International Journal of Foundations of Computer Science
© World Scientific Publishing Company

# SAT-based Model Checking for Region Automata*

FANG YU
*Institute of Information Science, Academia Sinica*
*Taipei 115, Taiwan, Republic of China* †

and

BOW-YAW WANG
*Institute of Information Science, Academia Sinica*
*Taipei 115, Taiwan, Republic of China* ‡

ABSTRACT

We propose a new SAT-based model checking algorithm to solve the reachability problem of real-time systems. In our algorithm, the behavior of region automata is encoded as Boolean formulas, and hence any SAT solver can be used to explore the region graph efficiently. Although our SAT-based algorithm performs better than other algorithms in flaw detection, it is less effective in proving properties. To overcome the problem, we incorporate a complete inductive method in our algorithm to improve the performance when the property is satisfied. We implement both algorithms in a tool called xBMC and report experimental results. The experiments show that the combination of efficient encoding and inductive methods offers an effective and practical method for the analysis of timing behavior.

*Keywords:* formal verification, model checking, region automata, real-time System, satisfiability.

## 1. Introduction

Verification of real-world software systems mandates the ability of handling a large number of system variables. In symbolic model checking, sets of states are encoded as binary decision diagrams (BDD's). System behavior can then be explored by various BDD computation. The technique is known to be useful in many case studies. But the size of BDD may grow significantly as the number of variables increases, systems with a large number of variables are therefore hard to verify by conventional BDD-based model checking algorithms. On the other hand, the memory consumption of Boolean satisfiability (SAT) solvers are less sensitive to the number of variables. SAT-based techniques therefore offer viable alternatives in recent years [7, 10]. A recent comparison [3] of the two techniques suggests that BDD-based algorithms require more space, but SAT-based algorithms take more

*This research is partially supported by NSC project 93-2213-E-001-012-.
†yuf@iis.sinica.edu.tw.
‡Corresponding Author: bywang@iis.sinica.edu.tw.

time. Recent development in SAT solvers [16] also improve the performance of SAT-based techniques.

However, the advantages of SAT-based algorithms are less clear in the analysis of timing behavior which is essential in embedded systems and protocol implementations. One difficulty in applying SAT-based techniques is the modeling of timing behavior. Göllü et al. [11] proposed discretizations of dense time automata and showed that a discretized trajectory traverses the same sequence of regions as its original dense-time trajectory.

Our contribution in this paper are three folds. Firstly, we encode the implicit simulation of region exploration algorithm in Boolean formulas and apply SAT-based bounded model checking techniques to the analysis of timed automata. We not only characterize regions as discrete interpretations, but also precisely encode these interpretations as Boolean formulas. To eliminate discretization side effects such as those mentioned by Göllü et al. [11], we suggest using an exceptional successor formula that prevents discrete distortion in timing behavior. We prove that the satisfiability of these Boolean formulas is equivalent to solving the forward reachability problem of dense-time systems.

Secondly, we incorporate an inductive method in our bounded model checking algorithm. Since bounded model checking is not efficient in proving the correctness of systems, heuristics such as induction have been proposed to circumvent the drawback. The basic idea of inductive method is to prove safety properties for all steps by assuming them in the previous steps. The induction technique has been known in literature [7, 18, 23], but none of them considers timing behavior.

By applying a loop-free inductive method, we enhance our bounded reachability analysis algorithm of region automata with induciton. When the inductive method is effective, it guarantees the given safety property and terminates the algorithm immediately. Compared with other encodings of timing behavior [3, 15, 19, 21, 22], the discretization of region automata allows us to deploy the inductive method rather straightforwardly. Different from conventional model checking algorithms for real-time systems [9, 14, 25], we leverage SAT solver's capability and inductive method's effectiveness in the analysis of timing behavior. Subsequently, we believe that our combined algorithm provides a plausible solution to alleviate state explosion, especially for those systems with many state variables.

Lastly, we implement our new algorithm in a verification tool and report experimental results. Our experimental results suggest that our tool, xBMC, is more scalable for bug hunting than both conventional (Kronos [9], Uppaal [14], RED [25]) and bounded (SAL [18]) model checkers by being able to verify Fischer's protocol up to 22 processes. In another experiment, we verify the correctness of the client authentication protocol, Cornell Single Sign-on Services(CorSSO) [12]. Our results show that xBMC can construct a proof in a handful of inductive steps. Hence, it offers a practical solution to both correctness guarantee and bug hunting in our experiments.

The rest of this paper is organized as follows. In Section 2 we briefly describe timed automata having both discrete and clock variables. In Section 3 we describe

how to encode the behavior of region automata as Boolean formulas. Reachability analysis is given in Section 4, and experimental results are summarized in Section 5. After discussing related works in Section 6, we conclude in Section 7.

## 2. Timed Automata

We consider real-time systems that contain discrete variables and clocks. Discrete variables can be updated by the system. Each clock has a real value and increases at a uniform rate. Initially, each clock is set to zero, and can be reset at any time. Resetting clocks, like update variables, does not induce any cost in the abstraction. Hence, an update or a reset can occur between any two non-simultaneous actions. Such a real-time system can be modelled as a timed automata proposed in [1]. Formally speaking, a timed automata is a tuple of $\langle D, X, A, I, E\rangle$, where:

- $D$ is a finite set of discrete variables. Each $d \in D$ has a predefined finite domain denoted by $\mathbf{dom}\,(d)$;
- $X$ is a finite set of clock variables;
- $A$ is an action set with each $\tau \in A$ consisting of a finite series of assignments to discrete variables;
- $I$ specifies an initial condition; and
- $E \subseteq \Phi\,(D, X) \times A \times 2^X$ is a finite set of edges. An edge $e : \langle \varphi, \tau, \lambda\rangle \in E$ consists of $\varphi \in \Phi\,(D, X)$ a triggering condition which specifies when the transition can be fired, $\tau \in A$ the action that updates the values of discrete variables, and $\lambda \subseteq X$ the set of reset clocks.

For a set $D$ of discrete variables and a set $X$ of clock variables, the set of constraints $\Phi\,(D, X)$ contains constraints $\varphi$ defined by $\varphi := \mathit{ff} \,|\, d = q \,|\, x \triangleleft c \,|\, \neg\varphi \,|\, \varphi_1 \vee \varphi_2$, where $d \in D$ and $q \in \mathbf{dom}\,(d)$, $x \in X$, $\triangleleft \in \{<, =, \leq\}$, and $c \in \mathbf{N}$ is a non-negative integer. We use the following abbreviations: $\mathit{tt} \equiv \neg\mathit{ff}$, $\varphi_1 \wedge \varphi_2 \equiv \neg\left((\neg\varphi_1) \vee (\neg\varphi_2)\right)$ and $\varphi_1 \rightarrow \varphi_2 \equiv \neg\varphi_1 \vee \varphi_2$. A discrete interpretation $s : D \mapsto \mathbf{N}$ assigns each discrete variable a non-negative integer. A clock interpretation $\nu : X \mapsto \mathbf{R}^{\geq \mathbf{0}}$ assigns a non-negative real value to each clock. We say that an interpretation pair $(s,\nu)$ satisfies constraint $\varphi$ if and only if $\varphi$ is true according to values given by $(s,\nu)$.

For any action $\tau$, $s\,[\tau]$ denotes the discrete interpretation after applying $\tau \in A$ to $s$. For $\delta \in \mathbf{R}^{\geq 0}$, $\nu + \delta$ denotes the clock interpretation that maps each clock $x$ to the value $\nu\,(x) + \delta$. For $\lambda \subseteq X$, $\nu\,[\lambda]$ denotes the clock interpretation that assigns 0 to each $x \in \lambda$ and agrees with $\nu$ over the rest of the clocks. The timed automata $\langle D, X, A, I, E\rangle$ defines a transition system $\langle Q, \rightarrow\rangle$, where $Q$ is the set of states and $\rightarrow$ is the transition relation. A *state* $(s, \nu)$ consists of $s$, a discrete interpretation of $D$ and $\nu$, a clock interpretation of $X$. We say $(s, \nu)$ is an *initial* state, where $s$ maps discrete variables to values that satisfy $I$ and $\nu\,(x) = 0$ for all $x \in X$. The transition relation $\rightarrow$ is defined by $\xrightarrow{\delta} \cup \xrightarrow{e}$, where:

- For a state $(s, \nu)$ and a $\delta \in \mathbf{R}^{+}$, $(s, \nu) \xrightarrow{\delta} (s, \nu + \delta)$.
- For a state $(s, \nu)$ and $\exists e : \langle \varphi, \tau, \lambda\rangle \in E$ such that $(s, \nu)$ satisfies $\varphi$, $(s, \nu) \xrightarrow{e} (s\,[\tau]\,, \nu\,[\lambda])$.

