Toward Unbounded Model Checking for Region Automata*

Fang Yu and Bow-Yaw Wang

Institute of Information Science, Academia Sinica
Taipei 115, Taiwan, Republic of China
{yuf, bywang}@iis.sinica.edu.tw

Abstract. The large number of program variables in a software verification model often makes model checkers ineffective. Since the performance of BDD's is very sensitive to the number of variables, BDD-based model checking is deficient in this regard. SAT-based model checking shows some promise because the performance of SAT-solvers is less dependent on the number of variables. As a result, SAT-based techniques often outperform BDD-based techniques in discrete systems with a lot of variables. Timed systems, however, have not been as thoroughly investigated as discrete systems. The performance of SAT-based model checking in analyzing timing behavior – an essential task for verifying real-time systems – is not so clear. Moreover, although SAT-based model checking may be useful in bug hunting, their capability in proving properties has often been criticized. To address these issues, we propose a new bounded model checker, $xBMC$, to solve the reachability problem of dense-time systems. In xBMC, regions and transition relations are represented as Boolean formulae via discrete interpretations. To support both property refutation and verification, a complete inductive algorithm is deployed, in addition to the requirement of reaching an intrinsic threshold, i.e. the number of regions. In an experiment to verify the client authentication protocol of Cornell Single Sign-on systems, xBMC outperforms the efficient model checker, RED [35], even if no bugs exist. We believe that xBMC may provide an effective and practical method for timing behavior verification of large systems.

Keywords: Induction, Verification, Model checking, Region automata , Real-time systems, BMC.

1 Introduction

Successful model checking for software verification mandates the ability to handle a large number of program variables. Because the size of binary decision diagrams (BDDs) may grow rapidly as the number of variables increases, software verification remains a difficult problem for conventional BDD-based model

* This research is partially supported by NSC project 93-2213-E-001-012-.

checkers. On the other hand, satisfiability (SAT) solvers are less sensitive to the number of variables and SAT-based model checking, i.e. bounded model checking (BMC), is showing some promise in this regard [9][13]. As Nierbert et al. [30] suggested, BMC has benefits in terms of bug hunting, especially for systems too large for complete verification, even though it is less efficient in guaranteeing the correctness of software systems.

A recent comparison [5] of the two techniques shows that BDD-based model checkers require more space, but SAT-based model checkers require more time. As a result of numerous proposals for improving the efficiency of SAT solvers [23][26], the performance of BMC's has improved automatically. Consequently, BMC has recently gained acceptance in the research community, especially for software verification [14][18]. However, for the analysis of timing behavior, which is considered essential for verifying embedded systems, protocol implementations and many other types of software, the advantages of SAT-based BMC are less clear. A fundamental problem with BMC is its lack of support for timing behavior modeling. We have addressed this issue in [38], where we applied BMC techniques to region automata and encoded the implicit simulation of a region-based state exploration algorithm as Boolean formulae. In that project we not only characterized regions as combinations of discrete interpretations, but also precisely encoded the settings of these interpretations as Boolean formulae. We proved that the satisfiability of these Boolean formulae is equivalent to solving the forward reachability problem of dense-time systems, within steps bounded by the number of regions.

Although SAT-based verification techniques are very useful in bug hunting, their capability in proving properties has often been criticized. An inductive method offers SAT-based verification an opportunity to prove safety properties efficiently. The basic idea, just like mathematical induction, is to prove the safety property for all steps by assuming the properties of the previous steps. Previous research has been devoted to this issue [10][15][28][33], but none of it supports timing behavior.

In this paper, we extend the above research to timed systems. By applying a loop-free inductive method to BMC, we implement xBMC for inductive reachability analysis of region automata. 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. Compared to research on the encoding of timing behavior [7][25][29][31][31], discretization in [38] allows us to deploy the inductive method rather straightforwardly. Compared to conventional model checkers [11][22][35], we provide a more effective and practical method to alleviate state explosion, especially for those systems too large to verify.

