# Clock Restriction Diagram: Yet Another Data-Structure for Fully Symbolic Verification of Timed Automata*

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**

Modern model-checkers for real-time systems are usually built around symbolic manipulation procedures of zones, which mean behavior-equivalent dense-time state subspaces and are represented by sets of clock difference constraints. We propose *CRD (Clock Restriction Diagram)*, which is a BDD-like data-structure for recording sets of zones, with related set-oriented operations for fully symbolic verification of realt-time systems. CRD with zones in *reduced form* (or *reduced CRD* in short), which contains minimal number of clock inequalities to characterize a zone, are defined and can be used as a canonical representation for state space. Reduced CRD is more space-efficient than former proposal like DBM, NDD, CDD, and RED. We develop an algorithm that converts given CRDs to reduced CRDs without first computing its *closure form* (which records the shortest path distance between every pair of clocks) in some representations. We implemented CRD in a new version of our verifier `red` for timed automata and compare its performance with Kronos's and UPPAAL's against several benchmarks.

## 1 Introduction

Fully symbolic verification technologies using BDD-like structures[4, 8] can be efficient in both space and time complexities with intense data-sharing in the manipulation of state space representations. For hardware and untimed system veification, BDD has shown great success. But for real-time system verification, so far, all BDD-like structures[1, 7, 16, 17] have not performed as well as the popular DBM (difference-bounded matrix) [10] which is a 2-dimensional matrix and nothing BDD-like. For example, NDD[1] uses binary encoding for clock readings and its performance is very sensitive to timing-constant magnitude. CDD[7] is a decision diagram for state-space membership. To compute the unique representation of a state-space, CDD has to be transformed to *closure form*[10, 7] which records the shortest-path distances between all pairs of clocks and is very space-inefficient. RED[16, 17] encodes the ordering of fractional parts of clock readings in the variable ordering and has achieved very high space-efficiency for systems with large number of clocks and small timing constants. RED is indeed a canonical representation of timed automaton state subspaces. But for large timing constants, RED's performance degrades rapidly.

Most modern model-checkers are built around some symbolic manipulation procedures[13] of *zones* implemented in data-structures like DBM, NDD, CDD, or RED. A zone means a behaviorally equivalent state subspace of a timed automaton and is symbolically represented by a set of difference constraints between clock pairs. At this moment, DBM is still the most popular and efficient data-structure. DBM-technology generally handles the complexity of timing constant magnitude very well. But when the number of clocks increases, its performance also degrades rapidly. Thus a new data-structure which can handle both the

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

complexities of timing constant magnitude and clock number can still be of great practical value. Instead of using those above-mentioned BDD-like structures, we here propose yet another one: *CRD (Clock Restriction Diagram)* for the fully symbolic verificaiton of timed automata. CRD is not a decision diagram for state space membership. Instead it is like a database for *zones*. We devise the new data-structure CRD exactly because CRD acts like a database (recording device) and is more suitable for comparison and manipulation of sets of clock diference constraints.

Since many zones can represent the same subspace, like DBM and CDD, neither is CRD a canonical representation of zone-characterized state spaces. One solution is to convert all zones to their *closure form*[10, 7] (called shortest-path closure in [15]), which is the set of all pairwise clock difference constraints derived from the all-pair shortest-path distances, and only store their closure form. Such conversion is expensive and, as we shall illustrate in section 4, incurs large space consumption with data-structures like CDD and CRD.

An alternative solution for unique representation of state spaces is zones in their *reduced form* (called shortest-path reduction in [15]) which contains minimal number of clock difference constraints chosen by a policy. As shown in [15], DBM with zones in reduced form can be space-efficient. One of the main idea in this paper is to use CRD with zones in reduced form (or *reduced CRD* in short) as unique representation for state-spaces to enhance verification efficiency. However, in [15], reduced form is obtained from closure form. If we naively follow the approach, we still have to make space for those intermediate representations for sets of zones in their closure form and there will never be much reduction in space consumption. Another contribution of this paper is in the design of a symbolic algorithm which computes the reduced CRD without having to first compute some representations of their zones in closure form.

Here is our presentation plan. Section 2 briefly defines timed automata as our model for discussion. Section 3 discusses the concept of zones. Section 4 formally defines CRD and its manipulations. Especially, subsection 4.1 illustrates how reduced CRD can be more space-efficient than the other technologies and subsection 4.4 explains our algorithm for computing reduced CRDs. Section 5 reports experiments and compares the performance of our CRD-based verification tools with those of Kronos[5, 11, 19] and UPPAAL[6].

# 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 $x - x' \sim c$ where $x, x' \in X \cup \{0\}$, "$\sim$" is one of $\leq, <, =, >, \geq$, and $c$ is an integer constant.

**Definition 1** **automata** A timed automaton $A$ is given as a tuple $\langle X, Q, q_0, I, \mu, T, \tau, \pi \rangle$ with the following restrictions. $X$ is the set of clocks. $Q$ is the set of modes. $q_0$ is the initial mode. $I \in B(X)$ is the initial condition on clocks. $\mu : Q \mapsto B(X)$ defines the invariance condition of each mode. $T \subseteq Q \times Q$ is the set of transitions. $\tau : T \mapsto B(X)$ and $\pi : T \mapsto 2^X$ respectively defines the triggering condition and the clock set to reset of each transition. ||

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

**Definition 2** **states** Given a timed automaton $A = \langle X, Q, q_0, I, \mu, T, \tau, \pi \rangle$, A state $\nu$ of $A$ is a valuation of $X \cup \{\texttt{mode}\}$ such that

- $\nu(\texttt{mode}) \in Q$ is the mode of $A$ in $\nu$ 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 $\nu$ 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 $\nu$ except that for every $x \in \bar{X}$, $\nu\bar{X}(x) = 0$.

