# Bounded Model Checking for Region Automata[*]

Fang Yu, Bow-Yaw Wang and Yao-Wen Huang

Institute of Information Science, Academia Sinica
Nankang, Taipei 115, Taiwan

{yuf, bywang, ywhuang}@iis.sinica.edu.tw

**Abstract.** For successful software verification, model checkers must be capable of handling a large number of program variables. Traditional, BDD-based model checking is deficient in this regard, but bounded model checking (BMC) shows some promise. However, unlike traditional model checking, for which time systems have been thoroughly researched, BMC is less capable of modeling timing behavior—an essential task for verifying many types of software. Here we describe a new bounded model checker we have named *xBMC*, which we believe solves the reachability problem of dense-time systems. In xBMC, regions and transition relations are represented as Boolean formulae using discrete interpretations. In an experiment using well-developed model checkers to verify Fischer's protocol, xBMC outperformed both traditional (Kronos [8], Uppaal [16], and Red [26]) and bounded (SAL [21]) model checkers by being able to verify up to 22 processes, followed by Red with 15 processes. Therefore, although xBMC is less efficient in guaranteeing system correctness, it provides an effective and practical method for timing behavior verification of large systems.

## 1 Introduction

The successful use of model checking for software verification requires the ability to handle a large number of program variables. Because of problems associated with state explosion, this remains a difficult problem for conventional, BDD-based model checkers. SAT-based bounded model checking (BMC) [7][9] is showing some promise in this regard. A recent comparison [5] of the two techniques shows that the first requires more space and the second more time. Therefore, as Nierbert et al. [22] suggested, even though BMC is less efficient in guaranteeing the correctness of software systems, it has benefits in terms of bug hunting, especially for systems too large for complete verification. Furthermore, since numerous proposals for improving SAT solver efficiency have been made [17][19], BMC's drawback, i.e. its speed, can be improved. Due to these factors, BMC has recently gained acceptance for software verification purposes [10][13]. However, an important deficiency with BMC is its lack of support for timing behavior modeling, considered essential for verifying many types of software (e.g., embedded systems and protocol implementations). This deficiency is the focus of this paper.

In model checking, the verification of most temporal safety properties can be reduced to reachability analysis [6][8]. Yovine [28] has defined the reachability problem as a question of (given two dense-time system states) whether there exists an execution that

[*] This work is partially supported by National Science Council NSC 92-2213-E-001 -023-.

starts in one state and reaches another. Our emphasis here is on solving the reachability problem of dense-time systems.

Timed automata are state transition graphs augmented with a finite set of clocks. Alur, Courcoubetis and Dill [1][2] defined finite equivalent classes, called *regions*, to represent infinite states, and used region automata to represent exact states. They thus proved the complexity of the reachability problem, but failed to provide a region automata implementation—a task that few researchers have attempted.

Biere et al. [7] have proposed a BMC-based approach for solving the reachability problem within bounded steps; their efforts have served as a catalyst for many studies on enhancing verification performance (e.g., [5][9][18][20][22][23][24][25]). In BMC, an initial state and a transition relation are transformed into SAT formulae. At each iteration, a copy of the transition relation (expressed in the following state variables) is added, and an efficient SAT solver is used to iteratively solve the expanding formulae.

We apply BMC techniques to region automata to make feasible the explicit implementation of regions. Göllü et al. [11] have proposed discretizations of dense time automata and have shown that a discrete time trajectory traverses the same timer region sequence as its corresponding dense time trajectory. This provides us with a sound base. In this project we not only characterize regions as combinations of discrete interpretations, but also precisely encode these interpretations' settings as Boolean formulae. To eliminate discretization side effects such as those induced in Göllü et al., we suggest using an exceptional successor formula that prevents timing behavior distortions.

We prove that solving these Boolean formulae's satisfiability is the equivalent of solving the forward reachability problem of dense-time systems. We attempt to incorporate these ideas into our *xBMC*, a bounded model checker that cooperates with zChaff [19], an efficient SAT solver. Our experimental results support that xBMC is more scalable for bug hunting than both traditional (Kronos [8], Uppaal [16], and Red [26]) and bounded (SAL [21]) model checkers by being able to verify Fischer's protocol up to 22 processes, followed by Red with 15 processes.

The rest of this paper is structured as follows. In Section 2 we briefly describe time automata with both discrete and clock variables. In Section 3 we provide details about our novel method that uses discrete interpretation formulae to encode exact behaviors of region automata. An explanation of the Boolean encoding of discrete interpretation formulae is given in Section 4. A reachability analysis is given in Section 5, and experimental results are summarized in Section 6. After discussing related works in Section 7, we offer our conclusions in Section 8.

## 2 Timed Automata

A timed automaton (TA) [1][2] is an automaton together with a finite set of clock variables. Its behavior consists of a) alternating discrete transitions that are constrained by guarded conditions among discrete and clock variables and b) time passages in which the automaton remains in one state while clock values increase at a uniform rate. To represent these behaviors using discrete interpretations, we define a TA that considers both discrete and clock variables, rather than one that only considers the discrete parts as locations.

## 2.1 Constraint and Interpretations

For a set $D$ of discrete variables and a set $X$ of clock variables, set $\Phi(D,X)$ of both constraints $\varphi$ is defined by the grammar $\varphi := ff \mid d = q \mid x \lhd c \mid \neg\varphi \mid \varphi_1 \vee \varphi_2$, where $d \in D$ and $q \in dom(d)$, $x \in X$, $\lhd \in \{<, \le, =\}$, and $c \in N$ is a natural number. Typical short forms are $tt \equiv \neg ff, \varphi_1 \wedge \varphi_2 \equiv \neg((\neg\varphi_1) \vee (\neg\varphi_2))$ and $\varphi_1 \to \varphi_2 \equiv \neg\varphi_1 \vee \varphi_2$.

A discrete interpretation $s$ assigns to each discrete variable a non-negative integer that represents one value from its predefined domain (i.e., $s : D \mapsto N$). A clock interpretation $v$ assigns a non-negative real value to each clock (i.e., $v : X \mapsto R^+$). We say that an interpretation pair $(s,v)$ for $D \cup X$ satisfies constraint $\varphi$ over $D \cup X$ if and only if $\varphi$ is evaluated as being true according to the values given by $(s,v)$.