A run $r : (s_0, \nu_0) \rightarrow (s_1, \nu_1) \rightarrow \cdots$ of a timed automata is an infinite sequence of states and transitions, where for all $i \in \mathbf{N}, (s_i, \nu_i) \in Q$. An arbitrary interleaving of the two transition types is permissible. A state $(s', \nu')$ is reachable from $(s, \nu)$ if it belongs to a run starting at $(s, \nu)$. Let $Run\,(s, \nu)$ denote the set of runs starting at $(s, \nu)$. We define $Reach\,(s, \nu) : \{(s', \nu')\,|\exists r : (s, \nu) \rightarrow (s_1, \nu_1) \rightarrow \cdots \in Run(s, \nu)$ and $i \in \mathbf{N}, (s_i, \nu_i) = (s', \nu')\}$ as the set of states reachable from $(s, \nu)$.

## 3. Boolean Encoding of Region Automata

### *3.1. Region*

System states change as time progresses, but some changed states are not distinguishable by constraints. Based on this observation, Alur et al. [1, 2] defined the equivalence of clock interpretations and proposed region graphs for the verification of timed automata. To be self-contained, we give the formal definition of *equivalence class* in Definition 1. For each $x \in X$ , let $c_x$ be the largest constant that $x$ is compared to within any triggering condition. For $t \in \mathbf{R}^{\geq 0}$, let $\lfloor t \rfloor$ denote $t$'s integral part, and $frac(t) = t - \lfloor t \rfloor$ denote $t$'s fraction.

**Definition 1** *For clock interpretations $\nu$ and $\nu'$ in a timed automata, we say $\nu \cong \nu'$ if and only if the following conditions hold.*

- *for each $x \in X$, either $\lfloor \nu(x) \rfloor = \lfloor \nu'(x) \rfloor$, or $\nu(x) > c_x$ and $\nu'(x) > c_x$,*
- *for each $x \in X$, $frac(\nu(x)) = 0$ if and only if $frac(\nu'(x)) = 0$,*
- *for each pair $(x_1, x_2)$ such that $\nu(x_1) \leq c_{x_1}$ and $\nu(x_2) \leq c_{x_2}$, $frac(\nu(x_1)) \leq frac(\nu(x_2))$ if and only if $frac(\nu'(x_1)) \leq frac(\nu'(x_2))$.*

It can be shown that $\cong$ defines an equivalent relation over clock interpretations. We use $[\nu]$ to denote the equivalence class that $\nu$ belongs to. Given a clock interpretation $\nu$, we define $\nu_d : X \mapsto \mathbf{N}$, a discrete interpretation of $X$, as follows.

$$\nu_d\,(x) = \begin{cases} 2\,\lfloor \nu(x) \rfloor\,, & if \quad \lfloor \nu(x) \rfloor \leq c_x \wedge frac\,(\nu(x)) = 0 \\ 2\,\lfloor \nu(x) \rfloor + 1, & if \quad \lfloor \nu(x) \rfloor \leq c_x \wedge frac\,(\nu(x)) \neq 0 \\ 2c_x + 1, & otherwise. \end{cases} \tag{1}$$

We also define three predicates: $E$, $O$, $M$ to determine whether the discrete value of a clock is even, odd or its maximum.

**Definition 2** *Given a discrete interpretation $\nu_d$, for some clock $x \in X$:*

- *$E(\nu_d(x))$ is true if and only if $\nu_d(x)$ is even,*
- *$O(\nu_d(x))$ is true if and only if $\nu_d(x)$ is odd and $\nu_d(x) < 2c_x + 1$.*
- *$M(\nu_d(x))$ is true if and only if $\nu_d(x) = 2c_x + 1$.*

To indicate the fraction ordering of $\nu$, we define $\nu_\gamma : X \times X \mapsto \{\prec, \simeq, \succ\}$ as a discrete interpretation over clock pairs.

$$\nu_\gamma\,(x_1, x_2) = \begin{cases} \prec, & if \quad frac\,(\nu\,(x_1)) < frac\,(\nu\,(x_2)) \\ \succ, & if \quad frac\,(\nu\,(x_1)) > frac\,(\nu\,(x_2)) \\ \approx, & if \quad frac\,(\nu\,(x_1)) = frac\,(\nu\,(x_2)) \end{cases} \tag{2}$$

Lemma 1 shows the mapping $[\nu] \mapsto (\nu_d, \nu_\gamma)$ is indeed well-defined.

**Lemma 1** $\nu \cong \nu' \leftrightarrow (\nu_d, \nu_\gamma) = (\nu'_d, \nu'_\gamma)$.

**Proof.** ($\Rightarrow$) Assume $\nu \cong \nu'$. For each $x \in X$, if $\nu(x) > c_x$(implies that $\nu'(x) > c_x$), $\nu_d(x) = \nu'_d(x) = 2c_x + 1$; if $\lfloor \nu(x) \rfloor = \lfloor \nu'(x) \rfloor$, since $frac(\nu(x)) = 0$ iff $frac(\nu'(x)) = 0$, $\nu_d(x) = \nu'_d(x) = 2\lfloor \nu(x) \rfloor (frac(\nu(x)) = 0)$ or $\nu_d(x) = \nu'_d(x) = 2\lfloor \nu(x) \rfloor + 1 (frac(\nu(x)) \neq 0)$. In all cases, $\nu_d(x) = \nu'_d(x)$. To prove $\nu_\gamma = \nu'_\gamma$, note that for any pair $(x_1, x_2) \in \nu_\gamma$,

- $\nu_\gamma(x_1, x_2) = \approx$ iff $frac(\nu(x_1)) \leq frac(\nu(x_2))$ and $frac(\nu(x_2)) \leq frac(\nu(x_1))$,
- $\nu_\gamma(x_1, x_2) = \prec$ iff $frac(\nu(x_1)) \leq frac(\nu(x_2))$ and $\neg(frac(\nu(x_2)) \leq frac(\nu(x_1)))$,
- $\nu_\gamma(x_1, x_2) = \succ$ iff $frac(\nu(x_2)) \leq frac(\nu(x_1))$ and $\neg(frac(\nu(x_1)) \leq frac(\nu(x_2)))$.

Since $frac(\nu(x_1)) \leq frac(\nu(x_2))$ if and only if $frac(\nu'(x_1)) \leq frac(\nu'(x_2))$, it follows that $\nu_\gamma(x_1, x_2) = \nu'_\gamma(x_1, x_2)$.

($\Leftarrow$) Assume $(\nu_d, \nu_\gamma) = (\nu'_d, \nu'_\gamma)$. For each $x \in X$, if $\nu_d(x) = \nu'_d(x) = 2c_x + 1$, both $\nu(x)$ and $\nu'(x)$ are greater than $c_x$; if $\nu_d(x) = \nu'_d(x) < 2c_x + 1, \lfloor \nu(x) \rfloor = \lfloor \nu'(x) \rfloor = \lfloor \frac{\nu_d(x)}{2} \rfloor$. The first condition of Definition 1 holds. Since $\nu_d(x) = \nu'_d(x)$ implies that $E(\nu_d(x))$ iff $E(\nu'_d(x))$, it follows that $frac(\nu(x)) = 0$ iff $frac(\nu'(x)) = 0$. Finally, as we had mentioned, for any pair $(x_1, x_2) \in \nu_\gamma$, each value of $\nu_\gamma(x_1, x_2)$ exactly specifies the fraction relation between $x_1$ and $x_2$. Then, $\nu_\gamma(x_1, x_2) = \nu'_\gamma(x_1, x_2)$ implies that $frac(\nu(x_1)) \leq frac(\nu(x_2))$ iff $frac(\nu'(x_1)) \leq frac(\nu'(x_2))$. □

Lemma 1 shows that $(\nu_d, \nu_\gamma)$ exactly represents $[\nu]$, while $\nu_d$ and $\nu_\gamma$ are defined in (1) and (2) respectively. For example, the equivalence class $1 < x_1 < x_2 < 2 \wedge x_3 = 1$ is represented by the pair $(\nu_d, \nu_\gamma)$, where $\nu_d(x_1) = 3, \nu_d(x_2) = 3, \nu_d(x_3) = 2$, and $\nu_\gamma(x_1, x_2) = \prec$. Accordingly, a *region*, $(s, [\nu])$, can be precisely represented as an *interpretation* state, $(s, \nu_d, \nu_\gamma)$, where three discrete interpretations $s : D \mapsto \mathbf{N}$, $\nu_d : X \mapsto \mathbf{N}$ and $\nu_\gamma : X \times X \mapsto \{\prec, \succ, \approx\}$ are involved.

### *3.2. Successor*

Following the work of Alur et al. [1], we give the definition of *successor*, which captures how system states move from one region into its subsequent region due to time passage.