Our experiments verify the correctness of the client authentication protocol of Cornell Single Sign-on systems(CorSSO) [19]. The experimental results show that xBMC is more efficient than RED [35] for correctness guarantee, as well as for bug hunting.

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 our previous effort that encodes the behavior of region automata as Boolean formulae. An inductive algorithm is given in Section 4. An inductive reachability analysis is given in Section 5, and experimental results are summarized in Section 6. After discussing related works in Section 7, we present our conclusions in Section 8.

2 Timed Automata

A timed automaton (TA) [1][2][4][37] is an automaton with a finite set of clock variables. Its behavior consists of a) alternating discrete transitions that are constrained by guarded conditions on discrete and clock variables and b) time passage in which the automaton remains in one state, while clock values increase at a uniform rate. For clarification of discretization purposes, we use a TA that contains both discrete and clock variables, rather than one that models 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 φ is defined by: $\varphi := ff \,|\, d = q | x \triangleleft c | \neg\varphi | \varphi_1 \vee \varphi_2$, where $d \in D$ and $q \in \mathbf{dom}(d)$, $x \in X$, $\triangleleft \in \{<, =, \leq\}$, and $c \in \mathbb{N}$ is a non-negative integer. Typical short forms are: $tt \equiv \neg ff$, $\varphi_1 \wedge \varphi_2 \equiv \neg((\neg\varphi_1) \vee (\neg\varphi_2))$ and $\varphi_1 \rightarrow \varphi_2 \equiv \neg\varphi_1 \vee \varphi_2$. A discrete interpretation s assigns each discrete variable a non-negative integer that represents one value from its predefined domain, i.e. $s : D \mapsto \mathbb{N}$. A clock interpretation ν assigns a non-negative real value to each clock, i.e. $\nu : X \mapsto \mathbb{R}^+$. We say that an interpretation pair (s,ν) satisfies constraint φ if and only if φ is evaluated as true, according to the values given by (s,ν).

2.2 Timed 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 $\mathbf{dom}(d)$,
2. X is a finite set of clock variables,
3. A is an action set with each $\tau \in A$ consisting of a finite series of discrete variable assignments,
4. I specifies an initial condition, and
5. $E \subseteq \Phi(D, X) \times A \times 2^X$ is a finite set of edges. An edge $e : \langle \varphi, \tau, \lambda \rangle \in E$ represents the transition consisting of:$\varphi \in \Phi(D, X)$ as a triggering condition which specifies when the transition can be fired, $\tau \in A$ as the action that changes the current discrete interpretation into the next one, and $\lambda \subseteq X$ as the set of reset clocks.

For an action , $s[\tau]$ denotes the discrete interpretation after applying $\tau \in A$ to s. For $\delta \in \mathbb{R}^+$, $\nu + \delta$ 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 ν over the rest of the clocks. The essence of a TA is a transition system $\langle Q, \rightarrow \rangle$, where Q is the set of states and $\rightarrow$ is the transition relation. A *state* of a TA is a pair (s, ν) such that s is a discrete interpretation of D and ν is a clock interpretation of X. We say (s, ν) is an *initial* state, where s maps discrete variables to values that satisfy I and $\nu(x) = 0$ for all $x \in X$. There are two types of $\rightarrow$, i.e. $\stackrel{\delta}{\rightarrow}$ and $\stackrel{e}{\rightarrow}$:

1. For a state (s, ν) and an increment $\delta \in \mathbb{R}^+$, $(s, \nu) \stackrel{\delta}{\rightarrow} (s, \nu + \delta)$.
2. For a state (s, ν) and an edge $e : \langle \varphi, \tau, \lambda \rangle$ such that (s, ν) satisfies φ, $(s, \nu) \stackrel{e}{\rightarrow} (s[\tau], \nu[\lambda])$.