## 2.2 Time Automata

A TA is a tuple of $\langle D, X, A, I, E \rangle$, where

1) $D$ is a finite set of discrete variables, with each $d \in D$ having a predefined finite domain denoted by $dom(d)$,
2) $X$ is a finite set of clock variables,
3) $A$ is an action set, which is a finite set of discrete variable assignments,
4) $I$ specifies an initial condition, and
5) $E \subseteq \Phi(D,X) \times A^* \times 2^X$ is a set of edges. An edge $\langle \varphi, a, \lambda \rangle$ represents a transition, with $\varphi$ acting as a triggering condition of $\Phi(D,X)$ that specifies where and when the transition can be fired. $a \in A^*$ is an action sequence that performs a series of discrete variable assignments. $\lambda \subseteq X$ is a set of clocks that are reset when the transition fires.

For some action $a \in A^*$, $s[a]$ denotes the discrete interpretation after applying $a$ to $s$. For $\delta \in R^+$, $v + \delta$ denotes a clock interpretation that maps each clock $x$ to the value $v(x) + \delta$. For $\lambda \subseteq X$, $v[\lambda := 0]$ denotes the clock interpretation that assigns 0 to each $x \in \lambda$ and that agrees with $v$ over the rest of the clocks.

The semantics of a TA is a transition system $\langle Q, \to \rangle$, where $Q$ is the set of *states* and $\to$ is the *transition relation*. A *state* of a TA is a pair $(s,v)$ such that $s$ is a discrete interpretation of $D$ and $v$ is a clock interpretation of $X$. A state $(s,v)$, where $s$ maps discrete variables to values that satisfy $I$ and $v(x) = 0$ for all $x \in X$, is an initial state of a TA. There are two types of $\to$:

1) For a state $(s,v)$ and an increment $\delta \in R^+$, $(s,v) \to_\delta (s, v+\delta)$.
2) For a state $(s,v)$ and an edge $<\varphi, \tau, \lambda>$ such that $(s,v)$ satisfies $\varphi$, $(s,v) \to_e (s[\tau], v[\lambda := 0])$.

A run $r$ of a TA is an infinite sequence of states and transitions $r = (s_0, v_0) \to (s_1, v_1) \to \cdots$, where for all $i \in N, (s_i, v_i) \in Q$. An arbitrary interleaving of the two transition types is permissible. $Run(s,v)$ denotes a set of runs starting at $(s,v) \in Q$. A state $(s',v')$ is reachable from $(s,v)$ if it belongs to some run starting at $(s,v)$. We define $Reach(s,v)$, where $Reach(s,v) = \{(s',v') \mid \exists r = (s_0,v_0) \to \cdots \in Run(s,v), \exists i \in N.(s_i,v_i) = (s',v')\}$, to be the set of states reachable from $(s,v)$.

## 3 Discrete Interpretations of Region Automata

In this section we give a brief description of region automata based on [1][2], and then propose a robust encoding method using discrete interpretations.

### 3.1 Paired Interpretations of Equivalent Classes

System states change as time progresses, but some changed states are not distinguished by constraints. Based on this observation, Alur et al. [1] defined clock interpretation equivalence and proposed the use of region graphs for the verification of timed automata. For each $x \in X$, let $c_X$ be the largest constant that x is compared to within any triggering condition. For $t \in R^+$, let $\lfloor t \rfloor$ denote *t*'s integral part and $frac(t)$ denote *t*'s fraction, which equals $t - \lfloor t \rfloor$. A formal definition of clock interpretation equivalence is given as:

**Definition 1.** *For clock interpretations $v$ and $v'$ in a TA, $v \cong v'$ if and only if*

1) *For each clock $x \in X$, either $\lfloor v(x) \rfloor$ and $\lfloor v'(x) \rfloor$ are the same, or $v(x)$ and $v'(x)$ are both greater than $c_X$.*
2) *For each pair of clocks, $x, y \in X$ such that $v(x) \le c_X$ and $v(y) \le c_Y$, $frac(v(x)) \le frac(v(y))$ if and only if $frac(v'(x)) \le frac(v'(y))$, and $frac(v(x)) = 0$ if and only if $frac(v'(x)) = 0$.*

We represent the set of clock assignments belonging to an equivalent class as a pair of discrete interpretations $v_d$ and $v_\gamma$, respectively mapping integral parts of clock assignments and fraction pair orderings. Given an equivalent class $[v]$, integral parts of the clock assignments stand for the discrete interpretation $v_d$ in (1), which maps each clock $x \in X$, where $v(x) = t$, into an integer representing an interval from $\{[0,0],(0,1),[1,1],\cdots,(c_X - 1, c_X),[c_X,c_X],(c_X,\infty)\}$.

$$v_d(x) = \begin{cases} 2\lfloor t \rfloor, \text{ if } \lfloor t \rfloor \le c_X \wedge frac(t) = 0 \\ 2\lfloor t \rfloor + 1, \text{ if } \lfloor t \rfloor \le c_X \wedge frac(t) \ne 0 \\ 2c_X + 1, \text{ otherwise} \end{cases} \qquad \textbf{(1)}$$

We say $v(x)$ falls into a *singular* interval when $v_d(x)$ is even and into an *open* interval when $v_d(x)$ is odd; in addition, $v(x)$ falls into a *maximal* interval when $v_d(x)$ is $2c_X + 1$. Given a discrete interpretation $v_d$, let $O(v_d) \subseteq X$ denote the set of clocks whose values are both odd and less than $2c_X + 1$. Then, the discrete interpretation $v_\gamma$ in (2) maps each clock pair $(x, y)$, where $x, y \in O(v_d)$ and $x < y$, into a relation from $\{\prec, \succ, \approx\}$, which stands for pair orderings of fractions of an equivalent class $[v]$.

$$v_\gamma(x,y) = \begin{cases} \prec, \text{ if } frac(v(x)) < frac(v(y)) \\ \succ, \text{ if } frac(v(x)) > frac(v(y)) \\ \approx, \text{ if } frac(v(x)) = frac(v(y)) \end{cases} \quad \textbf{(2)}$$

A pair $(v_d, v_\gamma)$ denotes a clock assignment $v$ such that $v_d$ and $v_\gamma$ follow (1) and (2), respectively. For example, an equivalent class $(1 < x < y < 2) \wedge z = 1$ is represented by the pair $(v_d, v_\gamma)$, where $v_d(x) = 3 \wedge v_d(y) = 3 \wedge v_d(z) = 2 \wedge v_\gamma(x, y) = \prec$. The equivalence of two discrete interpretation pairs is defined as:

**Definition 2.** *Two pairs of interpretations are equivalent (denoted as* $(v_d, v_\gamma) \cong (v'_d, v'_\gamma)$*) if and only if the following conditions hold.*
*1) For each clock* $x \in X$ , $v_d(x) = v'_d(x)$.
*2) For each pair (x,y), where* $x, y \in O(v_d)$ and $x < y$ , $v_\gamma(x, y) = v'_\gamma(x, y)$.