**Definition 3** **runs** Given a timed automaton $A = \langle X, Q, q_0, I, \mu, T, \tau, \pi \rangle$, 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
  - $(\nu_k(\texttt{mode}), \nu_{k+1}(\texttt{mode})) \in T$ and
  - $\nu_k + (t_{k+1} - t_k) \models \tau(\nu_k(\texttt{mode}), \nu_{k+1}(\texttt{mode}))$ and
  - $(\nu_k + (t_{k+1} - t_k))\pi(\nu_k(\texttt{mode}), \nu_{k+1}(\texttt{mode})) = \nu_{k+1}$.

||

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 $\eta$, 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 $\nu$-runs $\rho$, $\nu \models I \wedge (\texttt{mode} = q_0)$ implies $\rho \models \eta$. Our verification framework is *safety analysis problem* that when given $A$ and $\eta$, asks whether $A \models \eta$.

# 3 Zones and its canonical forms

Let $\mathcal{Z}$ be the set of integers. Given $c \geq 0$ and $c \in \mathcal{Z}$, let $\mathcal{I}_c$ be $\{\infty\} \cup \{d \mid d \in \mathcal{Z}; -c \leq d \leq c\}$. Also for any $d \in \mathcal{Z}$, $d + \infty = \infty + d = \infty$.

Given a safety analysis problem for a timed automaton $A$ with biggest timing constant $C_A$ used in $A$, a zone is a convex subspace of $\mathcal{R}^{|X|}$ constrained by half spaces represented by inequalities like $x - x' \sim d$, with $x, x' \in X \cup \{0\}$, $\sim \in \{\text{“}\leq\text{”}, \text{“}<\text{”}\}$, and $d \in \mathcal{I}_{C_A}$, such that when $d = \infty$, $\sim$ must be “$<$”. For convenience, let $\mathcal{B}_c = \{(\sim, d) \mid \sim \in \{\text{“}\leq\text{”}, \text{“}<\text{”}\}; d \in \mathcal{I}_c; d = \infty \Rightarrow \sim = \text{“}<\text{”}\}$. With respect to given $X$ and $C_A$, the set of all zones is finite. Formally, a zone $\zeta$ can be defined as a mapping $(X \cup \{0\})^2 \mapsto \mathcal{B}_{C_A}$. Alternatively, we may also define a zone $\zeta$ as the set $\{x - x' \sim d \mid \zeta(x, x') = (\sim, d)\}$. In the following, we shall use the two equivalent definitions flexibly as we see fit.

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

A few terms to define before we explain the second candidate for zone canonical form. Two clocks $x, x' \in X \cup \{0\}$ are *equivalent* in a zone $\zeta$, in symbols $x \equiv_\zeta x'$, iff $\exists d \in \mathcal{Z}(x - x' \leq -d \in \zeta^C \wedge x' - x \leq d \in \zeta^C)$. For convenience, assume that $X = \{x_1, \ldots, x_n\}$ and 0 is also named $x_0$. If $Y = \{x_{i_1}, x_{i_2}, \ldots, x_{i_k}\} \subseteq X \cup \{0\}$ is a maximal set of equivalent clocks in $\zeta$ such that $i_1 < i_2 < \ldots < i_k$, let $\min_{\equiv\zeta}(x_{i_j}) = x_{i_1}$ for all $1 \leq j \leq k$. Given $x - x' \sim d \in \zeta^C$, $(x, x')$ is *redundant* in $\zeta$ iff $x \not\equiv_\zeta x'$ and there is a $\bar{x} \in X \cup \{0\}$, with $x - \bar{x} \sim_1 d_1, \bar{x} - x' \sim_2 d_2 \in \zeta^C$, such that $x \not\equiv_\zeta \bar{x} \not\equiv_\zeta x'$ and $d = d_1 + d_2 \wedge (\sim = \text{“}<\text{”} \Rightarrow (\sim_1 = \text{“}<\text{”} \vee \sim_2 = \text{“}<\text{”}))$.

Another candidate for the canonical form of zones is the *reduced form* (called shortest-path reduction in [15]) which records only minimum number of constraints for each zone. In the following, we briefly restate from [15] how to convert a given zone $\zeta$ to its zone in reduced form, in symbols $\zeta^R$, in five steps.