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

3 Boolean Encoding of Region Automata

System states change as time progresses, but some changed states are not distinguishable by constraints. Based on this observation, Alur et al. [2] defined the equivalence of clock interpretations and proposed region graphs for the verification of timed automata. We represent the set of clock assignments of an equivalence class as (ν_d, ν_γ), a pair of discrete interpretations mapping integral parts and fraction orderings of clock assignments respectively. Given an equivalence class $[\nu]$, integral parts of the clock assignments stand for the discrete interpretation ν_d in (1), which maps each clock $x \in X$, assuming $\nu(x) = t$ and $t = \lfloor t \rfloor + frac(t)$, into an integer representing an interval from $\{[0,0], (0,1), [1,1], \cdots, (c_x - 1, c_x), [c_x, c_x], (c_x, \infty)\}$.

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

Given a discrete interpretation ν_d, we define $\mathtt{Inv}(\nu_d) : \{x | \exists k \in \mathbb{N}, k < c_x, \nu_d(x) = 2k+1\}$ as the set of clocks having non-zero fractions. Then, the discrete interpretation ν_γ in (2) maps each clock pair (x, y), where $x, y \in \mathtt{Inv}(\nu_d)$ and $x < y$, into a relation from $\{\prec, \succ, \approx\}$. Note that ν_γ stands for the fraction ordering of an equivalence class $[\nu]$.

$$\nu_\gamma(x,y) = \begin{cases} \prec, \texttt{if } frac(\nu(x)) < frac(\nu(y)) \\ \succ, \texttt{if } frac(\nu(x)) > frac(\nu(y)) \\ \approx, \texttt{if } frac(\nu(x)) = frac(\nu(y)) \end{cases} \tag{2}$$

It's clear that (ν_d, ν_γ) exactly represents $[\nu]$, while ν_d and ν_γ are defined in (1) and (2) respectively. For example, an equivalence class $(1 < x < y < 2) \wedge z = 1$ is represented by the pair (ν_d, ν_γ), where $\nu_d(x) = 3 \wedge \nu_d(y) = 3 \wedge \nu_d(z) = 2 \wedge \nu_\gamma(x,y) = \prec$. Accordingly, a *region*, $(s,[\nu])$, can be precisely represented as an *interpretation* state, (s, ν_d, ν_γ), where three discrete interpretations $s : D \mapsto \mathbb{N}$, $\nu_d : X \mapsto \mathbb{N}$ and $\nu_\gamma : \mathrm{Inv}(\nu_d) \times \mathrm{Inv}(\nu_d) \mapsto \{\prec, \succ, \approx\}$ are involved.

A TA's transition system then can be represented by a finite discrete interpretation graph $\left\langle Q_{\cong}, \overset{\cong}{\rightarrow} \right\rangle$, where $Q_{\cong}$ is the set of interpretation states and $\overset{\cong}{\rightarrow}$ is the interpretation transition relation. There are two types of $\overset{\cong}{\rightarrow}$, i.e. $\overset{\delta_{\cong}}{\rightarrow}$ and $\overset{e_{\cong}}{\rightarrow}$:

1. Given a state (s, ν_d, ν_γ), $(s, \nu_d, \nu_\gamma) \overset{\delta_{\cong}}{\rightarrow} \left(s', \nu'_d, \nu'_\gamma\right)$ if and only if ψ_{time} is evaluated as true, according to the values given by $\left(s, \nu_d, \nu_\gamma, s', \nu'_d, \nu'_\gamma\right)$.
2. Given a state (s, ν_d, ν_γ) and an edge $e : \langle \varphi, \tau, \lambda \rangle$, $(s, \nu_d, \nu_\gamma) \overset{e_{\cong}}{\rightarrow} \left(s', \nu'_d, \nu'_\gamma\right)$ if and only if ψ_{tran} is evaluated as true, according to the values given by $\left(s, \nu_d, \nu_\gamma, s', \nu'_d, \nu'_\gamma\right)$.