Note that the first condition implies that $O(v_d) = O(v'_d)$.

**Lemma 1.** $v \cong v' \leftrightarrow (v_d, v_\gamma) \cong (v'_d, v'_\gamma)$**.**

**Proof:** ($\Rightarrow$) For the first condition of Definition 2, if each clock $x \in X$ , while $v(x)$ and $v'(x)$ are both greater than $c_X$ , $v_d(x) = v'_d(x) = 2c_X + 1$ ; if $\lfloor v(x) \rfloor$ and $\lfloor v'(x) \rfloor$ are the same, since $frac(v(x)) = 0$ if and only if $frac(v'(x)) = 0$ , $v_d(x) = v'_d(x) = 2\lfloor v(x) \rfloor$ or $2\lfloor v(x) \rfloor + 1$. To prove the second condition of Definition 2, it is crucial to note that a) $\left(frac(v(x)) \le frac(v(y)) \wedge frac(v(y)) \le frac(v(x))\right) \leftrightarrow v_\gamma(x, y) = \approx$ , b) $\left(frac(v(x)) \le frac(v(y)) \wedge \neg\left(frac(v(y)) \le frac(v(x))\right)\right) \leftrightarrow v_\gamma(x, y) = \prec$ , and c) $\left(\neg\left(frac(v(x)) \le frac(v(y))\right) \wedge frac(v(y)) \le frac(v(x))\right) \leftrightarrow v_\gamma(x, y) = \succ$. Since for each pair of clocks, where $x, y \in O(v_d)$ and $x < y$ , $frac(v(x)) \le frac(v(y))$ if and only if $frac(v'(x)) \le frac(v'(y))$, it follows that $v_\gamma(x, y) = v'_\gamma(x, y)$. ($\Leftarrow$) The proof follows in a similar way. ▮

As shown in Lemma 1, sets constrained by equivalent discrete interpretation pairs are in the same equivalent class, and each equivalent class is represented by an equivalent discrete interpretation pair. Accordingly, a *region* $(s, [v])$ can be precisely represented as a tuple $(s, v_d, v_\gamma)$ , where three discrete interpretations $s : D \mapsto N, v_d : X \mapsto N$ , and $v_\gamma : X \times X \mapsto \{\prec, \succ, \approx\}$ are involved.

## 3.2 Successor

A successor relation that captures a region moving into a subsequent region due to time passage is defined as:

**Definition 3.** *Let $\alpha,\beta$ be two distinct regions of a TA. $\beta$ is the successor of $\alpha$, written as $succ(\alpha)=\beta$, if and only if for each $v\in\alpha$, there exists a positive $\delta\in R^{+}$ such that a) $v+\delta\in\beta$, and b) $v+\delta'\in\alpha\cup\beta$* for all $\delta'<\delta$.

A region $\alpha$ is said to be out of bounds if and only if for each $v\in\alpha$ and $x\in X, v(x)>c_x$. For an out-of-bound region $\alpha$, its successor relation is defined as $succ(\alpha)=\alpha$. Given a TA with $X=\{x,y\}$, regions can be separated into nine types according to x and y evaluations: point (•), slash (╱), vertical line (|), horizontal line (—), triangle (◢), back triangle (◤), vertical rectangle (▮), horizontal rectangle (▬) and square (■). Respective discrete interpretation conditions and successor relations of these are shown in Table 1.

**Table 1. Successor conditions in the two-clock system.**

| $I$ | $\frac{type}{type'}$ | $\frac{\psi_i}{\psi'_i}$ |
|---|---|---|
| 1 | $\frac{•}{╱■}$ | $\frac{v_d(x) \text{ is } even \text{ and } v_d(y) \text{ is } even}{v'_d(x)=v_d(x)+1, v'_d(y)=v_d(y)+1, v'_\gamma(x,y)=\approx}$ |
| 2 | $\frac{|}{◤▮▬■}$ | $\frac{v_d(x) \text{ is } even \text{ and } v_d(y) \text{ is } odd}{v'_d(x)=v_d(x)+1, v'_d(y)=v_d(y), v'_\gamma(x,y)=\prec}$ |
| 3 | $\frac{—}{◢▮▬■}$ | $\frac{v_d(x) \text{ is } odd \text{ and } v_d(y) \text{ is } even}{v'_d(x)=v_d(x), v'_d(y)=v_d(y)+1, v'_\gamma(x,y)=\succ}$ |
| 4 | $\frac{◢}{|}$ | $\frac{v_d(x) \text{ is } odd, v_d(x)<2c_x+1, v_d(y) \text{ is } odd, v_d(y)<2c_y+1, v_\gamma(x,y)=\succ}{v'_d(x)=v_d(x)+1, v'_d(y)=v_d(y)}$ |
| 5 | $\frac{◤}{—}$ | $\frac{v_d(x) \text{ is } odd, v_d(x)<2c_x+1, v_d(y) \text{ is } odd, v_d(y)<2c_y+1, v_\gamma(x,y)=\prec}{v'_d(x)=v_d(x), v'_d(y)=v_d(y)+1}$ |
| 6 | $\frac{╱}{•}$ | $\frac{v_d(x) \text{ is } odd, v_d(x)<2c_x+1, v_d(y) \text{ is } odd, v_d(y)<2c_y+1, v_\gamma(x,y)=\approx}{v'_d(x)=v_d(x)+1, v'_d(y)=v_d(y)+1}$ |
| 7 | $\frac{▮}{|}$ | $\frac{v_d(x) \text{ is } odd, v_d(x)<2c_x+1, v_d(y)=2c_y+1}{v'_d(x)=v_d(x)+1, v'_d(y)=v_d(y)}$ |
| 8 | $\frac{▬}{—}$ | $\frac{v_d(x)=2c_x+1, v_d(y) \text{ is } odd, v_d(y)=2c_y+1}{v'_d(x)=v_d(x), v'_d(y)=v_d(y)+1}$ |
| 9 | $\frac{■}{■}$ | $\frac{v_d(x)=2c_x+1 \text{ and } v_d(y)=2c_y+1}{v'_d(x)=v_d(x), v'_d(y)=v_d(y)}$ |

Accordingly, it is possible to define a formula for the two-clock successor $\phi_{x,y}$ in (3) using the conditions of current and succeeding discrete interpretations (see Table 1). Lemma 2 shows the correctness of encoding a two-clock successor as $\phi_{x,y}$.

$$\phi_{x,y} \equiv \bigvee_{1 \le i \le 9} \psi_i \wedge \psi'_i \qquad \textbf{(3)}$$

**Lemma 2.** *Given a two-clock TA and a region* $\alpha$ *represented by* $(s, v_d, v_\gamma)$, $succ(\alpha)$ *is represented by* $(s, v'_d, v'_\gamma)$ *if and only if* $\phi_{x,y}$ *is evaluated as true according to the values given by* $(v_d, v_\gamma, v'_d, v'_\gamma)$.