**step 1:** Calculate $\zeta^C$ (with all-pairs shortest-path algorihtm).

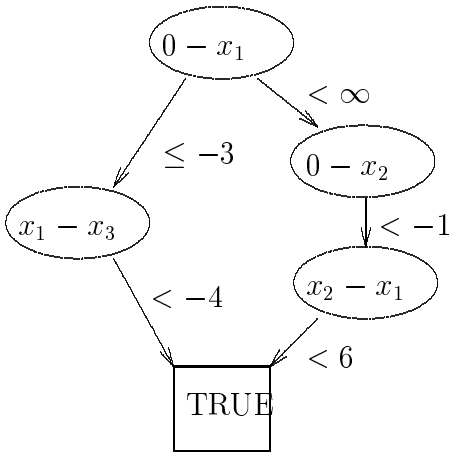


Figure 1: an example CRD

**step 2:** Partition $X \cup \{0\}$ into equivalent classes according to $\zeta^C$.

**step 3:** Difference constraints between elements in the same equivalent class are recorded with the policy to make a simple cycle. Suppose $Y = \{x_{i_1}, x_{i_2}, \ldots, x_{i_k}\} \subseteq X \cup \{0\}$ is a maximal set of equivalent clocks in $\zeta^C$ such that $i_1 < i_2 < \ldots < i_k$. Then $\zeta^R(x_{i_k}, x_{i_1}) = \zeta^C(x_{i_k}, x_{i_1})$ and for all $1 \leq j < k$, $\zeta^R(x_{i_j}, x_{i_{j+1}}) = \zeta^C(x_{i_j}, x_{i_{j+1}})$. In this way, elements $x_{i_1}, x_{i_2}, \ldots, x_{i_k}$ are connected in a simple cycle by constraints with bounds $\neq \infty$.

**step 4:** For any two $x, x' \in X \cup \{0\}$, if $(\min_{\equiv\zeta}(x), \min_{\equiv\zeta}(x'))$ is not redundant in $\zeta^C$, then $\zeta^R(\min_{\equiv\zeta}(x), \min_{\equiv\zeta}(x')) = \zeta^C(\min_{\equiv\zeta}(x), \min_{\equiv\zeta}(x'))$.

**step 5:** For every other pair of $x, x'$ not dealt with in steps (3) and (4), $x - x' < \infty \in \zeta^R$.

It is shown in [15] that $\zeta^C = (\zeta^R)^C$ and DBM with zones in reduced form can significantly save space in model-checking.

# 4 Clock Restriction Diagram

## 4.1 Definition

CRD is a directed acyclic graph for representation of sets of zones. It has similar structure as BDD without FALSE terminal. Each of the pairs $(x, x') \in (X \cup \{0\})^2$ is treated as an evaluation variable. By fixing an evaluation order, we can construct a CRD just as BDD, CDD, or RED. For example, given $C_A = 10$, the CRD for a set $\{\{0 - x_1 \leq -3, x_1 - x_3 < -4\}, \{0 - x_2 < -1, x_2 - x_1 < 6\}\}$ of two zones (constraints of the form $x - x' < \infty$ are omitted) is in figure 1. In CRD, a missing constraints on differences of clock pairs, say $x, x'$, is interpreted as $x - x' < \infty$. Thus in the root vertex, even no constraint is on $0 - x_1$ in the latter zone, we still construct an arc with $0 - x_1 < \infty$ from the root vertex. This is one major difference of our CRD from decision diagrams like CDD which interprets a missing restriction on $x, x'$ as $-\infty < x - x' < \infty$ with an implied lowerbound of $-\infty$ on $x - x'$.

An *evaluation index* $\omega : (X \cup \{0\})^2 \cup \{true\} \mapsto \{0, 1, \ldots, |(X \cup \{0\})^2|\}$ is a mapping such that $\omega(true) = |(X \cup \{0\})^2|$ and for every two $e, e' \in (X \cup \{0\})^2 \cup \{true\}$, $\omega(e) \neq \omega(e')$.

**Definition 4** *Clock Restriction Diagram (CRD)* A CRD is a labeled directed acyclic graph $D = (V, \phi, E, \lambda)$, with single source and single sink, constructed under a given evaluation index $\omega$ such that

- $V$ is the set of vertices;

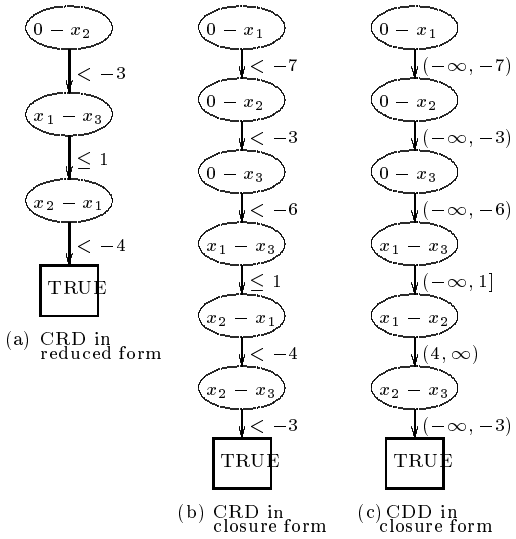


Figure 2: comparison of space-efficiency

- $\phi : V \mapsto (X \cup \{0\})^2 \cup \{true\}$ defines the evaluation variable at each vertex;
- $E \subseteq V \times V$ is the set of arcs such that for every $(v, v') \in E$, $\omega(\phi(v)) < \omega(\phi(v'))$ (i.e., evaluation ordering must be respected); and
- $\lambda : E \mapsto \mathcal{I}_{C_A}$ such that for every $(v, v'), (v, v'') \in E$, $v' \neq v'' \Rightarrow \lambda(v, v') \neq \lambda(v, v'')$.

