we still hope that there is a better way to check general redundancy.

Also, subsumption is another challenge. Straightforward implementation may use the complement of the current reachable state-space to filter those newly constructed weakest preconditions. Since the HRD of the current reachable state-space can be huge, its complement is very expensive to construct and maintain.

TABLE 4
Performance Comparison with TReX with Regard to Number of Processes

benchmarks	concurrency	Forward			Backward		
		TReX 1.3	**RED** 5.3 magnitude	**RED** 5.3 coeff.+PSPSC	TReX 1.3	**RED** 5.3 magnitude	**RED** 5.3 coeff.+PSPSC
Fischer's mutual exclusion (m	2 procs	1.12s	0.07s/17k	0.07s/15k	8.96s	0.08s/13k	0.04s/13k
	3 procs	O/M	1.86s/137k	0.78s/79k	N/A	0.66s/43k	0.49s/43k
	4 procs	O/M	197.9s/2714k	16.92s/539k	N/A	5.81s/180k	3.58s/158k
	5 procs	O/M	N/A	752.7s/5254k	N/A	59.27s/945k	24.71s/658k
	6 procs	O/M	N/A	N/A	N/A	567.1s/4341k	170.3s/2798k
reactor (m rods)	2 rods	O/M	O/M	O/M	N/A	0.06s/19k	0.05s/15k
	3 rods	O/M	O/M	O/M	N/A	0.37s/52k	0.22s/41k
	4 rods	O/M	O/M	O/M	N/A	2.71s/186k	1.42s/155k
	5 rods	O/M	O/M	O/M	N/A	36.89s/1039k	18.67s/884k
	6 rods	O/M	O/M	O/M	N/A	839.3s/8191k	461.8s/6941k

s: seconds; k: kilobytes of memory in data-structure;
O/M: Out of memory; N/A: not available.

APPENDIX A
PROCEDURE FOR HRD SUBTRACTION

Another basic operation used in Section 10.2 is set subtraction "$-$" which is implemented as follows: The procedure, in turn, is built upon another diadic procedure $\texttt{subtract}(B, D)$ which, in turn, eliminates all zones in B that are contained by some zones in D.

set Φ;
$\texttt{subtract}(B, D)$ {
  if $B = false$, return $false$;
  $\Phi := \emptyset$; return $\texttt{rec\_subtract}(B, D)$; }
$\texttt{rec\_subtract}(B, D)$ where $B = (\epsilon_B, (\beta_i, B_i)_{1\le i\le n})$,
    $D = (\epsilon_D, (\alpha_j, D_j)_{1\le j\le m})$ {
  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 $\epsilon_B \prec \epsilon_D$, construct
    $H := (\epsilon_B, (\beta_i, \texttt{rec\_subtract}(B_i, D))_{1\le i\le n})$;
  else if $\epsilon_B \succ \epsilon_D$, return B;
  else {
    $j := m; H := false; \dot{D} := false$;
    for $i := n$ to 1, do {
      while $j \ge 1 \wedge \beta_i \sqsubseteq \alpha_j$, do { $\dot{D} := \dot{D} \cup D_j$; $j := j - 1$;}
      $H := H \cup (\epsilon_B, (\beta_i, \texttt{rec\_subtract}(B_i, \dot{D})))$;
  } }
  $\Phi := \Phi \cup \{(B, D, H)\}$; return H;
}

APPENDIX B
PROCEDURE FOR HRD SPACE INTERSECTION

Space-intersection of two HRDs B and D, in symbols $B \wedge D$, is a new HRD for $\{\zeta_1 \wedge \zeta_2 \mid \zeta_1 \in B; \zeta_2 \in D\}$.