**Proof Sketch:** The correctness of this insight is shown by the conditions in Table 1. All possible cases are considered, since $\vee_{1 \le i \le 9} \psi_i$ is true. Since $\forall i \ne j, \psi_i \wedge \psi_j$ is false, each case presents a unique type. Finally, in all cases the condition of the current equivalent class and its successor is specified by $\psi_i \wedge \psi'_i$.■

In order to generalize $\phi_{x,y}$ into multi-clock systems, instead of inspecting all clock values each time, the entire consensus is derived by intersecting two-clock formulae. Our initial attempt detailed intersecting the $\phi_{x,y}$ of each distinct clock pair, but this raises contradictions. For example, given a region $1 < x < y < 2 \wedge z = 1$ represented by $v_d(x) = 3 \wedge v_d(y) = 3 \wedge v_d(z) = 2 \wedge v_\gamma(x,y) = \prec$, the conjunction of $\phi_{x,y}$ for each distinct clock pair implies

$$\begin{aligned} &(v'_d(x) = 3 \wedge v'_d(y) = 4) \\ &\wedge (v'_d(x) = 3 \wedge v'_d(z) = 3 \wedge v'_\gamma(x,z) = \succ) \\ &\wedge (v'_d(y) = 3 \wedge v'_d(z) = 3 \wedge v'_\gamma(y,z) = \succ) \end{aligned}$$

, which is contradictory because $v'_d(y) = 3 \wedge v'_d(y) = 4$. The contradiction occurs because *y* needs to increase when compared to *x*, but needs to remain the same value when compared to *z*. To prevent contradiction, we make the following observations. Contradictions arise from clocks falling into open (but not maximal) intervals. Clocks that have even values must increase. If none are found, clocks having the largest fraction of all clocks must increase. Based on these observations, we add an auxiliary case in which clocks that might produce contradictions are allowed to stutter.

**Table 2. Stuttering condition.**

| $\frac{type}{type'}$ | $\frac{\psi_S}{\psi'_S}$ |
|---|---|
| $\frac{◢╱◤▮▬}{◢╱◤▮▬}$ | $\frac{v_d(x) \text{ is } odd, v_d(y) \text{ is } odd, (v_d(x) < 2c_x + 1) \vee (v_d(y) < 2c_y + 1)}{v'_d(x) = v_d(x), v'_d(y) = v_d(y), v'_\gamma(x,y) = v_\gamma(x,y)}$ |

After the addition, the stuttering two-clock formula is $\phi^S_{x,y}$ in (4).

$$\phi^{s}{}_{x,y} = \phi_{x,y} \vee (\psi_s \wedge \psi'_s) \qquad \textbf{(4)}$$

For the same example, given a region $1 < x < y < 2 \wedge z = 1$, the conjunction of $\phi^{s}_{x,y}$ for each distinct clock pair implies that

$$\begin{aligned}&\left((v'_d(x) = 3 \wedge v'_d(y) = 4) \vee (v'_d(x) = 3 \wedge v'_d(y) = 3 \wedge v'_\gamma(x,y) = \prec)\right)\\ &\wedge (v'_d(x) = 3 \wedge v'_d(z) = 3 \wedge v'_\gamma(x,z) = \succ)\\ &\wedge (v'_d(y) = 3 \wedge v'_d(z) = 3 \wedge v'_\gamma(y,z) = \succ)\end{aligned}.$$

This is equal to $v'_d(x) = 3 \wedge v'_d(y) = 3 \wedge v'_d(z) = 3 \wedge v'_\gamma(x,y) = \prec \wedge v'_\gamma(x,z) = \succ \wedge v'_\gamma(y,z) = \succ$, a precise representation of the successor, $1 < z < x < y < 2$. However, while all of these clocks have odd values (i.e., falling in an open interval), auxiliary stuttering may incur distorted timing behavior in which all clocks refuse to increase. This can be prevented by adding a negation clause of all clocks stuttering. The successor condition for general cases is:

$$\phi \equiv \bigwedge_{x,y \in X, x<y} \phi^{s}{}_{x,y} \wedge \neg\left(\bigwedge_{x,y \in X, x<y} \psi_s \wedge \psi'_s\right) \qquad \textbf{(5)}$$

**Lemma 3.** *Given a TA and a region $\alpha$ represented by $(s, v_d, v_\gamma)$, $succ(\alpha)$ is represented by $(s, v'_d, v'_\gamma)$ if and only if $\phi$ is evaluated to true according to the values given by $(v_d, v_\gamma, v'_d, v'_\gamma)$.*

**Proof sketch:** (⇒) It is easy to see that $succ(\alpha)$ implies that, for each pair of clocks, $\phi^{s}_{x,y}$ is evaluated to true (according to Lemma 2), and not all stuttering cases are allowed (according to Definition 3). (⇐) Let $\chi = \{x \mid x \in X, v'_d(x) = v_d(x) + 1 \text{ or } v_d(x) = 2c_x + 1\}$. If $\phi$ is evaluated to true according to the values given by $(v_d, v_\gamma, v'_d, v'_\gamma)$, there exists at least one pair of clocks such that $\phi_{x,y}$ is evaluated to true, which implies that $\chi$ is not empty. If $\forall x \in \chi, v_d(x) = 2c_x + 1$, then $succ(\alpha) = \alpha$. If $\exists x \in \chi, v_d(x) < 2c_x + 1$, in the following, we prove that all cases satisfy Definition 3.

a) If $\exists x \in \chi, v_d(x)$ is even, then $\forall x \in \chi, v_d(x)$ is even. It can be seen as follows. Assume that there exists $y \in \chi$ and that $v_d(y)$ is odd. Then there exists some clock $x$ such that $v_d(x)$ is even and $\phi^{s}_{x,y}$ is false, which implies that $\phi$ cannot be evaluated to true. If $\forall x \in \chi$, $v_d(x)$ is even, then for each $v \in (v_d, v_\gamma)$, there exists a positive $\delta \in R^+$ such that i) $v + \delta \in (v'_d, v'_\gamma)$ and ii) $v + \delta' \in (v_d, v_\gamma) \cup (v'_d, v'_\gamma)$ for all $\delta' < \delta$.