**Definition 3** *Let $\alpha$, $\beta$ be two distinct regions of a timed automata. $\beta$ is the successor of $\alpha$, written as succ($\alpha$), if and only if for each $(s, \nu) \in \alpha$, there exists a $\delta \in R^+$such that $(s, \nu + \delta) \in \beta$, and $\forall 0 \leq \delta' < \delta, (s, \nu + \delta') \in \alpha \cup \beta$. For an out-of-bound region $\alpha$, i.e., $\forall (s, \nu) \in \alpha, \forall x \in X, \nu(x) > c_x$, its successor relation is defined as succ($\alpha$) = $\alpha$.*

Before answering how to encode successor relation for general interpretation states, we first focus on *2-clock* systems. Figure 3.2 shows all configurations found in a 2-clock system. Respective discrete interpretation conditions of current and next states are shown in Table 1.

Accordingly, we can define a 2-clock formula $\phi_2^{(1,2)}$ for the successor relation of a 2-clock system as follows.

$$\phi_2^{(1,2)} \equiv \bigvee_{i=1..11} \psi_i^{(1,2)} \wedge \psi_i'^{(1,2)} \tag{3}$$

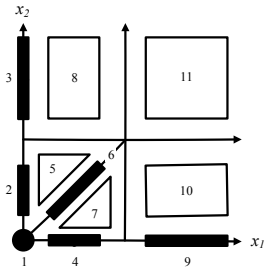


Fig. 1. Region types in a 2-clock system ($X = \{x_1, x_2\}$).

Table 1. Successor conditions in a two-clock system($X = \{x_1, x_2\}$)

| $i$ | current state:$\psi_i^{(1,2)}$ | next state:$\psi_i'^{(1,2)}$ |
|---|---|---|
| 1 | $E(\nu_d(x_1)) \wedge E(\nu_d(x_2))$ | $\nu'_d(x_1) = \nu_d(x_1) + 1 \wedge \nu'_d(x_2) = \nu_d(x_2) + 1$ $\wedge \nu'_\gamma(x_1, x_2) = \approx$ |
| 2 | $E(\nu_d(x_1)) \wedge O(\nu_d(x_2))$ | $\nu'_d(x_1) = \nu_d(x_1) + 1 \wedge \nu'_d(x_2) = \nu_d(x_2)$ $\wedge \nu'_\gamma(x_1, x_2) = \prec$ |
| 3 | $E(\nu_d(x_1)) \wedge M(\nu_d(x_2))$ | $\nu'_d(x_1) = \nu_d(x_1) + 1 \wedge \nu'_d(x_2) = \nu_d(x_2)$ |
| 4 | $O(\nu_d(x_1)) \wedge E(\nu_d(x_2))$ | $\nu'_d(x_1) = \nu_d(x_1) \wedge \nu'_d(x_2) = \nu_d(x_2) + 1$ $\wedge \nu'_\gamma(x_1, x_2) = \succ$ |
| 5 | $O(\nu_d(x_1)) \wedge O(\nu_d(x_2)) \wedge \nu_\gamma(x_1, x_2) = \prec$ | $\nu'_d(x_1) = \nu_d(x_1) \wedge \nu'_d(x_2) = \nu_d(x_2) + 1$ |
| 6 | $O(\nu_d(x_1)) \wedge O(\nu_d(x_2)) \wedge \nu_\gamma(x_1, x_2) = \approx$ | $\nu'_d(x_1) = \nu_d(x_1) + 1 \wedge \nu'_d(x_2) = \nu_d(x_2) + 1$ |
| 7 | $O(\nu_d(x_1)) \wedge O(\nu_d(x_2)) \wedge \nu_\gamma(x_1, x_2) = \succ$ | $\nu'_d(x_1) = \nu_d(x_1) + 1 \wedge \nu'_d(x_2) = \nu_d(x_2)$ |
| 8 | $O(\nu_d(x_1)) \wedge M(\nu_d(x_2))$ | $\nu'_d(x_1) = \nu_d(x_1) + 1 \wedge \nu'_d(x_2) = \nu_d(x_2)$ |
| 9 | $M(\nu_d(x_1)) \wedge E(\nu_d(x_2))$ | $\nu'_d(x_1) = \nu_d(x_1) \wedge \nu'_d(x_2) = \nu_d(x_2) + 1$ |
| 10 | $M(\nu_d(x_1)) \wedge O(\nu_d(x_2))$ | $\nu'_d(x_1) = \nu_d(x_1) \wedge \nu'_d(x_2) = \nu_d(x_2) + 1$ |
| 11 | $M(\nu_d(x_1)) \wedge M(\nu_d(x_2))$ | $\nu'_d(x_1) = \nu_d(x_1) \wedge \nu'_d(x_2) = \nu_d(x_2)$ |

Let $\nu_d, \nu_\gamma \models \phi$ denote that $\phi$ is true when variables in $\phi$ are assigned to values given by $\nu_d, \nu_\gamma$. Lemma 3 shows the correctness of the discretization.

**Lemma 2** *Given a 2-clock timed automata with $X = \{x_1, x_2\}$ and a region $\alpha$ represented by $(s, \nu_d, \nu_\gamma)$ such that $\nu_d, \nu_\gamma \models \psi_i^{(1,2)}$ (for some $i = 1 \ldots 11$), succ($\alpha$) is represented by $(s, \nu'_d, \nu'_\gamma)$ if and only if $\nu_d, \nu_\gamma, \nu'_d, \nu'_\gamma \models \psi_i'^{(1,2)}$.*

**Proof.** The proof is straightforward since in a 2-clock system, the successor of a region is exactly the first region it encounters in the northeast direction. □

**Lemma 3** *Let $\alpha, \beta$, represented by $(s, \nu_d, \nu_\gamma)$ and $(s', \nu'_d, \nu'_\gamma)$ respectively, be two regions of a 2-clock timed automata with $X = \{x_1, x_2\}$, $\beta = succ(\alpha)$ if and only if $s = s'$ and $\nu_d, \nu_\gamma, \nu'_d, \nu'_\gamma \models \phi_2^{(1,2)}$.*

**Proof.** The correctness is shown by the following observations of Table 1.

- All cases are considered, i.e., $\bigvee_{i=1\ldots11} \psi_i^{(1,2)} = 1$.
- Each case presents a unique type, i.e., $\bigvee_{i \neq j} \psi_i^{(1,2)} \wedge \psi_j^{(1,2)} = 0$.
- According to Lemma 2, the condition of the current equivalence class and its successor in the $i^{th}$ case is exactly specified by $\psi_i^{(1,2)} \wedge \psi_i'^{(1,2)}$.

□

We derive a general formula for *n-clock* systems by intersecting 2-clock formulas, instead of inspecting all clock values in one time. Our initial attempt detailed intersecting $\phi_2$ of each distinct clock pair, i.e., $\bigwedge_{x_i, x_j \in X, i<j} \phi_2^{(i,j)}$. Apparently such an intuitive conjunction easily raises contradictions. Come back to the previous

example, where a region $1 < x_1 < x_2 < 2 \wedge x_3 = 1$ is represented by $\nu_d(x_1) = 3 \wedge \nu_d(x_2) = 3 \wedge \nu_d(x_3) = 2 \wedge \nu_\gamma(x_1, x_2) = \prec$. The conjunction of 2-clock formulas, i.e. $\phi_2^{(1,2)} \wedge \phi_2^{(1,3)} \wedge \phi_2^{(2,3)}$, implies

$$\begin{array}{ll} & (\nu'_d(x_1) = 3 \wedge \nu'_d(x_2) = 4) & //(x_1, x_2) \\ \wedge & (\nu'_d(x_1) = 3 \wedge \nu'_d(x_3) = 3 \wedge \nu'_\gamma(x_1, x_3) = \succ) & //(x_1, x_3) \\ \wedge & (\nu'_d(x_2) = 3 \wedge \nu'_d(x_3) = 3 \wedge \nu'_\gamma(x_2, x_3) = \succ). & //(x_2, x_3) \end{array}$$

Obviously, $\nu'_d(x_2) = 4 \wedge \nu'_d(x_2) = 3$ makes the predicate evaluated as false. This is because that we require $x_2$ to increase when compared to $x_1$ but require it to remain the same value(stutter) when compared to $x_3$. We call clocks that raise contradictions *contradictory clocks*. Two important observations help us prevent these contradictions.

- All contradictory clocks belong to the set:$\{x|O(\nu_d(x))\}$.
- Contradictions should be solved by either enforcing clocks in $\{x|E(\nu_d(x))\}$ to increase, or enforcing clocks having the largest fraction part to increase if no clocks have even values.

Instead of enforcing clocks increasing, our solution is adding an auxiliary case to allow contradictory clocks stuttering.