There is at most one $v \in V$ such that $\phi(v) = true$ and this $v$ is the single sink of the CRD. ||

There are two simple reductions which can be performed on a CRD. First, a vertex with a single outgoing arc with label $(<, \infty)$ can be bypassed and discarded. Second, a zone $\zeta_1$ recorded in a CRD can be discarded if there is another $\zeta_2$ in the same CRD such that $\zeta_1$ is identical to $\zeta_2$ except that for exactly one pair $x, x' \in X \cup \{0\}$, $x - x' \sim_1 d_1 \in \zeta_1$, $x - x' \sim_2 d_2 \in \zeta_2$, and either $d_1 < d_2$ or $d_1 = d_2 \wedge \sim_1 = \text{“}<\text{”} \wedge \sim_2 = \text{“}\leq\text{”}$. The first reduction is sound because a constraint like $x - x' < \infty$ is in fact no constraint. The second reduction is sound because $\zeta_1$ obviously means a subspace of $\zeta_2$. Both reductions can be performed locally from each CRD vertex.

In average, CRD with zones in reduced form (reduced CRD in short) is much more space-efficient than previous data-structures[1, 7, 16, 17]. For example, given $X = \{x_1, x_2, x_3\}$ and $C_A = 10$, in figure 2, we have the representations of zone $\{0 - x_2 < -3, x_1 - x_3 \leq 1, x_2 - x1 < -4\}$ in reduced CRD (figure 2(a)), in CRD with zones in closure form (figure 2(b)), and in CDD with zones in closure form (called tightened form in [7]) without FALSE terminal vertex (figure 2(c)). It is easy to see that as the number of clocks increases, reduced CRD will perform better and better.

## 4.2 Basic manipulations on CRD

Since CRD is a recording device for zones rather than a decision device for state-space membership, its operations are more set-oriented than proposition-oriented. For convenience of discussion, given a CRD, we may just represent it as the set of zones recorded in it. Here we present the basic manipulations on CRDs which will be useful in explaining our algorithm. Set-union ($\cup$), set-intersection ($\cap$), and set-exclusion ($-$) of two zone sets respectively represented by two CRDs are straightforward. For example, given CRDs $D_1 : \{\zeta_1, \zeta_2\}$ and $D_2 : \{\zeta_2, \zeta_3\}$, $D_1 \cap D_2$ is the CRD for $\{\zeta_2\}$; $D_1 \cup D_2$ is the CRD for $\{\zeta_1, \zeta_2, \zeta_3\}$; and $D_1 - D_2$ is the CRD for $\{\zeta_1\}$.

Set-extraction ($|$) selects zones satisfying certain features from a zone set. Suppose $D$ is the CRD for

$\{\zeta_1, \ldots, \zeta_k\}$ and $D'$ is the CRD for $\{\zeta'_1, \ldots, \zeta'_h\}$, then $D|D' = \{\zeta_i \mid 1 \leq i \leq k; \exists 1 \leq j \leq h \forall x - x' \sim d \in \zeta'_j (d \neq \infty \Rightarrow x - x' \sim d \in \zeta_i)\}$.

Given two zones $\zeta_1$ and $\zeta_2$, $\zeta_1 * \zeta_2$ is a new zone representing the space-intersection of $\zeta_1$ and $\zeta_2$. Formally speaking, for every $x, x'$ with $\zeta_1(x, x') = (\sim_1, d_1)$ and $\zeta_2(x, x') = (\sim_2, d_2)$, $\zeta_1 * \zeta_2(x, x') = (\sim_1, d_1)$ if $d_1 < d_2 \vee (d_1 = d_2 \wedge \sim_1 =$ "$<$"); or $\zeta_1 * \zeta_2(x, x') = (\sim_2, d_2)$ otherwise. Space-intersection $(*)$ of two CRDs $D_1$ and $D_2$, in symbols $D_1 * D_2$, is a new CRD for $\{\zeta_1 * \zeta_2 \mid \zeta_1 \in D_1; \zeta_2 \in D_2\}$.

### 4.3 CRD and BDD

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

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

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

### 4.4 Computing reduced CRD

In [15], the reduced form is derived from closure form of DBM. Following that approach, it is impossible to cut down the space requirement since we still have to make space for the closure form of some intermediate representations. We used a CRD+BDD to compute reduced CRD without having to first compute its closure form. Remember in [15], clocks are classified into equivalent classes. We introduce four auxiliary variables START, STOP, BOUND, and EQUIVALENT encoded with BDD variables. START and STOP range in $X \cup \{0\}$. BOUND ranges in $\mathcal{B}_{C_A}$. EQUIVALENT is either *true* or *false*. Given a set $Z$ of zones and a particular valuation $(x, x', (\sim, d), e)$ of (START, STOP, BOUND, EQUIVALENT), we can define a mapping $\Phi_Z$ from $(x, x', (\sim, d), e)$ to a maximal subset of $Z$ such that for any $\zeta \in \Phi_Z(x, x', (\sim, d), e)$, $\zeta^C(x, x') = (\sim, d)$ and $e \Leftrightarrow x \equiv_\zeta x'$. Our idea is to add these four variables to a given CRD and change the diagram into a mapping from valuations of the four variables to CRDs. With information in such mappings, we can decide whether an existing constraint should be deleted or an absent constraint should be added in the reduced CRD.

Table 1 gives a skeleton view of our algorithm to compute reduced CRD. Initially $D$ is a pure CRD. Line (1) adds the four auxiliary variables to $D$ and make it a CRD+BDD. Lines (2) eliminates empty zones represented by inconsistent sets of constraints. Line (3) should be obvious from its procedure name. Lines (4) and (6) respectively add and eliminate constraints between elements in the same equivalent classes. Line (5) adds constraints between the $\min_{\equiv}$'s of different equivalent classes. Line (7) deletes constraints between inequivalent elements in $X \cup \{0\}$ if one of them is not a $\min_{\equiv}$. Line (8) deletes redundant constraints between $\min_{\equiv}$'s of different equivalent classes. Line (9) gets rid of the auxiliary variables and makes $D$ a pure and reduced CRD.