b) If $\forall x \in \chi, v_d(x)$ is odd, then $\forall x \in \chi$, $x$ has the largest fraction part. It can be seen as follows. Assume that there exists $y \in \chi$ and that $y$ has a not-largest fraction part. Then there exists some clock $x$ such that $x$ has a larger fraction part than $y$ and $\phi^{S}_{x,y}$ is false, which implies that $\phi$ cannot be evaluated to true. If $\forall x \in \chi$, $x$ has the largest fraction part, then for each $v \in \left(v_d, v_\gamma\right)$, there exists a positive $\delta \in R^+$ such that $v + \delta \in \left(v'_d, v'_\gamma\right)$, and $v + \delta' \in \left(v_d, v_\gamma\right) \cup \left(v'_d, v'_\gamma\right)$ for all $\delta' < \delta$.█

## 3.3 Discrete Transitions

In this sub-section, we describe how to trigger an edge using discrete interpretations. Since we use discrete intervals to represent clock values, the first step here is to transform $\varphi$ into a discrete constraint (denoted as $Dis(\varphi)$) by replacing all clock constraints $x \lhd c$ with $v_d(x) \lhd 2c$. The set $\Phi_d(D, X)$ of discrete constraints $Dis(\varphi)$ is defined by the grammar $Dis(\varphi) := ff \mid d = q \mid v_d(x) \lhd 2c \mid \neg Dis(\varphi) \mid Dis(\varphi_1) \vee Dis(\varphi_2)$.

Actions of transitions include a) applying an assignment sequence $\tau$ (denoted as $s[\tau]$), and b) resetting a set of clocks $\lambda$ (denote as $v[\lambda := 0]$). Let $v_d[\lambda := 0]$ denote the discrete interpretation for $X$ that a) assigns 0 to each $x \in \lambda$ and b) agrees with $v_d$ over the rest of the clocks. Since the domain of $v_\gamma$ depends on $O(v_d)$, we remove the reset clocks from $v_\gamma$. Let $v_\gamma\left[O(v_d)\right]$ denote the discrete interpretation that a) agrees with $v_\gamma$ over the pair $(x, y)$, where both $x, y \in O(v_d)$, and b) discards other pairs. Given an edge $\langle \varphi, \tau, \lambda \rangle$, the transition condition over discrete interpretations is defined as:

$$T_{tran} \equiv Dis(\varphi) \wedge s' = s[\tau] \wedge v'_d = v_d[\lambda := 0] \wedge v'_\gamma = v_\gamma\left[O(v'_d)\right] \qquad \textbf{(6)}$$

**Lemma 4.** *Given a TA and two states* $(s, v)$ *and* $(s', v')$, $(s, v) \to_e (s', v')$ *if and only if* $T_{tran}$ *is evaluated to true according to values given by* $\left(s, v_d, v_\gamma, s', v'_d, v'_\gamma\right)$, *where* $\left(v_d, v_\gamma\right)$ *represents [v] and* $\left(v'_d, v'_\gamma\right)$ *represents* $[v']$.

## 3.4 Transition Systems

A TA's transition system is represented by a finite discrete interpretation graph $\langle Q_{\cong}, \to_{\cong} \rangle$, where $Q_{\cong}$ is the set of *interpretation states* and $\to_{\cong}$ is the *interpretation transition relation*. An interpretation state $\left(s, v_d, v_\gamma\right)$ is a triple of discrete interpretations. There are two types of $\to_{\cong}$:

1) For a state $\left(s, v_d, v_\gamma\right)$, $\left(s, v_d, v_\gamma\right) \to_{\cong_\delta} \left(s, v'_d, v'_\gamma\right)$, such that $T_{time}$ in (7) is evaluated to true according to the values given by $\left(s, v_d, v_\gamma, s', v'_d, v'_\gamma\right)$.
2) For a state $\left(s, v_d, v_\gamma\right)$ and an edge $\langle \varphi, \tau, \lambda \rangle$, $\left(s, v_d, v_\gamma\right) \to_{\cong_e} \left(s', v'_d, v'_\gamma\right)$, such that $T_{tran}$ is evaluated to true according to the values given by $\left(s, v_d, v_\gamma, s', v'_d, v'_\gamma\right)$.

$$T_{time} \equiv (s = s') \wedge \phi \qquad \textbf{(7)}$$

$$T \equiv T_{tran} \vee T_{time} \qquad \textbf{(8)}$$

A step condition $T$ of $\rightarrow_{\cong}$ in (8) is the disjunction of (6) and (7). We define the steps and reachable states of a discrete interpretation graph in Definition 4.

**Definition 4.** *We say* $(s, v_d, v_\gamma)$ *can reach* $(s', v'_d, v'_\gamma)$ *in one step, denoted as* $(s, v_d, v_\gamma) \rightarrow_{\cong} (s', v'_d, v'_\gamma)$ *, if and only if* $T$ *is evaluated to true according to values given by* $(s, v_d, v_\gamma, s', v'_d, v'_\gamma)$*. We define* $Reach(s, v_d, v_\gamma)$ *to be the set of interpretation states reachable from* $(s, v_d, v_\gamma)$*.*

$$Reach(s, v_d, v_\gamma) = \left\{ (s', v'_d, v'_\gamma) \mid (s, v_d, v_\gamma) \rightarrow_{\cong}^{*} (s', v'_d, v'_\gamma) \right\}$$

*, where* $\rightarrow_{\cong}^{*}$ *is the reflexive and transitive closure of* $\rightarrow_{\cong}$*.*

The reachability problem of dense-time systems (e.g., whether one state $(s', v')$ is reachable from another state $(s, v)$ ) can then be solved by following Lemma 5.

**Lemma 5**. *Given a TA and two states* $(s, v)$ *and* $(s', v')$, $(s', v') \in Reach(s, v)$ *if and only if* $(s', v'_d, v'_\gamma) \in Reach(s, v_d, v_\gamma)$.

Since regions are finite, we can perform complete reachability analysis by solving bounded reachability problems (detailed in Section 5).

## 4. Boolean Encoding

Before delving into the reachability analysis, we first describe here how we encode discrete interpretation formulae as CNF Boolean ones.