Table 2. Stuttering conditions for $x_i$ and $x_j$

| | current state($\psi_s$) | next state($\psi'_s$) |
|---|---|---|
| $s$ | $O(\nu_d(x_i)) \wedge O(\nu_d(x_j))$ | $\nu'_d(x_i) = \nu_d(x_i) \wedge \nu'_d(x_j) = \nu_d(x_j) \wedge \nu'_\gamma(x_i, x_j) = \nu_\gamma(x_i, x_j)$ |

By disjoining stuttering conditions defined in Table 2, we define *(i,j)-clock formula* for n-clock systems as follows.

$$\phi_n^{(i,j)} \equiv \phi_2^{(i,j)} \vee (\psi_s^{(i,j)} \wedge \psi_s'^{(i,j)}) \tag{4}$$

Recall the previous example, the conjunction of all (i,j)-clock formulas implies

$$\begin{array}{ll} & \big((\nu'_d(x_1) = 3 \wedge \nu'_d(x_2) = 4) \vee (\nu'_d(x_1) = 3 \wedge \nu'_d(x_2) = 3 \wedge \nu'_\gamma(x_1, x_2) = \prec)\big) \\ \wedge & (\nu'_d(x_1) = 3 \wedge \nu'_d(x_3) = 3 \wedge \nu'_\gamma(x_1, x_3) = \succ) \\ \wedge & (\nu'_d(x_2) = 3 \wedge \nu'_d(x_3) = 3 \wedge \nu'_\gamma(x_2, x_3) = \succ). \end{array}$$

The predicate is equal to $\nu'_d(x_1) = 3 \wedge \nu'_d(x_2) = 3 \wedge \nu'_d(x_3) = 3 \wedge \nu'_\gamma(x_1, x_2) = \prec \wedge \nu'_\gamma(x_1, x_3) = \succ \wedge \nu'_\gamma(x_2, x_3) = \succ$. This is the interpretation state of $1 < x_3 < x_1 < x_2 < 2$, which is exact the successor of $1 < x_1 < x_2 < 2 \wedge x_3 = 1$.

Adding auxiliary stuttering cases helps us prevent contradictions, but it may induce distort behavior such as all clocks stutter. This may happen only when $\forall x \in X, O(\nu_d(x))$ holds. We prevent all pairs stuttering by adding the formula $\phi_{nzeno}$ as follows.

$$\phi_{nzeno} \equiv \neg \left( \bigwedge_{1 \le i < j \le n} \psi_s^{(i,j)} \wedge \psi_s'^{(i,j)} \right) \tag{5}$$

Finally, the general successor formula for n-clock systems is given in Equation 6.

$$\phi_n \equiv \left( \bigwedge_{1 \le i < j \le n} \phi_n^{(i,j)} \right) \wedge \phi_{nzeno} \tag{6}$$

**Lemma 4** *Let $\alpha, \beta$, represented by $(s, \nu_d, \nu_\gamma)$ and $(s', \nu'_d, \nu'_\gamma)$ respectively, be two regions of an n-clock timed automata, $\beta = succ(\alpha)$ if and only if $\nu_d, \nu_\gamma, \nu'_d, \nu'_\gamma \models \phi_n$.*

**Proof.** ($\Rightarrow$) It is easy to see that $\beta = succ(\alpha)$ implies that, for each pair of clocks $(x_i, x_j)$, $\phi_n^{(i,j)}$ is evaluated to true (according to Lemma 3), and since $\alpha, \beta$ are distinct (according to Definition 3), not all stuttering cases are allowed, i.e. $\nu_d, \nu_\gamma, \nu'_d, \nu'_\gamma \models \phi_{nzeno}$.

($\Leftarrow$) Assume $\nu_d, \nu_\gamma, \nu'_d, \nu'_\gamma \models \phi_n$. We prove that all cases satisfy Definition 3. Let $\chi = \{x | x \in X, \nu'_d(x) = \nu_d(x) + 1.\}$.

(i) $\forall x \in X, M(\nu_d(x))$: In this case, $(s, \nu_d, \nu_\gamma)$ is an out-of-bound region. For $x_i, x_j \in X$, $\nu_d, \nu_\gamma \models \psi_{11}^{(i,j)}$, which follows that $\nu_d = \nu'_d$, i.e., $\alpha = \beta = succ(\alpha)$.

(ii) $\exists x \in X, E(\nu_d(x))$: We assert that $\chi = \{x | x \in X, E(\nu_d(x))\}$. It can be seen as follows. We first prove that $\forall x \in \chi, E(\nu_d(x))$. Assume that there exists $x_i \in \chi$ and $O(\nu_d(x_i))$. Then for any $x_j \in \{x | x \in X, E(\nu_d(x))\}$, $\psi_4^{(i,j)} \wedge \psi_4'^{(i,j)}$ is violated. This raises a contradiction. We then prove that $\forall x \in X$ and $E(\nu_d(x))$, $x \in \chi$. Assume that there exists $x_i \in \{x | x \in X, E(\nu_d(x))\}$ but $x_i \notin \chi$. Then we can find some $x_j \in X$, such that a) if $E(x_j)$, $\psi_1^{(i,j)} \wedge \psi_1'^{(i,j)}$ is violated, b)if $O(x_j)$, $\psi_2^{(i,j)} \wedge \psi_2'^{(i,j)}$ is violated, and c)if $M(x_j)$, $\psi_3^{(i,j)} \wedge \psi_3'^{(i,j)}$ is violated. In all cases, the assumption raises a contradiction. Since only clocks in $\{x | x \in X, E(\nu_d(x))\}$ are progressed, for any $\nu \in \alpha$ and $\nu' \in \beta$, we can choose a $\delta$ such that $0 < \delta < 1 - \max_{x \in X} frac(\nu(x))$, and $\forall 0 \le \delta' < \delta$, $\nu + \delta \in (\nu'_d, \nu'_\gamma)$ and $\nu + \delta' \in (\nu_d, \nu_\gamma) \cup (\nu'_d, \nu'_\gamma)$.

(iii) $\exists x \in X, O(\nu_d(x))$ and $\forall x \in X, \neg E(\nu_d(x))$: We assert that $\forall (x_i, x_j) \in \nu_\gamma$, (1)$\nu_\gamma(x_i, x_j) = \approx$, if $x_i, x_j \in \chi$, (2)$\nu_\gamma(x_i, x_j) = \prec$, if $x_i \notin \chi$ and $x_j \in \chi$, and (3) $\nu_\gamma(x_i, x_j) = \succ$, if $x_i \in \chi$ and $x_j \notin \chi$. In other words, we assert that $\chi = \{x | x \in X, O(\nu_d(x)),$ and $x$ has the largest fraction$\}$. Again, we first prove that $\forall x \in \chi$, $x$ has the largest fraction. Assume that there exists $x_i \in \chi$, and $x_j \in \{x | x \in X, O(\nu_d(x))\}$ such that $x_j$ has a larger fraction than $x_i$ (i.e., $\nu_\gamma(x_i, x_j) = \prec$). It's easy to see that $\psi_5^{(i,j)} \wedge \psi_5'^{(i,j)}$ is violated and a contradiction arises. We then prove that all clocks having the largest fraction are in $\chi$. Assume that there exists $x_i \in \{x | x \in X, O(\nu_d(x))\}$, such that $x_i$ has the largest fraction but $x_i \notin \chi$. Then we can find some $x_j \in X$, such that a) if $O(x_j)$ and $\nu_\gamma = \approx$, $\psi_6^{(i,j)} \wedge \psi_6'^{(i,j)}$ is violated, b)if $O(x_j)$ and $\nu_\gamma = \succ$, $\psi_7^{(i,j)} \wedge \psi_7'^{(i,j)}$ is violated and c)if $M(x_j)$, $\psi_8^{(i,j)} \wedge \psi_8'^{(i,j)}$ is violated. Since $\forall x \in X, \neg E(\nu_d(x))$, in all cases, the assumption raises a contradiction. Finally, since all and only clocks having the largest fraction are progressed, for any $\nu \in \alpha$ and $\nu' \in \beta$, we can choose a $\delta$ such that $\delta = 1 - frac(\nu(x)), x \in \chi$, and for all $0 \le \delta' < \delta$, $\nu + \delta \in (\nu'_d, \nu'_\gamma)$ and $\nu + \delta' \in (\nu_d, \nu_\gamma) \cup (\nu'_d, \nu'_\gamma)$.