set Ψ;
$\wedge(B, D)$ {
  if $B = false$ or $D = false$, return $false$;
  $\Psi := \emptyset$; return $\texttt{rec} \wedge (B, D)$;
}
$\texttt{rec} \wedge (B, D)$ where $B = (\epsilon_B, (\beta_i, B_i)_{1\le i\le n})$,
    $D = (\epsilon_D, (\alpha_j, D_j)_{1\le j\le m})$ {
  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 $\epsilon_B \prec \epsilon_D$, construct CRD
    $H := (\epsilon_B, (\beta_i, \texttt{rec} \wedge (B_i, D))_{1\le i\le n})$;
  else if $\epsilon_B \succ \epsilon_D$, construct CRD
    $H := (\epsilon_D, (\alpha_j, \texttt{rec} \wedge (B, D_j))_{1\le j\le m})$;
  else {
    $i := n; j := m; H := false; \dot{B} := false; \dot{D} := false$;
    while $i \ge 1$ and $j \ge 1$, do {
      if $\beta_i = \alpha_j$, {
        $\dot{B} := \dot{B} \cup B_i; \dot{D} := \dot{D} \cup D_j;$ ..... (i)
        $H := H \cup (\epsilon_B, (\beta_i, \texttt{rec} \wedge (B_i, \dot{D}) \cup \texttt{rec} \wedge (\dot{B}, D_j)))$;
        $i := i - 1; j := j - 1$;
      }
      else if $\beta_i \sqsubset \alpha_j$, {
        while $j \ge 1 \wedge \beta_i \sqsubset \alpha_j$, do { $\dot{D} := \dot{D} \cup D_j$; $j := j - 1$;}..(j)
        $H := H \cup (\epsilon_B, (\beta_i, \texttt{rec} \wedge (B_i, \dot{D})))$;
      }
      else if $\alpha_j \sqsubset \beta_i$, {
        while $i \ge 1 \wedge \alpha_j \sqsubset \beta_i$, do { $\dot{B} := \dot{B} \cup B_i$; $i := i - 1$;}..(k)
        $H := H \cup (\epsilon_B, (\alpha_j, \texttt{rec} \wedge (\dot{B}, D_j)))$;
    } }
    if $i \ge 1$, $H := H \cup (\epsilon_B, (\beta_h, \texttt{rec} \wedge (B_h, \dot{D}))_{1\le h\le i})$;
    if $j \ge 1$, $H := H \cup (\epsilon_B, (\alpha_k, \texttt{rec} \wedge (\dot{B}, D_k))_{1\le k\le j})$;
  }
  $\Psi := \Psi \cup \{(B, D, H)\}$; return H;
}

Especially, in Lines (i), (j), and (k), 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 $O(|B|^2 \cdot |D|^2)$.

ACKNOWLEDGMENTS

The author would like to thank the TReX team, especially Mihaela Sighireanu and Aurore Collomb-Annichini, for kindly implementing a TReX version without the reduced package for him. A preliminary version of the paper appeared in Proceedings of CAV 2004, LNCS 3114, Springer-Verlag. This work was partially supported by NSC,

Taiwan, ROC under grants NSC 92-2213-E-002-103 and NSC 92-2213-E-002-104. Model-checker/simulator **RED** 5.3 is available at http://cc.ee.ntu.edu.tw/~val.