ψ_{time} defines the successor relation formula for capturing a region moving into a subsequent region due to time passage, while ψ_{tran} defines the discrete transition formula for triggering an edge using discrete interpretations. Then the transition relation formula T is $\psi_{tran} \vee \psi_{tran}$. In [38], we proposed formal definitions and encoding methods of these formulae, and proved that these encodings represent the exact behavior of region automata.

Our state variables B are given in (3), 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 $\sum \lceil \lg |\mathbf{dom}(d)| \rceil + \sum \lceil \lg (2c_x + 2) \rceil + |X|\,|X-1|$. To perform BMC, we add a copy of B to the set of state variables at each iteration.

$$\begin{aligned} B = & \{b_{d_k} | d \in D, 0 \le k \le \lceil \lg |\mathbf{dom}(d)| \rceil\} \cup \\ & \{b_{x_k} | x \in X, 0 \le k \le \lceil \lg (2c_x + 2) \rceil\} \cup \\ & \left\{b_{xy_k} | x, y \in X, x < y, k \in \{0,1\}\right\} \end{aligned} \tag{3}$$

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 Induction

Although SAT-based model checking is very useful for bug hunting [9][13][38], 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 prove the safety property for all steps by assuming the property on previous steps. Here, we briefly illustrate the technique. Interested readers are referred to [10][15][28][33] 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 $\texttt{Reach}(q_0)$. 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 the 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. In the literature, induction with arbitrary depths has been proposed. Unfortunately, the inductive technique cannot prove all safety properties, even with arbitrary depths. In [10][15][28][33], 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. In Figure 1, which shows the flow of the inductive method, B_i represents the state variables of i^{th} step and $P(B)$ is true when the valuation of B satisfies the predicate $P(\bullet)$.

The flow tries to establish the loop-free inductive step within a given bound. The inductive step essentially checks whether it is possible to reach any bad state following several safe states. If it is impossible, we stop; otherwise, the length is increased and the algorithm repeats the loop.

5 Reachability Analysis

In this section, we describe how we deal with the reachability problem by solving the satisfiability problem of an expanding Boolean formula iteratively. Moreover, we show how to apply loop-free induction in BMC efficiently.

Induction(P, MaxBound)
  var i:0..MaxBound;
begin
  i:=0;
  loop forever
    if ($\neg SAT((P(B_0) \wedge (B_0 \rightarrow B_1) \wedge \ldots \wedge P(B_i) \wedge (B_i \rightarrow B_{i+1}) \wedge$
      $\neg P(B_{i+1})) \wedge (\bigwedge_{j<i} B_i \neq B_j))$) return "success";
    if (i=MaxBound) return "fail within MaxBound steps";
    i:=i+1;
end.

Fig. 1. Flow of Inductive Method

5.1 Bounded Reachability Analysis

Given an initial condition, a risk condition, a transition condition and an integer bound, we solve the bounded reachability problem by iteratively calling the SAT solver. We unfold the interpretation transition relation until the SAT solver returns a truth assignment, or reaches the bound. Let $I(B_i)$ and $R(B_i)$ respectively denote the CNF formulae of the given initial and risk conditions over B_i. The implementation of `BoundedFwdReach()` is given in Figure 2. By conjoining the formula with the negation clause of the risk condition, each intermediate result is saved for use in later iterations to speed up the decision procedure of the SAT-solver. In the next section, we also show that this conjunction can be used directly to apply the inductive method.

If the risk state is reachable, the formula will be satisfied at the i^{th} step, and a truth assignment will be returned by the SAT solver. 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

BoundedFwdReach(I, R, MaxBound)
  var i:0..MaxBound;
begin
  i:=0; F:=I(B_0);
  loop forever
    if (SAT(F$\wedge$ R(B_i))) return "reachable";
    if (i=MaxBound) return "unreachable within MaxBound steps";
    F:=F$\wedge\neg$ R(B_i)$\wedge$($B_i \rightarrow B_{i+1}$);
    i:=i+1;
end.