□

*3.3. Discrete Transition*

In this sub-section, we describe the conditions of interpretation states to trigger an edge. Since we use discrete intervals to represent clock values, the first step is to transform the triggering condition $\varphi$ into $\varphi_d$ by replacing $x \lhd c$ with $\nu_d(x) \lhd 2c$, for all $x \in X$. Actions of transitions include a) applying an assignment sequence $\tau$, and b) resetting a set of clocks $\lambda$. Let $\nu_d[\lambda]$ denote the discrete interpretation that a) assigns 0 to each $x \in \lambda$ and b) agrees with $\nu_d$ over the rest of the clocks. And let $\nu_\gamma[\lambda]$ denote the discrete interpretation that a) agrees with $\nu_\gamma$ over clocks in $\{x|O(\nu_d[\lambda](x))\}$, and b) discards other pairs. Given an edge $e : \langle \varphi, \tau, \lambda \rangle$, we can define $\psi^{(e)}$, the conditions of interpretation states to trigger $e$, as Equation 7. And then, given a timed automata, we define its discrete transition formula $\phi_{tran}$ as Equation 8.

$$\psi^{(e)} \equiv \varphi_d \wedge s' = s[\tau] \wedge \nu'_d = \nu_d[\lambda] \wedge \nu'_\gamma = \nu_\gamma[\lambda] \tag{7}$$

$$\phi_{tran} \equiv \bigvee_{e \in E} \psi^{(e)} \tag{8}$$

**Lemma 5** $\nu \models \varphi \leftrightarrow \nu_d \models \varphi_d$.

**Proof.** The correctness comes from that $\forall x \in \{x|x \in X, x \lhd c \in \varphi\}$, $\nu(x) \lhd c \leftrightarrow \nu_d(x) \lhd 2c$. ($\Rightarrow$). Since $c \in \mathbf{N}$, this can be seen that a) if $\lhd$ is $=$, $\nu(x) = c$ implies that $\nu_d(x) = 2\nu(x) = 2c$, b) if $\lhd$ is $<$, $\nu(x) < c$ implies that $\lfloor \nu(x) \rfloor \leq c - 1$. Then either $\nu_d(x) = 2\lfloor \nu(x) \rfloor \leq 2c - 2 < 2c$ or $\nu_d(x) = 2\lfloor \nu(x) \rfloor + 1 \leq 2c - 1 < 2c$, and c) if $\lhd$ is $\leq$, this can be proved via a) and b). ($\Leftarrow$) The proof of is similar. □

**Lemma 6** *Let $\alpha, \beta$, represented by $(s, \nu_d, \nu_\gamma)$ and $(s', \nu'_d, \nu'_\gamma)$ respectively, be two regions of an n-clock timed automata, for any state $(s, \nu) \in \alpha$ and $(s', \nu') \in \beta$, $(s, \nu) \xrightarrow{e} (s', \nu')$ if and only if $s, \nu_d, \nu_\gamma, s', \nu'_d, \nu'_\gamma \models \psi_{tran}$.*

**Proof.** It follows Lemma 5 and the correctness of discrete encodings. □

*3.4. Interpretation Graph*

The transition system of a timed automata is represented by a finite discrete interpretation graph $\left\langle Q_\cong, \xrightarrow{\cong} \right\rangle$, where $Q_\cong = \{(s, \nu_d, \nu_\gamma) | \exists (s, \nu) \in Q, (s, \nu) \in (s, \nu_d, \nu_\gamma)\}$ and $\xrightarrow{\cong}$ is defined by $\xrightarrow{\delta_\cong} \cup \xrightarrow{e_\cong}$.

- Given a state $(s, \nu_d, \nu_\gamma)$, $(s, \nu_d, \nu_\gamma) \xrightarrow{\delta_\cong} \left(s', \nu'_d, \nu'_\gamma\right)$ if and only if $s, \nu_d, \nu_\gamma, s', \nu'_d, \nu'_\gamma \models \phi_{time}$.
- Given a state $(s, \nu_d, \nu_\gamma)$, $(s, \nu_d, \nu_\gamma) \xrightarrow{e_\cong} \left(s', \nu'_d, \nu'_\gamma\right)$ if and only if $s, \nu_d, \nu_\gamma, s', \nu'_d, \nu'_\gamma \models \phi_{tran}$.

$$\phi_{time} \equiv s = s' \wedge \phi_n \tag{9}$$

$\phi_{time}$ (Equation 9) defines the successor relation formula for capturing a region moving into a subsequent region due to time passage, while $\phi_{tran}$ defines the discrete transition formula for triggering some edge using discrete interpretations.

Note that $(s, \nu_d, \nu_\gamma)$ is reachable from $(s', \nu'_d, \nu'_\gamma)$ in one step, i.e., $(s, \nu_d, \nu_\gamma) \xrightarrow{\delta\cong} (s', \nu'_d, \nu'_\gamma)$, only when $s, \nu_d, \nu_\gamma, s', \nu'_d, \nu'_\gamma \models \phi_{time} \vee \phi_{tran}$. Generally, we define that $(s, \nu_d, \nu_\gamma)$ is reachable from $(s', \nu'_d, \nu'_\gamma)$ by exactly using k steps (k>0) in Definition 4, and a set of reachable states within k steps can then be defined as Definition 5.

**Definition 4** $(s, \nu_d, \nu_\gamma) \xrightarrow{\delta^k_\cong} (s', \nu'_d, \nu'_\gamma)$ *if and only if,* $\forall 0 \leq i < k$, *there exists* $(s^i, \nu^i_d, \nu^i_\gamma)$, *such that a)* $(s, \nu_d, \nu_\gamma) = (s^0, \nu^0_d, \nu^0_\gamma)$, *b)*$(s', \nu'_d, \nu'_\gamma) = (s^k, \nu^k_d, \nu^k_\gamma)$, *and c)* $s^i, \nu^i_d, \nu^i_\gamma, s^{i+1}, \nu^{i+1}_d, \nu^{i+1}_\gamma \models \phi_{time} \vee \phi_{tran}$.

**Definition 5** $Reach((s, \nu_d, \nu_\gamma), k) = \{(s, \nu_d, \nu_\gamma)\} \cup \{(s', \nu'_d, \nu'_\gamma) | \exists i, 0 < i \leq k, (s, \nu_d, \nu_\gamma) \xrightarrow{\delta^i_\cong} (s', \nu'_d, \nu'_\gamma)\}$

**Lemma 7** *Given a timed automata and two states* $(s, \nu)$ *and* $(s', \nu')$, $(s', \nu') \in Reach(s, \nu)$ *if and only if* $\exists k \in \boldsymbol{N}$, $(s, \nu) \in (s, \nu_d, \nu_\gamma)$ *and* $(s', \nu') \in Reach((s, \nu_d, \nu_\gamma), k)$.

**Proof.** Since regions are finite, there exists an intrinsic bound, i.e. the number of regions, to explore all reachable regions. □

Lemma 7 shows that it is sound and complete to use interpretation graph doing reachability analysis of real-time systems. We propose three algorithms to achieve this in Section 4.

### *3.5. Boolean Encoding*

One set of our state variables $B$ is defined in Equation 10, in which a set of Boolean variables is used to encode interpretation states. Given each discrete variable's domain and each clock's largest constraint value, the number of state variables, i.e. $|B|$, equals $\sum \lceil \lg |\mathbf{dom}\,(d)| \rceil + \sum \lceil \lg (2c_x + 2) \rceil + |X|\,|X - 1|$. We than adopt standard Boolean encoding method. For example, given a timed automata with one discrete variable $d : \{D_1, D_2\}$ and two clocks $x_1 : c_{x_1} = 1$, $x_2 : c_{x_2} = 1$, a region $d = D_1 \wedge 0 < x_1 < x_2 < 1$ would be encoded as $\overline{b^d_0} \wedge \overline{b^1_1} \wedge b^1_0 \wedge \overline{b^2_1} \wedge b^2_0 \wedge \overline{b^{12}_1} \wedge b^{12}_0$.

$$B = \begin{array}{ll} \left\{ b^d_k | d \in D, 0 \leq k < \lceil \lg |\mathbf{dom}\,(d)| \rceil \right\} & \cup \\ \left\{ b^i_k | x_i \in X, 0 \leq k < \lceil \lg (2c_x + 2) \rceil \right\} & \cup \\ \left\{ b^{ij}_k | x_i, x_j \in X, i < j, k \in \{0, 1\} \right\} & \end{array} \tag{10}$$

To perform BMC, we add a copy of $B$ to the set of state variables at each iteration. Finally, we build a circuit representation to translate a bit-vector logic (used to build the equation for a concrete transition relation) into conjunctive normal form (CNF).

## 4. Reachability Analysis

In this section, we describe how we deal with the reachability analysis by solving the satisfiability of Boolean formulas. We first propose a bounded algorithm, and then use loop-free constraints to release bounds. In the last of this section, we briefly induce inductive methods and blend *windowed induction* [23] in our algorithm.

```
BoundedFwdReach(A, R, k)
   var i:0..k;
begin
   i:=0; F:=I(B_0);
   loop forever
      if (i=k) return "unreachable within k steps";
      if (SAT(F∧R(B_i))) return"reachable";
      F:=F∧¬R(B_i)∧(B_i → B_{i+1});
      i:=i+1;
end.
```

Figure 2: Bounded Forward Reachability Analysis. I($B_0$) indicates the circuit representation of $A$'s initial states, R($B_i$) indicates the circuit representation of risk states over $B_i$, and ($B_i \rightarrow B_{i+1}$) indicates the circuit representation of $\phi_{time} \vee \phi_{tran}$ over $B_i$ and $B_{i+1}$.