### 4.1 State Variables

The definition of our state variables $B$ is given in (9), in which a set of bit vectors is used to encode an interpretation state. Given each discrete variable's domain and each clock's largest constraint value, $|B|$ is $\Sigma_{d \in D} \lceil \lg |dom(d)| \rceil + \Sigma_{x \in X} \lceil \lg(2c_x + 2) \rceil + |X||X-1|$ . To perform bounded model checking, we add a copy of $B_i$ to the set of state variables at the $i$th iteration.

$$B = \left\{ b_d^k \mid d \in D, 0 \le k \le \lceil \lg |dom(d)| \rceil \right\} \cup \left\{ b_x^k \mid x \in X, 0 \le k \le \lceil \lg(2c_x + 2) \rceil \right\} \qquad (9)$$
$$\cup \left\{ b_{xy}^k \mid x, y \in X, x < y, k = 0 \text{ or } 1 \right\}$$

### 4.2 Discrete Interpretation Encoding

Using state variables *B*, an interpretation state $\left(s,v_d,v_\gamma\right)$ is encoded into a Boolean formula $\beta\left(B\right)$, where $\beta\left(B\right)$ is the conjunction of discrete, interval and relation sub-formulae given in (10). Following standard bit encoding, sub-formulae are built with regard to their individual discrete interpretations. Note that an even-valued clock *x* (the 0th bit is zero) will be encoded with $\neg b_x^0$, making it possible to encode a condition such as "*v*$_d$*(x) is even*" using one literal. This characteristic significantly reduces the complexity in solving the successor formulae described in Section 3.2.

$$\beta\left(B\right)=\bigwedge_{d\in D,0\le k\le\left\lceil \lg\left|dom\left(d\right)\right|\right\rceil}\begin{cases}b_d^k,\text{ if the }k\text{th bit of the value of }s\left(d\right)\text{ is 1}\\ \neg b_d^k,\text{ otherwise}\end{cases} \tag{10}$$
$$\bigwedge_{x\in X,0\le k\le\left\lceil \lg\left|2c_x+2\right|\right\rceil}\begin{cases}b_x^k,\text{ if the }k\text{th bit of the value of }v_d\left(x\right)\text{ is 1}\\ \neg b_x^k,\text{ otherwise}\end{cases}$$
$$\bigwedge_{x,y\in O\left(v_d\right),x<y}\begin{cases}\neg b_{xy}^1\wedge b_{xy}^0,\text{ if }v_\gamma\left(x,y\right)=\prec\\ b_{xy}^1\wedge\neg b_{xy}^0,\text{ if }v_\gamma\left(x,y\right)=\succ\\ b_{xy}^1\wedge b_{xy}^0,\text{ if }v_\gamma\left(x,y\right)=\approx\end{cases}$$

### 4.3 Formula Encoding

We reserve our Boolean state variables using a bit-vector. The translation of a bit-vector logic (used to build the equation for a concrete transition relation) into conjunctive normal form (CNF) is straight forward: we build a circuit representation and then translate it into CNF.

## 5. Reachability Analysis

In this section, we describe how we deal with the reachability problem by iteratively solving the satisfiability of an expanding Boolean formula. We also prove that our procedure provides a sound and complete solution when we reach a big enough bound. Let $Reach_k\left(s,v_d,v_\gamma\right)$ denote the set of states reachable from $\left(s,v_d,v_\gamma\right)$ by unfolding exactly *k* steps. Lemma 6 proves that all regions of a given TA can be reached within constant steps.

**Lemma 6.** *Given a TA having n regions,* $\left(s',v'_d,v'_\gamma\right)\in Reach_k\left(s,v_d,v_\gamma\right)$ *and* $k>n$ *implies the existence of some* $k'<k$ *such that* $\left(s',v'_d,v'_\gamma\right)\in Reach_{k'}\left(s,v_d,v_\gamma\right)$.

**Proof:** If $\left(s',v'_d,v'_\gamma\right)\in Reach_k\left(s,v_d,v_\gamma\right)$ and k>n, then there exists some region that was reached more than once. Assume we reached the region in the *i*th and the *j*th steps, we can derive a new path by removing steps *i+1* to *j*. This implies the existence of some $k' < k$ such that $\left(s', v'_d, v'_\gamma\right) \in Reach_{k'}\left(s, v_d, v_\gamma\right)$.█

Let $T(i)$ denote $\underline{T}$ in (8) over state variables $B_i$ and $B_{i+1}$, and $\beta_{s,v_d,v_\gamma}(i)$ denote $\beta(B)$ in (10) according to the interpretation state $\left(s, v_d, v_\gamma\right)$ over state variables $B_i$. Lemma 7 shows that the bounded reachability problem is equivalent to the satisfiability problem.

**Lemma 7.** $\left(s', v'_d, v'_\gamma\right) \in Reach_k\left(s, v_d, v_\gamma\right)$ *if and only if*
$SAT\left(\beta_{s,v_d,v_\gamma}(0) \wedge T(0) \wedge \cdots \wedge T(k-1) \wedge \beta_{s',v'_d,v'_\gamma}(k)\right)$.

Given an initial condition, a risk condition, a transition condition and an integer bound, we iteratively solve the bounded reachability problem by calling the SAT solver for bounded forward reachability analyses. We unfold the interpretation transition relation until the SAT solver returns a truth assignment or reaches the bound. Let I(*i*) and R(*i*) respectively denote the CNF formulae of the given initial and risk conditions over $B_i$. The implementation of `BoundedFwdReach()` is given in Fig. 1. By conjoining the formula with the negation clause of the risk condition, each iteration's result is saved for use in later iterations

```
BoundedFwdReach(I, R, T, MaxBound)
  var i: 0.. MaxBound;
begin
    k := 0; F := I(i);
    loop forever
      if(SAT(F∧R(i)))return reachable;
      if(i=MaxBound)return unreachable within MaxBound;
      F := F∧¬R(i)∧T(i);
      i := i+1;
end.
```

**Fig. 1.** `BoundedFwdReach()` implementation.

**Theorem.** *Given a TA having n regions,* `BoundedFwdReach()` *is sound and complete when* $\text{MaxBound} \geq n$.

If the risk state is reachable, the formula will be satisfied at some step, and a truth assignment will be returned by zChaff. The procedure will then terminate and generate a counterexample. The formula will keep on expanding if a risk state is not reached. Therefore, if the risk state is unreachable, the procedure terminates when either MaxBound is reached or memory is exhausted. Given a TA having *n* regions, the final formula will contain $n|B|$ branching variables. Since *n* is exponential to both a) the number of clocks and b) each clock's largest constant, the threshold is usually prohibitively expensive.