REFERENCES

[1] A. Annichini, E. Asarin, and A. Bouajjani, "Symbolic Techniques for Parametric Reasoning about Counter and Clock Systems," *Proc. Int'l Conf. Computer Aided Verification,* 2000.
[2] E. Asarin, M. Bozga, A. Kerbrat, O. Maler, A. Pnueli, and A. Rasse, "Data-Structures for the Verification of Timed Automata," *Proc. Hybrid and Real-Time Systems Int'l Workshop,* pp. 346-360, 1997.
[3] A. Annichini, A. Bouajjani, and M. Sighireanu, "TReX: A Tool for Reachability Analysis of Complex Systems," *Proc. 13th Int'l Conf. Computer Aided Verification,* pp. 368-372, 2001.
[4] R. Alur, C. Courcoubetis, T.A. Henzinger, and P.-H. Ho, "Hybrid Automata: An Algorithmic Approach to the Specification and Verification of Hybrid Systems," *Proc. Hybrid Systems Workshop,* pp. 209-229, 1993.
[5] R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine, "The Algorithmic Analysis of Hybrid Systems," *Theoretical Computer Science,* vol. 138, no. 1, pp. 3-34, 1995.
[6] R. Alur and D.L. Dill, "Automata for Modelling Real-Time Systems," *Proc. Int'l Colloquium on Automata, Languages and Programming,* pp. 322-335, 1990.
[7] R. Alur, T.A. Henzinger, and P.-H. Ho, "Automatic Symbolic Verification of Embedded Systems," *Proc. IEEE Real-Time System Symp.,* pp. 2-11, 1993.
[8] R. Alur, T.A. Henzinger, and M.Y. Vardi, "Parametric Real-Time Reasoning," *Proc. 25th ACM Symp. Theory of Computing,* pp. 592-601, 1993.
[9] F. Balarin, "Approximate Reachability Analysis of Timed Automata," *Proc. 17th IEEE Real-Time Systems Symp.,* p. 52, 1996.
[10] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang, "Symbolic Model Checking: 10^{20} States and Beyond," *Proc. IEEE Symp. Logic in Computer Science,* 1990.
[11] G. Behrmann, K.G. Larsen, J. Pearson, C. Weise, and Wang Yi, "Efficient Timed Reachability Analysis Using Clock Difference Diagrams," *Proc. Int'l Conf. Computer Aided Verification,* 1999.
[12] R.E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," *IEEE Trans. Computers,* vol. C-35, no. 8, pp. 677-691, Aug. 1986.
[13] D.L. Dill, "Timing Assumptions and Verification of Finite-State Concurrent Systems," *Proc. Int'l Conf. Computer Aided Verification,* 1989.
[14] M. Fujita, H. Fujisawa, and Y. Matsunaga, "Variable Ordering Algorithms for Ordered Binary Decision Diagrams and Their Evaluation," *IEEE Trans. Computer-Aided Design,* vol. 12, pp. 6-12, Jan. 1993.
[15] T.A. Henzinger, P.-H. Ho, and H. Wong-Toi, "HyTech: The Next Generation," *Proc. IEEE Real-Time Systems Symp.,* pp. 56-65, 1995.
[16] T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine, "Symbolic Model Checking for Real-Time Systems," *Proc. Seventh Symp. Logics in Computer Science,* pp. 394-406, 1992.
[17] S. Minato, N. Ishiura, and S. Yajima, "Shared Binary Decision Diagram with Attributed Edges for Efficient Boolean Function Manipulation," *Proc. ACM/IEEE 27th Design Automation Conf.,* pp. 52-57, 1990.
[18] J. Moller, J. Lichtenberg, H.R. Andersen, and H. Hulgaard, "Difference Decision Diagrams," *Proc. Ann. Conf. European Assoc. for Computer Science Logic,* Sept. 1999.
[19] S. Malik, A.R. Wang, R.K. Brayton, and A. Sangiovanni-Vincentelli, "Logic Verification Using Binary Decision Diagrams in a Logic Synthesis Environment," *Proc. Int'l Conf. Computer-Aided Design,* pp. 6-8, 1988.
[20] P. Pettersson and K.G. Larsen, "UPPAAL2k," *Bull. European Assoc. Theoretical Computer Science,* vol. 70, pp. 40-44, 2000.
[21] F. Wang, "Efficient Data-Structure for Fully Symbolic Verification of Real-Time Software Systems," *Proc. Int'l Conf. Tools and Algorithms for the Construction and Analysis of Systems,* 2000.
[22] F. Wang, "Region Encoding Diagram for Fully Symbolic Verification of Real-Time Systems," *Proc. 24th Int'l Computer Software and Applications Conf.,* Oct. 2000.
[23] F. Wang, "RED: Model-Checker for Timed Automata with Clock-Restriction Diagram," *Proc. Workshop Real-Time Tools,* Technical Report 2001-014, ISSN 1404-3203, Dept. of Information Technology, Uppsala Univ., Aug. 2001.
[24] F. Wang, "Symbolic Verification of Complex Real-Time Systems with Clock-Restriction Diagram," *Proc. Int'l Conf. Formal Techniques for Networked and Distributed Systems,* Aug. 2001.
[25] F. Wang, "Efficient Verification of Timed Automata with BDD-Like Data-Structures," *Proc. Int'l Conf. Verification, Model Checking and Abstract Interpretation,* 2003.
[26] F. Wang, "Model-Checking Distributed Real-Time Systems with States, Events, and Multiple Fairness Assumptions," *Proc. 10th Int'l Conf. Algebraic Methodology and Software Technology,* July 2004.
[27] F. Wang, G.-D. Hwang, and F. Yu, "TCTL Inevitability Analysis of Dense-Time Systems," *Proc. Eighth Conf. Implementation and Application of Automata,* July 2003.
[28] F. Wang, G.-D. Hwang, and F. Yu, "Numerical Coverage Estimation for the Symbolic Simulation of Real-Time Systems," *Proc. Int'l Conf. Formal Techniques for Networked and Distributed Systems,* Sept.-Oct. 2003.
[29] F. Wang, A. Mok, and E.A. Emerson, "Symbolic Model-Checking for Distributed Real-Time Systems," *Proc. First Int'l Symp. Formal Methods Europe,* Apr. 1993.
[30] H. Wong-Toi, "Symbolic Approximations for Verifying Real-Time Systems," PhD thesis, Stanford Univ., 1995.
[31] S. Yovine, "Kronos: A Verification Tool for Real-Time Systems," *Int'l J. Software Tools for Technology Transfer,* vol. 1, nos. 1-2, Oct. 1997.

Farn Wang received the BS degree in electrical engineering from National Taiwan University in 1982, the MS degree in computer engineering from Natinal Chiao-Tung University in 1984, and the PhD degree in computer sciences at the University of Texas at Austin in 1993. From September 1986 to May 1987, he was a research assistant at Telecommunication Laboratories, Ministry of Communications, R.O.C. From August 1993 to October 1997, he was an assistant research fellow at the Institute of Information Science (IIS), Academia Sinica, Taiwan, R.O.C. From October 1997 to July 2002, he was an associate research fellow at IIS. In August 2002, he became an associate professor in the Department of Electrical Engineering, National Taiwan University. He is interested in automating human verification experiences to develop verification tools with precision and efficiency. He has architected and implemented several tools for the verification of timed and hybrid systems. The tools include RED: a model-checker for timed and hybrid systems and SGM: an efficient and user-friendly verification tool for timed systems. He is a member of the IEEE Computer Society.