### *4.1. Bounded Reachability Analysis*

In Section 3.4, we have shown the correctness of using bounded interpretation graph to do reachability analysis of real-time systems. A brief bounded algorithm is given in Figure 2. Given a timed automata $A$, a set of risk states $R$, and an integer bound ($k : k \in \mathbf{N}$), we determine whether $R$ is reached within $k$ steps. This can be done by solving the unfolding formulas iteratively until either the SAT solver returns a truth assignment, or the procedure had repeated $k$ times. By conjoining the formula with the negation of $R(B_{i-1})$ at the $i^{th}$ step, each intermediate result is saved for use in later iterations, and as a result, speeding up the decision procedure of the SAT-solver. Moreover, in Section 4.3, we will show that this extra constraint also helps us construct inductive proofs.

If some state in $R$ is reachable at the $i^{th}$ step, the formula will be satisfied, and a truth assignment over $\bigcup B_i$ will be returned. We then generate a counterexample by interpreting state variables' values. On the other hand, the formula will keep on expanding while no states in $R$ are reached. In that case, the procedure terminates while either having repeated $k$ times, or exhausting system resources.

**Theorem 1** *BoundedFwdReach(A, R, k) is sound and complete when k is not less than the number of regions in A.*

Given a timed automata having $n$ regions, we require $k \geq n$ in Theorem 1. However, since $n$ is exponential to both a) the number of clocks and b) each clock's largest constraint constant, the threshold is usually prohibitively expensive.

### *4.2. Loop-free Reachability Analysis*

Choosing the number of regions as the reachability diameter reflects the worst case. A better option might be the steps of the longest shortest path. However, to calculate it in advance is usually infeasible. To conquer this hurdle, we adopt loop-free termination. A loop-free algorithm is proposed in Figure 3. By inserting loop-free constrains, i.e. $\bigwedge_{j<i} B_i \neq B_j$ at the $j^{th}$ iteration, we enforce SAT-solvers

*LoopFreeReach*(A, R)
**begin**
  i:=0; F:=I($B_0$);
  **loop forever**
    if (SAT(F∧R($B_i$))) return"reachable";
    F:=F∧¬R($B_i$)∧($B_i \rightarrow B_{i+1}$)∧($\bigwedge_{j<i} B_i \neq B_j$);
    if (not SAT(F) return"unreachable by loop-free";
    i:=i+1;
**end.**

Figure 3: Loopfree Reachability Analysis.

searching distinct states. Once there is no solution, i.e., no distinct state found in the next step, the procedure terminates and $R$ is unreachable. Since all states are distinct, a loop-free path is exactly a shortest path, and hence, if we concern complete verification, there is no need to determine the bound ahead.

The loop-free restrictions help us find a shortest path once a bug exists. Once no bugs exist, the procedure will cover the steps of the longest shortest path. This algorithm proves correctness with considering the diameter of reachability graph.

### *4.3. Inductive Reachability Analysis*

Although SAT-based model checking is very useful for bug hunting [6, 10, 28], its ability to prove properties is often criticized. The inductive method offers SAT-based model checking an opportunity to prove safety properties efficiently. The basic idea, like mathematical induction, is to construct induction proof of the property for all steps by assuming the property on previous steps.

#### 4.3.1. Induction

Here, we briefly illustrate the technique. Interested readers are referred to [7, 18, 23] for further discussion. Let $q_0 \in Q$ where $q_0$ is the initial state and $P(\bullet)$ a predicate over states in $Q$. We would like to prove that for any state $q$ reachable from $q_0$, $P(q)$ holds by induction. Firstly, we verify if $P(q_0)$ holds. If not, an error is reported for $q_0$. If $P(q_0)$ holds, we check whether $P(q) \wedge (q \rightarrow q') \wedge \neg P(q')$ can be satisfied, i.e. whether it is possible to reach a state $q'$ that does not satisfy $P(\bullet)$ from any state $q$ satisfying $P(\bullet)$. If it is impossible, we know that $P(\bullet)$ must hold for any state in $Reach(q_0)$. To see this, recall that $P(q_0)$ holds. We argue that all successors of $q_0$ must satisfy $P(\bullet)$. If not, $P(q) \wedge (q \rightarrow q') \wedge \neg P(q')$ must be satisfiable for some successor $q'$, which is a contradiction. Similarly, we can prove all states reachable from $q_0$ in two steps must satisfy $P(\bullet)$, and so on. Hence we conclude all reachable states must satisfy $P(\bullet)$. In the example, the depth of the inductive step is one. We call it *simple induction*. However, the capability of simple induction is very limited. Just like mathematical induction, it may be necessary to assume several previous steps in order to prove the property. This is regarded as *windowed induction*, i.e., induction with arbitrary depths. Unfortunately, the

**IndFwdReach**(A, R)
begin
  i:=0; F:=I($B_0$);
  loop forever
    if (not SAT(F\I∧R($B_i$)) return"unreachable by induction";
    if (SAT(F∧R($B_i$))) return"reachable";
    F:=F∧¬R($B_i$)∧($B_i \rightarrow B_{i+1}$)∧($\bigwedge_{j<i} B_i \neq B_j$);
    if (not SAT(F) return"unreachable by loop-free";
    i:=i+1;
end.

Figure 4: Inductive Forward Reachability Analysis

inductive technique cannot prove all safety properties, even with arbitrary depths. In [7, 18, 23], various mechanisms are proposed to make induction complete. Here, we use loop-free induction. In loop-free induction, additional constraints are applied to prevent loops. Consider a self-loop transition, followed by a state that does not satisfy $P(\bullet)$. The problematic state can always be reached by an inductive step of arbitrary depth. It suffices to consider a short path leading to the problematic state and still prove the soundness and completeness of the induction. By requiring all previous states to be distinct, loop-free induction eliminates unsubstantial paths. Based on the discretization scheme in Section 3, we can deploy loop-free induction to speed up the verification of safety properties.

4.3.2. Inductive Reachability Analysis

Since the number of regions is exponential to the number of clocks, as well as each clock's largest constant, the threshold is usually prohibitively high. In *IndFwdReach()*, we combine loop-free induction with *BoundFwdReach()* and obtain a complete inductive algorithm for forward reachability analysis. Note that, in Figure 4, we denote the released formula as $F \setminus I$, i.e. removing the clauses of $I$ from $F$. An extra check for whether an induction proof is constructed is used to determine completeness in early steps. We regard $P(\bullet)$ as $\neg R(\bullet)$, i.e. the negation of the risk condition. Once the induction succeeds (a proof is constructed), we can conclude that all reachable states must satisfy $\neg R(\bullet)$, which implies that the risk state is unreachable.

One limitation of this inductive method is that we can not predict in which step an induction proof is constructed successfully. Although regions are finite and we can guarantee "success" when the property is satisfied, in the worst case, the inductive method may not determine termination ahead. However, when the inductive method is effective, it can verify the given safety property within a handful of steps, regardless of the diameter of the reachability graph. In Section 5.2, we conduct an experiment to show the effectiveness of induction.

## 5. Experiment

We have implemented our algorithms in a prototype xBMC. The SAT solver used in the prototype is zChaff [7]. In this section, we compare the performance of xBMC with other formal analysis tools in two case studies: Fischer's mutual exclusion protocol and Cornell Single Sign-On Services (CorSSO) [12].

The model checking tools used in the following experiments are Kronos [9], RED [25], Uppaal [14] and SAL 2 (infBMC) [19]. Uppaal provides a friendly interface and is able to verify safety and bounded liveness properties of real-time systems. Kronos and RED 4.0, additionally, support full TCTL verification. Among these three tools, they essentially use the conventional symbolic model checking algorithm with different symbolic representations. Kronos and Uppaal use Difference Bounded Matrices (DBM's) [13] to represent system configurations. RED, on the other hand, uses Clock Restriction Diagrms (CRD's) [25].

In contrast to the aforementioned conventional model checkers, infBMC is a bounded model checker included in SAL 2, a suit of tools developed by the SRI's Symbolic Analysis Laboratory for analyzing state machines. InfBMC supports verification of infinite state systems using a special decision procedure [17, 18] that solves the satisfiability of combinations of linear real and integer arithmetics.