Due to page-limit, we shall only describe details of two subprocedures in table 1. The algorithm in table 2 shows how we compute the values of variables START, STOP, and BOUND. The algorithm is in the style of Kleene closure. Here we use a short Boolean expression like $x - x' \sim d \wedge \text{START} = x \wedge \text{STOP} = x'$ to mean a CRD+BDD for the expression. Especially atom $x - x' \sim d$ means that the constraint $x - x' \sim d$ is in the zone. Line (11) initializes the BOUND variable for all pairs to value "$(<, \infty)$," meaning no constraint.

```
reduced(D) {
  D := Kleene(D);                                                    (1)
  D := eliminate_negative_self_cycles(D);                            (2)
  D := partition_equivalent_classes(D);                              (3)
  D := add_constraints_in_equivalent_classes(D);                     (4)
  D := add_constraints_between_mins(D);                              (5)
  D := eliminate_constraints_in_equivalent_classes(D);               (6)
  D := eliminate_constraints_between_nonmins(D);                     (7)
  D := eliminate_redundancy_between_mins(D);                         (8)
  D := eliminate_variable(D, {START, STOP, BOUND, EQUIVALENT});      (9)
  return D;
} /* reduced() */
```

Table 1: Symbolic procedure for calculating reduced CRD.

The loop starting at line (12) is to further initialize the BOUND variable values according to the constraints present in the zones. The loop starting at line (13) is pretty much a Kleene-closure routine, which changes the BOUND variable values by considering all path compositions. In line (15), $D \cap D_1 \cap D_2$ is the set of zones $\zeta$ such that $x - \bar{x} \sim_1 d_1, \bar{x} - x' \sim_2 d_2 \in \zeta^C$. The Boolean expression in the big parenthesis at the end of line (15) serves to filter in those zones in which $(x, x')$ is truly redundant.

Suppose we have a CRD+BDD $D$ with the four auxiliary variables properly calculated. Procedure `eliminate_redundancy_between_mins`() in table 1 can be constructed with the procedure in table 3 nested in a triple loop on $x, \bar{x}, x'$. After line (21) in table 3 is executed, $D_0$ is the CRD+BDD for zones in which $x, x'$ are inequivalent. Similarly $D_1$ ($D_2$), after line (22) (line (23)) is executed, is the CRD+BDD for zones in which $x, \bar{x}$ ($\bar{x}, x'$) are inequivalent. After line (26) is executed, $H_0$ is the pure CRD (without BDD variables) for zones $\zeta$ such that $x - x' \sim d \in \zeta^C$ and $x \not\equiv_\zeta x'$. Similar thing can be said after executing lines (27), (28) for $H_1, H_2$ respectivley. Then after the intersection in line (29), $H_0$ becomes the CRD for all zones $\zeta$ such that $x \not\equiv_\zeta \bar{x} \not\equiv_\zeta x' \not\equiv_\zeta x$ and $(x, x')$ is redundant due to $\bar{x}$ in $\zeta$.

We have to point out that the presentation of the procedures is for conciseness and not for execution performance. For example, in our implementation, line (24) is replaced by search routines through CRD while line (25) is replaced by ranges mapping of variable BOUND.

# 5 Implementation and experiments

## 5.1 Implementation

We have implemented our CRD-technology in version 3 of our tool `red` which was previously announced in [16, 17] and supports the modelling and safety-analysis of real-time systems with multiprocesses, pointer data-structures, and synchronizations (synchronous send and receive) from one process to another. The new version, together with benchmarks, will soon be available at:

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

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

Forward analysis is default in `red` while backward analysis is invoked by option "`-b`." We have also implemented two reduction techniques in `red`. The first is reduction by elimination of inactive variables[14, 18] which is always executed. A variable is inactive in a state iff it is not read in any computation from the state before its content is overwritten. Contents of inactive variables can be omitted from state information