## 6. Experimental Results

*xBMC 2.0* is written in C and makes use of zChaff [19]. Experiments were run against Fischer's mutual exclusion protocol, which consists of *n* timed automata with each automaton modeling an individual process (Fig 2). Mutual exclusion property was considered violated when A<B. The largest constraint for the local clock of each process was adjusted by increasing the value of *B* and keeping A=1.

We compare our model checker with Kronos [8], Uppaal [16], Red [26] and SAL 2 (infBMC) [21]. The first three tools support full TCTL verification of timed systems, but use different data structures for system state representation. Kronos and Uppaal use DBMs (Difference Bounded Matrices) [15], while Red uses CRDs (Clock Restriction Diagrams) [26]. infBMC is a bounded model checker included in SAL 2 [21], 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 [20] that solves the satisfiability of combinations of real and integer linear arithmetic.

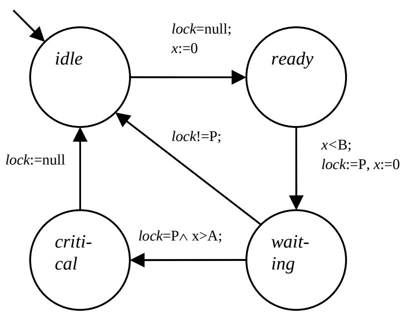

**Fig. 2.** Each process in Fischer's protocol has one local clock *x* and one discrete variable *l (*denoting location), where dom*(l)*={*idle, ready, waiting, critical*}. Processes can access the global pointer *lock* (e.g., assigning the pointer to itself [*lock*:=P]). Initially, all processes are in *idle* and *lock* points to none of them.

**Table 3.** Process number impact on each tool's performance when checking mutual exclusion violations of Fischer's protocol. "O/M" indicates that the model checker ran out of memory. All experiments were performed on a Pentium IV 1.7 GHz computer with 256MB of RAM running the Linux operating system.

| | **Kronos 2.5.2** | | **Uppaal 3.5.1** | | **Red 5.0** | | **SAL 2.1 ( infBMC )** | | **xBMC 2.0** | |
|---|---|---|---|---|---|---|---|---|---|---|
| # | *B*=2 | *B*=4000 | *B*=2 | *B*=4000 | *B*=2 | *B*=4000 | *B=2* | *B=4000* | *B*=2 | *B*=4000 |
| 4 | 0.12s | 0.11s | 0.03s | 0.02s | 0.57s | 0.56s | 86.98s | 95.45s | 3.28s | 20.31s |
| 5 | 0.52s | O/M | 0.03s | 0.04s | 1.95s | 1.95s | 420.98s | 275.82s | 10.49s | 37.32s |
| 6 | O/M | | 0.06s | 0.06s | 5.70s | 4.82s | O/M | O/M | 14.66s | 47.63s |
| 7 | | | 0.16s | 0.17s | 14.47s | 12.90s | | | 16.83s | 47.04s |
| 9 | | | 1.17s | 1.21s | 75.5s | 74.31s | | | 46.90s | 91.35s |
| 11 | | | 5.08s | 9.35s | 321.04s | 353.61s | | | 129.46s | 200.84s |
| 13 | | | 12.21s | O/M | 1129.18s | 1345.08s | | | 111.59s | 447.39s |
| 14 | | | O/M | | 2005.23s | 2471.07s | | | 237.89s | O/M |
| 15 | | | | | 4234.41s | 4238.34s | | | 531.73s | |
| 16 | | | | | O/M | O/M | | | 453.83s | |
| 17 | | | | | | | | | 414.29s | |
| 19 | | | | | | | | | 528.66s | |
| 21 | | | | | | | | | 641.27s | |
| 22 | | | | | | | | | 587.01s | |
| 23 | | | | | | | | | O/M | |

Performance results are shown in Table 3. All verification processes that did not crash reached a violated state. When *B=2*, Kronos failed to construct the product automaton of the system while verifying 6 processes. Uppaal ran efficiently until the number of processes reached 14. Red demonstrated an exceptional data sharing capability and outperformed the other tools in terms of memory utilization and successfully checked 15 processes. infBMC reported all counterexamples at the 10th iterations, but its internal decision procedure crashed while verifying 6 processes. xBMC was capable of reporting all counterexamples within 14 iterations and successfully checked 22 processes. When $B$=4000, the increased number of variables limited xBMC to handling up to 13 processes. On the other hand, performance among the other tools was not significantly affected by increasing values of constraint constants.

## 7. Related Work and Discussion

Due to the many advantages described in Section 1, SAT-based model checking has recently gained considerable favor among software verification researchers. Clarke et al. [10] developed a SAT-based bounded model checker for ANSI C, and we used xBMC to verify Web application code security in an earlier project [13]. Although both projects were successful, neither supported timing behavior modeling.

The verification of timed automata via satisfiability checking has been the focus of several investigations, with most researchers encoding atomic constraint evaluations rather than regions themselves. Niebert et al. [22] represented the bounded reachability problem in Boolean variables and numerical constraints of Pratt’s difference logic, but faced difficulties. The difficulty lay in solving the mixed constraints, which was done by their in-house solver. Audemard et al. [5] treated clocks as real variables and reduced bounded verification of timed systems to the satisfiability of a math formula with linear mathematical relations involving real variables. They demonstrated this approach by implementing a new solver, MATHSAT. They also showed that bounded verification was considerably improved by using symmetry reductions. Moura et al. [20] also used real variables to represent infinite state systems. In [25], Sorea checked full LTL formulae 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.

The approach closest to ours was described by Penczek, Wozna and Zbrzezny in [23][27]. Based on Numerical Decision Diagrams (NDDs) [4], they obtained the set of a region’s representatives by dividing each unit interval into 2$n$ segments ($n$ = number of clocks). Compared to our method, which encodes an exact region based on fraction order and that region’s successor in one step, theirs incurs more discrete time steps. In [27], they considerably improved their performance by applying forward projection. According to their reported experimental results, our xBMC demonstrated compatible performance (in fact, better than their original explicit discretization encoding without applying forward projection) while verifying Fischer’s protocol in similar conditions.

To our best knowledge on previous works concerning BMC application to timed automata (based on either discretization or general polyhedra/zones), our approach is the first that uses region graphs. One obvious drawback with region graphs is their prohibitive size, which grows exponentially with the number of clocks and with each clock’s maximal constant. In particular, when applying standard model checking techniques to region graphs, verification becomes infeasible even for moderately-sized systems. However, our results appear promising and shows that region graph encoding may be feasible in practice because: a) using regions (as opposed to general zones/polyhedra) implies simple transition relations, and b) SAT-based BMC is applicable to very large systems and is efficient when the transition relations are not too complex.