Fig. 2. Bounded Forward Reachability Analysis

5.2 Toward Unbounded 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 Fig. 3, we denote the released formula as F/I, i.e. removing the clauses of I from F. Two extra checks for the loop-free requirement and the induction are used to help determine completeness in early steps. If the loop-free requirement is not satisfied, which means no new states can be reached in the next step, the procedure can terminate and the risk state is unreachable. For the induction, we regard $P(\bullet)$ as $\neg R(\bullet)$, i.e. the negation of the risk condition. Once the induction succeeds, 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 it can terminate with "success" returned. Although regions are finite and we can guarantee success when MaxBound exceeds the number of regions, in the worst case, the inductive method may not determine termination ahead, but only induce overhead. 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 6, we conduct an experiment to show the effectiveness of induction.

IndFwdReach(I, R, MaxBound)
  var i:0..MaxBound;
begin
  i:=0; F:=I(B_0);
  loop forever
    if ($\neg$SAT(F)) return "unreachable by loop-free";
    else if (SAT(F$\wedge$R(B_i))) return "reachable";
    else if ($\neg$SAT((F/I)$\wedge$R(B_i))) return "unreachable by induction";
    else if (i=MaxBound) return "unreachable within MaxBound steps";
    else
      F:=F$\wedge\neg$ R(B_i)$\wedge$($B_i \rightarrow B_{i+1}$)$\wedge$($\bigwedge_{j<i} B_i \neq B_j$);
      i:=i+1;
end.

Fig. 3. Inductive Forward Reachability Analysis

6 Experiment

6.1 Cornell Single Sign-On

Cornell Single Sign-On (CorSSO) [19] is a distributed service for network authentication. It delegates client identity checking to multiple servers by threshold

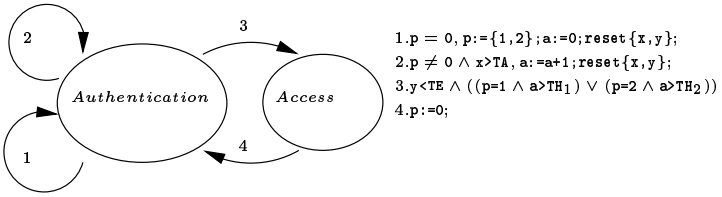

Fig. 4. Each client in the CorSSO protocol has two discrete variables and two clocks: p recording the chosen sub-policy, a for the number of collected authentications, x used to constraint the time spent collecting a certificate, and y used to constraint the expired time of the authorized sub-policy. There is also an auxiliary discrete variable to record locations. Initially, all processes are in *Authentication* with all variable values equaling zero.

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, he has to identify himself 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 he had complied with any sub-policy, i.e. obtaining sufficient certificates from the specified authentication servers within a specified time.

Unlike monolithic authentication schemes where the server is usually overloaded with authentication requests, the authentication policies in CorSSO allow a client to prove her 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 our experiments, we model client behavior. In the simplified client model shown in Figure 4, a client has only two locations: *Authentication* and *Access*. In *Authentication*, he firstly chooses one of the two policies by setting the value of p non-deterministically. If the first policy is chosen, i.e. $p = 1$, he needs to collect more than $\texttt{TH}_1$ certificates from the authentication servers. Similarly, more than $\texttt{TH}_2$ certificates are needed if the second policy is chosen. Then he starts to collect certificates. If he had obtained sufficient certificates within the time limit, he can then move to *Access*. Finally, he can discard the certificates and reset the policy, i.e. $p := 0$, and then return to *Authentication*.

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.