$$
\begin{array}{l}
\texttt{Kleene}(D)\ \{ \\
\quad D := D \cap \bigvee_{x,x' \in X \cup \{0\}} (\mathrm{START} = x \wedge \mathrm{STOP} = x' \wedge \mathrm{BOUND} = (<, \infty)); \qquad (11) \\
\quad \text{for all } x, x' \in X \cup \{0\}, \text{ with } x \neq x', \text{ and } (\sim, d) \in \mathcal{B}_{C_A}, \{ \qquad (12) \\
\qquad B := D \left| \begin{pmatrix} x - x' \sim d \wedge \mathrm{START} = x \wedge \mathrm{STOP} = x' \\ \wedge \quad \mathrm{BOUND} \in \{(\sim', d') \,|\, d' \in [d+1, \infty) \vee (\sim' = \text{“}\leq\text{”} \wedge d' = d)\} \end{pmatrix}; \right. \\
\qquad D := (D - B) \cup \texttt{set\_variable}(B, \mathrm{BOUND}, (\sim, d)); \\
\quad \} \\
\quad \text{for all } \bar{x} \in X \cup \{0\}, \text{ do } \{ \qquad (13) \\
\qquad \text{for all } x, x' \in X \cup \{0\} \text{ such that } x \neq x' \neq \bar{x} \neq x, \text{ do } \{ \\
\qquad\quad \text{for all } (\sim_1, d_1), (\sim_2, d_2) \in \mathcal{B}_{C_A}, \{ \qquad (14) \\
\qquad\qquad D_1 := D \cap (\mathrm{START} = x \wedge \mathrm{STOP} = \bar{x} \wedge \mathrm{BOUND} = (\sim_1, d_1)); \\
\qquad\qquad D_1 := \texttt{eliminate\_variable}(D_1, \{\mathrm{START}, \mathrm{STOP}, \mathrm{BOUND}\}); \\
\qquad\qquad D_2 := D \cap (\mathrm{START} = \bar{x} \wedge \mathrm{STOP} = x' \wedge \mathrm{BOUND} = (\sim_2, d_2)); \\
\qquad\qquad D_2 := \texttt{eliminate\_variable}(D_2, \{\mathrm{START}, \mathrm{STOP}, \mathrm{BOUND}\}); \\
\qquad\qquad B := D \cap D_1 \cap D_2 \cap \left( \begin{array}{l} \mathrm{START} = x \wedge \mathrm{STOP} = x' \\ \wedge \mathrm{BOUND} \in \left\{ (\sim', d') \left| \begin{array}{l} d' \in [d_1 + d_2 + 1, \infty) \\ \vee (\sim' = \text{“}\leq\text{”} \wedge d' = d_1 + d_2) \end{array} \right. \right\} \end{array} \right); \qquad (15) \\
\qquad\qquad \text{if } \sim_1 = \text{“}\leq\text{”} \text{ and } \sim_2 = \text{“}\leq\text{”}, \text{ then } \tilde{\sim} := \text{“}\leq\text{”}; \text{ else } \tilde{\sim} := \text{“}<\text{”}; \\
\qquad\qquad D := (D - B) \cup \texttt{set\_variable}(B, \mathrm{BOUND}, (\tilde{\sim}, d_1 + d_2)); \\
\qquad\quad \} \\
\qquad \} \\
\quad \} \\
\quad \text{return } D; \\
\} \ /^* \ \texttt{Kleene()} \ ^*/
\end{array}
$$

Table 2: Symbolic procedure for classifying zones according to shortest path distances.

without any effect on the computations. The second reduction technique is reduction by process symmetry[12, 14, 18] and is invoked with option "`s`." Two states can become equivalent after some permutation of process identifiers in a symmetric system. The technique replaces a group of permutationally equivalent regions by a single member in such a group.

## 5.2 Experiments

We choose to compare performance with Kronos[5, 11, 19] (version 2.4.4) and UPPAAL[6] (version 3.0.39) which are perhaps the two best-known model-checkers for real-time systems. UPPAAL supports on-the-fly forward analysis with various searching and reduction strategies. Kronos supports forward and backward model-checking of TCTL[2].

Two experiments have been carried out. The first is for performance w.r.t. number of clocks while the second is for performance w.r.t. magnitude of timing constants. Three benchmarks are used and briefly explained in the following. The first benchmark is a version of the famous Fischer's timed mutual exclusion algorithm [3, 14, 16, 18]. The algorithm relies on a global lock and a local clock per process to control access to the critical section. Two timing constants used are 10 and 19. The input file of two processes to `red` is in appendix A. The property to be safety-analyzed is that at any moment, no more than two processes are in the critical section.

The second benchmark is a timed production line in which a production item is placed on the track in every 20 or more seconds. In a system which can hold up to $n$ items on the track, an item is expected to receive processing after $20(n+1)$ seconds on the track. After having being processed for less than 20 seconds, the processed item gets off the track. The track is modelled by an automaton with 2 modes and

```
eliminate_one_redundancy_between_mins(D, x, x̄, x') {
  D_0 := D ∩ (START = x ∧ STOP = x' ∧ ¬EQUIVALENT);
  D_0 := eliminate_variable(D_0, {START, STOP, EQUIVALENT});                    (21)
  D_1 := D ∩ (START = x ∧ STOP = x̄ ∧ ¬EQUIVALENT);
  D_1 := eliminate_variable(D_1, {START, STOP, EQUIVALENT});                    (22)
  D_2 := D ∩ (START = x̄ ∧ STOP = x' ∧ ¬EQUIVALENT);
  D_2 := eliminate_variable(D_2, {START, STOP, EQUIVALENT});                    (23)
  for all (∼, d), (∼_1, d_1), (∼_2, d_2) ∈ B_{C_A},                             (24)
    if d > d_1 + d_2 ∨ (d = d_1 + d_2 ∧ (∼= "<" ⇒ (∼_1= "<"∧ ∼_2= "<"))), then {  (25)
      H_0 := eliminate_variable(D_0 ∧ BOUND = (∼, d), {BOUND});                 (26)
      H_1 := eliminate_variable(D_1 ∧ BOUND = (∼_1, d_1), {BOUND});             (27)
      H_2 := eliminate_variable(D_2 ∧ BOUND = (∼_2, d_2), {BOUND});             (28)
      H_0 := H_0 ∩ H_1 ∩ H_2;                                                   (29)
      D := (D − H_0) ∪ variable_eliminate(H_0, x − x');
    }
  }
  return D;
} /* eliminate_one_redundancy_between_mins() */
```

Table 3: Symbolic procedure for elimination of an redundancy instance.

a local clock while each production item is modelled by one with 4 modes and a local clock. The input file to `red` for systems with track capacity of two production items is in appendix B. We want to verify that at any moment, at most one item is being processed.