## 8. Conclusion and Future Work

It is well known BMC is more efficient in identifying bugs and verifying systems with a large number of program variables. However, it is difficult for BMC to model timing behavior. To address this problem, we used a robust method to explicitly encode regions, reducing the reachability problems of dense-time systems to satisfiability problems. The results of our experiments indicate that even without enhancements (e.g., symmetry reduction, forward projection, and abstraction), our region encoding is more efficient in verifying timing behavior of large systems when compared with other well-developed tools. However, as with all bounded model checkers, xBMC is not as effective as the other tools for guaranteeing correctness. Therefore, one of our follow-up works [29] is to apply ground decision procedures based on induction to support complete computations without threshold requirements. Other future work includes: a) applying enhancements to further improve efficiency, and b) to integrate xBMC with WebSSARI [13] in an effort to verify the timing behavior of real-world Web applications.

## References

[1] Alur, R., Courcoubetis, C. and Dill, D.,"Model-checking for Real-time Systems." In IEEE 5th Annual Symposium on Logic In Computer Science (LICS), Philadelphia, June 1990.

[2] Alur, R. and D. L. Dill, "A Theory of Timed Automata." Theoretical Computer Science 126, pp. 183-235, 1994

[3] Amla, N., Kurshan, R., McMillan, K. and Medel, R. K., "Experimental Analysis of Different Techniques for Bounded Model Checking." In Proc. TACAS'03, LNCS, Poland, 2003.

[4] Asarin, E., Bozga, M., Kerbrat, A., Maler, O., Pnueli, A. and Rasse, A., "Data-structures for the Verification of Timed Automata." In Proc. HART'97, LNCS 1201, pp. 346-360, 1997.

[5] Audemard, G., Cimatti, A., Korniowicz, A. and Sebastiani, R., "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, 2002.

[6] Beer, I., Ben-David, S. and Landver, A., "On-the Fly Model Checking of RCTL Formulas." In Proc. of the 10th CAV, LNCS 818, 1998.

[7] Biere, A., Cimatti, A., Clarke, E.M., Fujita, M., Zhu, Y.,"Symbolic Model Checking Using SAT Procedures Instead of BDDs." In Proc. DAC'99, pp. 317-320, 1999.

[8] Bozga, M., Daws, C., Maler, O., Olivero, A., Tripakis, S. and Yovine, S., "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.

[9] Clarke, E., Biere, A., Raimi, R.,and Zhu, Y.,"Bounded Model Checking Using Satisfiability Solving." In Formal Methods in System Design, July 2001.

[10] Clarke, E., Kroening, D., Yorav, K., “Behavioral Consistency of C and Verilog Programs using Bounded Model Checking.” In Proc. DAC’03, Session 23.3, Anaheim, CA, 2003.

[11] Göllü, A., Puri, A., and Varaiya, P., “Discretization of timed automata.” In Proc. of the 33rd IEEE conferene on decision and control, pp. 957-958, 1994.

[12] Henzinger, T.A., Nicollin, X., Sifakis, J. and Yovine, S., “Symbolic Model Checking for Real-Time Systems.” Information and Computation, Vol. 111, pp. 193-244, 1994.

[13] Huang, Y.-W., Yu, F., Hang, C., Tsai, C.-H., Lee, D.-T. and Kuo, S.-Y., “Verifying Web Applications Using Bounded Model Checking.” In Proc. DSN’04, Italy, June, 2004.

[14] Laroussinie, F., Larsen, K. G. and Weise, C., "From timed automata to logic - and back." In Proc. MFCS'95, LNCS 969, pp. 529-539

[15] Larsen, K. G., Pettersson, P., and Wang, Y., ”Compositional and Symbolic Model Checking of Real-time System.” In Proc. RTSS’95, Pisa, Italy, 1995.

[16] Larsen, K. G., Pettersson, P., and Wang, Y., ”UPPAAL in a Nutshell.” In Int. Journal on Software Tools for Technology Transfer 1(1-2), pp. 134-152, 1998.

[17] Lu, Feng, Wnag, Li-C., Cheng, Kwang-Ting, Huan, Ric C-Y, “ A Circuit SAT Solver With Signal Correlation Guided Learning.” In Proc. DATE’03, March, 2003.

[18] Moller, M. O., Rue, H., and Sorea, M., “Predicate Abstraction for Dense Real-time Systems.” in Theory and Practice of Timed Systems (TPTS'2002), 2002.

[19] Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L. and Malik, S., “Chaff: Engineering an Efficient SAT Solver.” In Proc. DAC'01, June 2001.

[20] de Moura, L., Rueß, H. and Sorea, M., ”Lazy Theorem Proving for Bounded Model Checking over Infinite Domains.” In Proc. CADE’02, LNCS 2392, pp. 438-455, 2002.

[21] de Moura, L., Owre, S., Rueß, H., Rushby, J., Shanker, N. and Sorea, M., “SAL 2.” In Proc. CAV'04, 2004.

[22] Niebert, P., Mahfoudh, M., Asarin, E., Bozga, M., Jain, N., and Maler, O., ”Verification of Timed Automata via Satisfiability Checking.” In Proc. FTRTFT'02, LNCS 2469, 2002.

[23] Penczek, W., Wozna, B. and Zbrzezny, A., “Towards Bounded Model Checking for the Universal Fragment of TCTL.” In Proc. FTRTFT’02, LNCS 2469, pp. 265-288, 2002.

[24] Seshia, S. A., Bryant, R. E., “Unbounded, Fully Symbolic Model Checking of Timed Automata using Boolean Methods.” In Proc. CAV'03. LNCS 2725, Springer-Verlag, 2003.

[25] Sorea, M.,”Bounded Model Checking for Timed Automata.” CSL Technical Report SRI-CSL-02-03, 2002.

[26] Wang, F., “Efficient Verification of Timed Automata with BDD-like Data-Structures.” In Proc. VMCAI’03, LNCS 2575, Springer-Verlag, 2003.

[27] Wozna, B., Penczek, W. and Zbrzezny, A., “Checking Reachability Properties for Timed Automata via SAT.” Fundamenta Informaticae, Vol. 55(2), pp. 223-241, 2003.

[28] Yovine, S., “Model-checking Timed Automata.” Embedded Systems, LNCS 1494, 1998.

[29] Yu, F. and Wang, B.-Y. “Toward Unbounded Model Checking for Region Automata.” Paper Submitted.