### *5.1. Fisher's Mutual Exclusion*

We first verify Fischer's mutual exclusion protocol. In our model, there are $n$ timed automata in the protocol. Each timed automata modeling a process (Figure 5.1). Mutual exclusion property is known to be unsatisfied when $A < B$. The largest constraint for the local clock in each process is adjusted by increasing the value of $B$ and keeping $A = 1$.

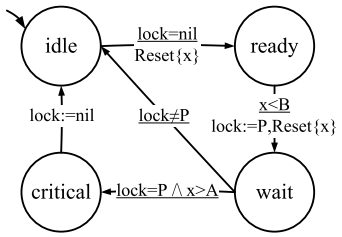


Fig. 5. A Process of Fischer's Mutual Exclusion Protocol.

When $c_x = 2$ (Table 3), Kronos fails to construct the product automata of the system while verifying 6 processes. InfBMC reports all counterexamples in 10 iterations, but its internal decision procedure crashed while verifying 6 processes. Uppaal runs efficiently until the number of processes reached 14. RED has the best capacity among them. It can check up to 15 processes. XBMC is capable of reporting all counterexamples in 14 iterations and verifying up to 22 processes. In Table 4, xBMC can only handle up to 13 processes while performance among the other tools is not significantly affected by increasing constraint constants. The result reveals a deficiency of our method, the sensitivity of constraint constants.

Table 3. Detecting mutual exclusion violations of Fischer's protocol with $c_x = 2$. "O/M" indicates that the model checker ran out of memory.

| #p | Kronos 2.5.2 | Uppaal 3.5.2 | RED 5.0 | SAL 2.1(infBMC) | xBMC |
|---|---|---|---|---|---|
| 4 | 0.12s | 0.03s | 0.57s | 86.98s | 3.28s |
| 5 | 0.52s | 0.03s | 1.95s | 420.98s | 10.49s |
| 6 | O/M | 0.06s | 5.70s | O/M | 14.66s |
| 7 | | 0.16s | 14.47s | | 16.83s |
| 9 | | 1.17s | 75.5s | | 46.90s |
| 11 | | 5.08s | 321.04s | | 129.46s |
| 13 | | 12.21s | 1129.18s | | 111.59s |
| 14 | | O/M | 2005.23s | | 237.89s |
| 15 | | | 4234.41s | | 531.73s |
| 16 | | | O/M | | 453.83s |
| 17 | | | | | 414.29s |
| 19 | | | | | 528.66s |
| 21 | | | | | 641.27s |
| 22 | | | | | 587.01s |
| 23 | | | | | O/M |

Table 4. Detecting mutual exclusion violations of Fischer's protocol with $c_x = 4000$.

| #p | Kronos 2.5.2 | Uppaal 3.5.2 | RED 5.0 | SAL 2.1(infBMC) | xBMC |
|---|---|---|---|---|---|
| 4 | 0.11s | 0.02s | 0.56s | 95.45s | 20.31s |
| 5 | O/M | 0.04s | 1.95s | 275.82s | 37.32s |
| 6 | | 0.06s | 4.82s | O/M | 47.63s |
| 7 | | 0.17s | 12.90s | | 47.04s |
| 9 | | 1.21s | 74.31s | | 91.35s |
| 11 | | 9.35s | 353.61s | | 200.84s |
| 13 | | O/M | 1345.08s | | 447.39s |
| 14 | | | 2471.07s | | O/M |
| 15 | | | 4238.34s | | |
| 16 | | | O/M | | |

### *5.2. Cornell Single Sign-On Service*

Cornell Single Sign-On (CorSSO) [12] is a distributed service for network authentication. It delegates client identity checking to multiple servers by threshold cryptography. In CorSSO, there are three kinds of principles, namely, authentication servers, application servers and clients. For a client to access the services provided by an application server, it has to identify itself by the authentication policy specified by the application server. The authentication policy consists of a list of sub-policies each specifying a set of authentication servers. A client is allowed to access the application server if it has complied with any sub-policy by obtaining sufficient certificates from the specified authentication servers in time.

Unlike monolithic authentication schemes where the server is usually overloaded with authentication requests, the authentication policies in CorSSO allow a client to prove its identity by different criteria. With threshold cryptography, each criterion is divided into requests to multiple authentication servers. The authentication process is therefore distributed, so the load of each authentication server is more balanced.

In the experiments, we model client behavior. In the simplified client model shown in Figure 5.2, a client has only two locations: *auth*(Authentication Mode) and *access*(Access Mode). In *auth*, it firstly chooses one of the two policies by setting the value of $p$ non-deterministically. If the $i^{th}$ policy is chosen($p = i$), it needs to collect more than $\texttt{TH}_i$ certificates from the authentication servers. If it then obtains sufficient certificates in time, it can move to *access*.

To model timing constraints, we use two clock variables $x$ and $y$. Let us suppose that it spends at least `TA` to acquire a certificate. Then one new certificate can be added until $x$ exceeds `TA`, and once it collected, $x$ is reset for the next certificate. Furthermore, all certificates for a sub-policy must be collected within `TE`, which is modeled by $y$. Note that $y$ is reset each time the client choosing a new sub-policy.

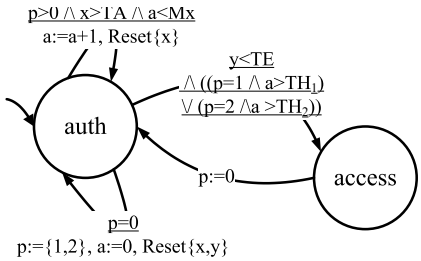


Fig. 6. A Client in CorSSO Protocol. $p$ is the chosen sub-policy, $a$ contains the number of collected authentications, $x$ and $y$ are the timers for collecting certificates and expiring sub-policy respectively.

We compare the performance of our model checker with RED. We first verify the safety property that all clients in *access* have acquired sufficient certificates required by the chosen policy. Then we implant a bug by mistyping $\texttt{TH}_2$ for $\texttt{TH}_1$ in Figure 5.2. This may raise a violation of the safety property if $\texttt{TH}_1 < \texttt{TH}_2$. Systems with 2 to 25 clients are checked by both xBMC and RED.[a] Both RED and xBMC report the safety property is satisfied for normal cases, and the artificial bugs are detected by both tools as expected. The performance results are shown in Table 5. Instead of exploring all regions in the system, xBMC guarantees the correctness by induction at the third step. On the other hand, the traditional reachability analysis in RED has to explore all representatives of equivalent states. Consequently, the time spent by xBMC is only a fraction of that required by RED. For all cases with the crafted bug, xBMC reports that the property is falsified at the $12^{th}$ step. Since SAT-based BMC is efficient for finding defects in design, the performance of xBMC is in accord with our expectations. Compared to RED, xBMC spends only 3.33% time cost to find a counterexample in the system with 11 clients. Note that xBMC stops once a bug is detected, which means that the performance in bug hunting may not necessarily depend on system size.

## 6. Related Work and Discussion

The verification of timed automata by checking satisfiability has been the topic of several research projects. Most research works encode the evaluation of atomic constraint to variants of predicate calculus with real variables. Niebert et al. [20] represented the bounded reachability problem in Boolean variables and numerical constraints of Pratt's difference logic. Audemard et al. [4] took a clock as a real variable, and reduced the bounded verification of timed systems to the satisfiability

[a] We did not turn on the symmetry reduction option in RED, even though the system is highly symmetric.

Table 5. Verifying CorSSO using RED and xBMC

| | **Correctness Guarantee** | | **Bug Hunting** | |
|---|---|---|---|---|
| #p | *RED 5.0* | *xBMC* | *RED 5.0* | *xBMC* |
| 2 | 0.25s | 0.03s | 0.24s | 10.00s |
| 3 | 2.71s | 0.03s | 2.64s | 29.11s |
| 4 | 18.24s | 0.11s | 17.50s | 50.55s |
| 5 | 89.25s | 0.26s | 85.23s | 99.81s |
| 6 | 338.86s | 0.41s | 316.28s | 153.97s |
| 7 | 1076.37s | 0.59s | 990.16s | 278.96s |
| 8 | 2960.56s | 0.94s | 2734.60s | 554.69s |
| 9 | 7169.19s | 4.94s | 6545.04s | 739.46s |
| 10 | 15950.74s | 5.87s | 14727.29s | 582.09s |
| 11 | 33201.08s | 12.38s | 30722.57s | 746.34s |
| 12 | T/O | 17.81s | T/O | O/M |
| 20 | N/A | 185.45s | N/A | O/M |
| 25 | N/A | 484.78s | N/A | O/M |

of a math-formula with linear mathematical relations having real variables. Moura et al. [17] also used real variables to represent infinite state systems.