The third benchmark is the FDDI token-passing protocol in a ring network from [5, 11]. We need one process to model the ring network and the other processes to model the stations. The input to `red` for FDDI with two station processes is given in appendix C. 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. Each station process uses two local clocks $x$ and $y$ and requests for 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 (detected with $x = 20$) transmission, and second, in the last cycle of token-passing, all processes together have not used much network time (detected with $y <= 50n + 20$ where $n$ is the number of stations). The second local clock $y$ is used to accumulate the time used by all processes in one token-passing cycle of the network. The property to verify is that no two processes can be sending messages (in either synchronous mode or asynchronous mode) at the same time.

### 5.2.1 Performance w.r.t. number of clocks

Table 4 shows performance data from Kronos, UPPAAL, and `red` w.r.t. implementations of the benchmarks with various numbers of concurrent processes. `red` is executed both with and without symmetry reduction option while UPPAAL and Kronos are invoked without symmetry reduction. UPPAAL is invoked with options "`a`," "`S`," and "`T`." The performance data shows that even without symmetry reduction, CRD-technology is still more space-efficient and scales better w.r.t. number of clocks. For the timed production line benchmark, according to the trend of the space consumption, we expect that `red` can verify the benchmark with up to six production items within 256 MB space. Also symmetry reduction very much alleviates the factorial explosion in the many different ordering among the clock values.

We remind the readers that compared to those well-established tools like Kronos and UPPAAL, our implementation does not employ too many reduction techniques, e.g., the global state-space reduction technique used in [15]. In the future, when such techniques are incorporated with CRD-based technology, we believe

| benchmarks | concurrency | Kronos | UPPAAL | `red` | `red` w. symmetry |
|---|---|---|---|---|---|
| Fischer's mutual exclusion | 3 processes | 0.03s$^{\dagger}$ | 0.05s$^{*}$ | 5.73s/73k$^{*}$ | 4.3s/21k$^{\otimes}$ |
| | 4 processes | 0.14s$^{\dagger}$ | 2s$^{*}$ | 126s/292k$^{*}$ | 21s/40k$^{\otimes}$ |
| | 5 processes | 0.989s$^{\dagger}$ | 286s$^{*}$ | 2147s/1543k$^{*}$ | 73s/77k$^{\otimes}$ |
| | 6 processes | O/M$^{\dagger}$ | O/M$^{*}$ | 19689s/8868k$^{*}$ | 222s/136k$^{\otimes}$ |
| | 7 processes | O/M$^{\dagger}$ | O/M$^{*}$ | 257235s/55056k$^{*}$ | 556s/236k$^{\otimes}$ |
| | 8 processes | O/M$^{\dagger}$ | O/M$^{*}$ | O/M$^{\otimes}$ | 1320s/353k$^{\otimes}$ |
| | 9 processes | O/M$^{\dagger}$ | O/M$^{*}$ | O/M$^{\otimes}$ | 2607s/509k$^{\otimes}$ |
| | 10 processes | O/M$^{\dagger}$ | O/M$^{*}$ | O/M$^{\otimes}$ | 4925s/674k$^{\otimes}$ |
| Timed production line | 3 items | 0.31s$^{\dagger}$ | 0.24s$^{*}$ | 41.9s/160k$^{*}$ | 4.32s/48k$^{\otimes}$ |
| | 4 items | 67.51s$^{\dagger}$ | 55.6s$^{*}$ | 1251s/830k$^{*}$ | 19.6s/90k$^{\otimes}$ |
| | 5 items | O/M$^{\dagger}$ $>$ 1252s | O/M$^{\otimes}$ $>$ 8205s | 20798s/5655k$^{\otimes}$ | 64.1s/141k$^{\otimes}$ |
| FDDI token-ring passing | 11 stations | 399s$^{\dagger}$ | 4057s$^{\otimes}$ | 725s/720k$^{\otimes}$ | N/A |
| | 12 stations | O/M$^{\dagger}$ | O/M$^{\otimes}$ | 1087s/864k$^{\otimes}$ | N/A |
| | 15 stations | O/M$^{\dagger}$ | O/M$^{\otimes}$ | 3482s/1394k$^{\otimes}$ | N/A |
| | 20 stations | O/M$^{\dagger}$ | O/M$^{\otimes}$ | 16528s/2671k$^{\otimes}$ | N/A |
| | 25 stations | O/M$^{\dagger}$ | O/M$^{\otimes}$ | 56791s/4514k$^{\otimes}$ | N/A |

$^{*}$: data collected on a Pentium II 333MHz with 192MB memory running LINUX;
$^{\otimes}$: data collected on a Pentium II 366MHz with 256MB memory running LINUX;
$^{\dagger}$: data collected on a Pentium III 800MHz with 256MB memory running LINUX;
s: seconds; k: kilobytes of memory in data-structure; O/M: Out of memory; N/A: not available;