Seshia and Bryant [22]proposed an unbounded, fully symbolic model checking technique by translating Quantified Separation Logic to equivalent quantified Boolean formula. Based on their method, both SAT and BDD techniques can be applied to verify real time systems. They used CUDD packages to implement their tool TMV. It would be interesting to compare zone and region algorithms in SAT-based model checking but their SAT-based model checker is unavailable to the public. The closest research to ours is Penczek et al [21, 26]. They also target region automata but encode them by slicing each time unit into more segments. Loop-free termination is applied but induction is not incorporated. In Table 6, we compare their encodings and ours to verify Fisher's protocol. Obviously, xBMC induces less variables and clauses, no matter whether their enhancement, i.e. forward projection(BBMC-ARG), is applied.

Table 6. Formulas comparison via detecting mutual exclusion violations of Fischer's protocol with A=1 and B=2. BBMC found the witness at the 12th iteration while xBMC found the witness at the 15th iteration

| | BBMC-RG | | BBMC-ARG | | xBMC | |
|---|---|---|---|---|---|---|
| *#p* | #variables | #clauses | #variables | #clauses | #variables | #clauses |
| 2 | 5,434 | 15,197 | 5,533 | 15,102 | 4,502 | 13,770 |
| 5 | 37,488 | 110,471 | 30,851 | 90,079 | 22,577 | 77,948 |
| 10 | 171,229 | 513,965 | 126,801 | 379,470 | 83,652 | 300,176 |
| 15 | 358,999 | 1,081,790 | 311,501 | 942,085 | 182,842 | 645,297 |
| 20 | 824,374 | 2,493,481 | 556,987 | 1,686,384 | 321,347 | 1,150,023 |

Researchers have also tried to determine whether iterative satisfiability analysis can terminate early if more restrictive formulas are generated based on satisfiability results from previous iterations. Moura et al. [18, 19] achieved this by using induction rules to prove the safety properties of infinite systems. Although they were able to detect cases where early termination was possible, they could not guarantee termination. Sorea [24] checked full LTL formulas based on predicate abstraction to extend BMC capabilities. Compared to encoding abstract predicates, encoding regions themselves provides at least two advantages – simplicity and an intrinsic bound for termination.

Unlike other reachability analysis techniques for timed automata, discretization

allows us to deploy inductive methods rather straightforwardly. However, it is unclear how to apply the same technique in BDD [5], DBM [13] or CRD [25]. It would also be interesting to develop a corresponding inductive method for them and compare their performance with our discretization approach.

## 7. Conclusion and Future Work

BMC is more efficient in identifying bugs, especially for systems with a large number of program variables. However, its correctness guarantee performance could be disappointing. With induction, it is now possible to prove safety properties efficiently by BMC in some cases. With the help of discretization, we are able to migrate the success of the discrete-system verification to timing-behavior analysis. We applied induction algorithms to our previous research on discretization of region automata, and thereby reduced the reachability analysis of dense-time systems to satisfiability. The results of our primitive experiments indicate that even without enhancements (e.g. symmetry reduction, forward projection, and abstraction), our approach is more efficient than RED in correctness guarantee as well as bug hunting. However, two limitations of our approach are that a) the performance is sensitive to the largest constraint constant, and b) the performance depends on whether and when the induction successes.

## References


1. R. Alur, C. Courcoubetis and D. Dill, "Model-checking for Real-time Systems." In IEEE 5th Annual Symposium on Logic In Computer Science (LICS), Philadelphia, June 1990.
2. R. Alur and D. L. Dill, "A Theory of Timed Automata." Theoretical Computer Science 126, pp. 183-235, 1994.
3. N. Amla, R. Kurshan, K. McMillan and R. K. Medel, "Experimental Analysis of Different Techniques for Bounded Model Checking." In Proc. of TACAS'03, LNCS, Warsaw, Poland, 2003.
4. G. Audemard, A. Cimatti, A. Korniowicz and R. Sebastiani, "Bounded Model Checking for Timed Systems." In Doron Peled and Moshe Y. Vardi, editors, Formal Techniques for Networked and Distributed Systems - FORTE'02, LNCS 2529, pp. 243-259. Springer-Verlag, November 2002.
5. R. E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation." IEEE Trans. Computers 35(8), pp. 677-691, 1986.
6. I. Beer, S. Ben-David and A. Landver, "On-the Fly Model Checking of RCTL Formulas." In Proc. of the $10^{th}$ CAV, LNCS 818, 1998.
7. A. Biere, A. Cimatti, E.M. Clarke, M. Fujita, Y. Zhu,"Symbolic Model Checking Using SAT Procedures Instead of BDDs." In Proc. of the 36th Design Automation Conference, pp. 317-320, 1999.
8. P. Bjesse and K. Claessen, "SAT-based verification without state space traversal." In Proc. 3rd Int. Conf. On Formal Methods in Computer Aided Design, LNCS 1954, pp. 372, 2000.
9. M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis and S. Yovine, "Kronos: a Model-Checking Tool for Real-Time Systems." In Proc. of the 10th Conference on

Computer-Aided Verification, CAV'98. LNCS 1427, Springer-Verlag, 1998.

10. E. Clarke, A. Biere, R. Raimi,and Y. Zhu, "Bounded Model Checking Using Satisfiability Solving." In Formal Methods in System Design, July 2001.
11. A. Göllü, A. Puri, and P. Varaiya, "Discretization of timed automata." In Proc. of the 33rd IEEE conference on decision and control, pp. 957-958, 1994.
12. W. Josephson, E. G. Sirer and F. B. Schneider, "Peer-to-Peer Authentication with a Distributed Single Sign-On Services." In Proc. of IPTPS04, 2004.
13. K. G. Larsen, P. Pettersson, and Y. Wang, "Compositional and Symbolic Model Checking of Real-time System." In Proc. RTSS'95, Pisa, Italy, 1995.
14. K. G. Larsen, P. Pettersson, and Y. Wang, "UPPAAL in a Nutshell." In Int. Journal on Software Tools for Technology Transfer 1(1-2), pp. 134-152, 1998.
15. M. O. Moller, H. Rueß, and M. Sorea, "Predicate Abstraction for Dense Real-time Systems." In Proc. of Theory and Practice of Timed Systems (TPTS'2002), 2002.
16. M.W. Moskewicz, C.F. Madigan, Y. Zhao, L. Zhang and S. Malik, "Chaff: Engineering an Efficient SAT Solver." In Proc. of the 38th Design Automation Conference (DAC'01), June 2001.
17. L. de Moura, H. Rueß and M. Sorea, "Lazy Theorem Proving for Bounded Model Checking over Infinite Domains." In Proc. of CADE'02, LNCS 2392, pp. 438-455, 2002.
18. L. de Moura, H. Rueß and M. Sorea, "Bounded Model Checking and Induction: from Refutation to Verification." In Proc. of CAV'03, LNCS 2725, pp. 14-26, 2003.
19. L. de Moura, S. Owre, H. Rueß, J. Rushby, N. Shanker and M. Sorea, "SAL 2", accepted for publication, CAV'2004.
20. P. Niebert, M. Mahfoudh, E. Asarin, M. Bozga, N. Jain, and O. Maler,"Verification of Timed Automata via Satisfiability Checking." In Proc. of FTRTFT'02, LNCS 2469, pp. 225-244, Springer-Verlag, 2002.
21. W. Penczek, B. Wozna and A. Zbrzezny, "Towards Bounded Model Checking for the Universal Fragment of TCTL." In Proc. of FTRTFT'02, LNCS 2469, pp. 265-288, Springer-Verlag, 2002.
22. S. A. Seshia and R. Bryant, "Unbounded, Fully Symbolic Model Checking of Timed Automata using Boolean Methods." In Proc. of CAV'03. LNCS 2725, Springer-Verlag, 2003.
23. M. Sheeran, S. Singh and G. Stålmarck, "Checking Safety Properties Using Induction and a SAT-solver." In Proc. of FMCAD 2000. LNCS 1954:108, 2000.
24. M. Sorea, "Bounded Model Checking for Timed Automata." CSL Technical Report SRI-CSL-02-03, 2002.
25. F. Wang, "Efficient Verification of Timed Automata with BDD-like Data-Structures." VMCAI'03, LNCS 2575, Springer-Verlag, 2003.
26. B. Wozna, W. Penczek and A. Zbrzezny, "Checking Reachability Properties for Timed Automata via SAT." Fundamenta Informaticae, Vol. 55(2), pp. 223-241, 2003.
27. F. Yu and B.-Y. Wang, "Toward Unbounded Model Checking for Region Automata." In Proc. of ATVA 2004, LNCS 3299, pp. 20-33, Taipei, Taiwan, Nov. 2004.
28. F. Yu, B.-Y. Wang and Y.-W. Huang, "Bounded Model Checking for Region Automata." In Proc. of FORMATS and FTRTFT 2004, LNCS 3253, pp. 246-263, Grenoble, France, Sep. 2004.