Table 4: Performance data of scalability w.r.t. number of processes

| Tools | # proc | $C_A = 38$ | $C_A = 76$ | $C_A = 152$ | $C_A = 304$ | $C_A = 608$ | $C_A = 1216$ |
|---|---|---|---|---|---|---|---|
| Kronos | 3 | 0.023s$^{\dagger}$ | 0.023s$^{\dagger}$ | 0.013s$^{\dagger}$ | 0.034s$^{\dagger}$ | 0.026s$^{\dagger}$ | 0.032s$^{\dagger}$ |
| | 4 | 0.134s$^{\dagger}$ | 0.124s$^{\dagger}$ | 0.124s$^{\dagger}$ | 0.124s$^{\dagger}$ | 0.114s$^{\dagger}$ | 0.120s$^{\dagger}$ |
| UPPAAL | 3 | 0.03s$^{\otimes}$ | 0.04s$^{\otimes}$ | 0.04s$^{\otimes}$ | 0.05s$^{\otimes}$ | 0.05s$^{\otimes}$ | 0.04s$^{\otimes}$ |
| | 4 | 1.43s$^{\otimes}$ | 1.43s$^{\otimes}$ | 1.46s$^{\otimes}$ | 1.46s$^{\otimes}$ | 1.45s$^{\otimes}$ | 1.44s$^{\otimes}$ |
| `red` | 3 | 5.68s/65k$^{\otimes}$ | 5.68s/65k$^{\otimes}$ | 5.74s/65k$^{\otimes}$ | 5.73s/65k$^{\otimes}$ | 5.69s/65k$^{\otimes}$ | 5.69s/65k$^{\otimes}$ |
| | 4 | 85.8s/279k$^{\otimes}$ | 85.4s/279k$^{\otimes}$ | 85.4s/279k$^{\otimes}$ | 85.6s/279k$^{\otimes}$ | 85.5s/279k$^{\otimes}$ | 85.5s/279k$^{\otimes}$ |

$^{\otimes}$: data collected on a Pentium II 366MHz with 256MB memory running LINUX;
$^{\dagger}$: data collected on a Pentium III 800MHz with 256MB memory running LINUX;
s: seconds; k: kilobytes of memory in data-structure; O/M: Out of memory; N/A: not available;

Table 5: Performance data of scalability w.r.t. timing-constant magnitude

there will be much room for performance enhancement.

### 5.2.2 Performance w.r.t. timing constant magnitude complexity

The performance of some previous technologies, e.g. NDD and RED, does not scale very well to the magnitude of timing constant. The data in table 5 is collected by runnning Kronos, UPPAAL, and `red` against Fischers' mutual exclusion algorithm with various timing constant magnitudes. The table shows that CRD is at least as good as DBM technology as long as performance scalability with respect to timing constant complexity is concerned. In fact, the data may imply that the performance of CRD-technology can be irrelevant to the timing constant complexity in average cases.

# 6 Conclusion

After many previous attempts[7, 16, 17] of BDD-like data-structures for fully symbolic verification of real-time systems, we here propose yet another new data-structure: CRD and carry out implementation to show the possibility of performance enhancement through the data-sharing capability of BDD-like data-structures. However, the space-efficiency reported in the paper will not be possible without the design of our algorithm to compute reduced CRD without having to first compute closure CRD.

CRD seems also very compatible with many reductions techniques in the literature, for example, partial-order reduction[9] and global state-space reduction[15]. We plan to incorporate those techniques in our implementation to further enhance the performance.

# References


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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[19] 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 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 <= 10 may x:= 0; lock:= P; goto waiting; }
mode waiting true {
  when (x > 19 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];
```

# B Timed production line with track capacity of two

```
#define WAIT_TIME 60  /* 20(n+1), n is the track capacity */
#define PROC_TIME  20

process count = 3;
local clock x;
global synchronizer request_on_track;

/*** track ************************************************/
mode track_open true { when ?request_on_track true may x:=0; goto track_closed; }
mode track_closed x <= PROC_TIME { when x >= PROC_TIME may goto track_open; }

/*** production item  *************************************************/
mode item_idle true { when true may goto item_request; }
mode item_request true { when !request_on_track true may x:= 0; goto item_on_track; }
mode item_on_track x <= WAIT_TIME { when x >= WAIT_TIME may x:= 0; goto item_processing; }
mode item_processing x < PROC_TIME { when true may goto item_idle; }

initially track_open[1] and item_idle[2] and item_idle[3];

risk item_processing[2] and item_processing[3];
```

# C FDDI with two station processes

```
#define SA 20
#define td  0
#define TRTT  120 /* 50n+2, n is the number of stations. */

process count = 3; /* 3 is for ring, the others for stations. */
local clock x, y;
```

```
global synchronizer tt1, tt2, rt1, rt2;

/*** Ring network ***/
mode ring_to_1 x =< td { when !tt1 x = td  may goto ring_1; }
mode ring_1 true { when ?rt1 true  may x:= 0; goto ring_to_2; }
mode ring_to_2 x =< td { when !tt2 x = td  may goto ring_2; }
mode ring_2 true { when ?rt2 true  may x:= 0; goto ring_to_1; }

/*** Station 1 ***/
mode station_1_idle true {
when ?tt1 true may y:= x; x:= 0; goto station_1_sync;
}
mode station_1_sync x <= SA {
when !rt1 x = SA and y => TRTT may goto station_1_idle;
when x = SA and y < TRTT may goto station_1_async;
}
mode station_1_async y =< TRTT {
when !rt1 true may goto station_1_idle;
}

/*** Station 2 ***/
mode station_2_idle true {
when ?tt2 true may y:= x; x:= 0; goto station_2_sync;
}
mode station_2_sync x <= SA {
when !rt2 x = SA and y => TRTT may goto station_2_idle;
when x = SA and y < TRTT may goto station_2_async;
}
mode station_2_async y =< TRTT {
when !rt2 true may goto station_2_idle;
}

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

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