6.2 Experimental Results

We compare the performance of our model checker with RED [35], a full TCTL model checker for timed systems. In the experiments, we first verify the safety property that all clients in *Access* have acquired sufficient certificates necessitated by the chosen policy. Then we implant a bug by mistyping $\mathtt{TH}_2$ for $\mathtt{TH}_1$ in the transition 3 in Figure 4. This may raise a violation against the safety property once $\mathtt{TH}_1 < \mathtt{TH}_2$. Systems with two to eleven clients are checked by both xBMC and RED. It is noted that we did not turn on the symmetry reduction option in RED, even though the systems under test are highly symmetric[1]. Since the present technique does not take symmetry into consideration, we think it would be unfair to compare it with other heuristics. Both RED and xBMC report the safety property is satisfied for normal cases, and the manually-inserted bugs are detected by both tools as expected. The performance results[2] are shown in Table 1. 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[3]. For all bug inserted cases, 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.

	Correctness Guarantee		**Bug Hunting**	
# of clients	*RED 5.0*	*XBMC 2.1*	*RED 5.0*	*XBMC 2.1*
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

Table 1. Experimental Results of xBMC and RED

[1] Symmetry reduction is not activated by default.

[2] All experiments were performed on a Pentium IV 1.7 GHz computer with 256MB of RAM running the Linux operating system.

[3] RED performs much better if symmetry reduction is used. In fact, it outperforms xBMC almost universally with the heuristic.

7 Related Work and Discussion

Due to the success of hardware verification by SAT-based techniques, SAT-based model checking has recently gained considerable attention among software verification researchers. Clarke et al. [14] developed a SAT-based bounded model checker for ANSI C, and the present authors used xBMC to verify Web application code security in an earlier project [18]. Although both projects focused on software verification, neither supported timing behavior analysis.

The verification of timed automata by checking satisfiability has been the topic of several research projects. Most research work encodes the evaluation of atomic constraint to variants of predicate calculus with real variables. Niebert et al. [30] represented the bounded reachability problem in Boolean variables and numerical constraints of Pratt's difference logic, while Audemard et al. [7] took a clock as a real variable, and reduced the bounded verification of timed systems to the satisfiability of a math-formula with linear mathematical relations having real variables. Moura et al. [27] also used real variables to represent infinite state systems. Penczek et al. [31][36] handled timing behavior by discretization, in which they divided each time unit into $2n$ segments (n = number of clocks). Finally, Yu et al. [38] explicitly encoded region automata and proved that the reachability analysis of dense-time systems is equivalent to solving the satisfiability of Boolean formulae iteratively. However, most of these approaches do not emphasize the correctness guarantee.

Some researchers have tried to determine whether iterative satisfiability analysis can terminate early if more restrictive formulae are generated based on satisfiability results from previous iterations. Moura et al. [28] 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. In [34], 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 for termination.

McMillan [24] uses interpolations as an over-approximation of the reachable states. His technique not only verifies the correctness of safety properties, but also guarantees termination. However, it has yet to support timing analysis. Compared to interpolation, where the analysis of internal information in SAT-solvers is required, the inductive method can be implemented by treating SAT-solvers as black boxes. We would like to investigate the merging of interpolations and our approach in the future.

Unlike other reachability analysis techniques for timed automata, discretization allows us to deploy the inductive method rather straightforwardly. However, it is unclear how to apply the same technique in BDD [12], DBM [21] or CRD [35]. It would also be interesting to develop a corresponding inductive method for them and compare their performance with our discretization approach.

8 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 can 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, one limitation of our approach is that the performance depends on whether and when the induction successes.

In the future, we plan to investigate on enhancements to improve the efficiency. Notably, the interpolation technique proposed in [24] is of interest to us. Secondly, we would like to integrate xBMC with WebSSARI [18] in order to verify the timing behavior of real-world Web applications.

References

1. Alur, R., "Timed Automata." In Proc. of CAV'99, LNCS 1633, pp. 8-22, Springer, 1999.
2. 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.
3. Alur, R., Courcoubetis, C. and Dill, D., Halbwachs, N. and H. Wong-Toi, "An Implementation of Three Algorithms for Timing Verification Based on Automata Emptiness." In Proc. of the 13^{th} IEEE Real-Time Systems Symposium, pp. 1557-166, 1992.
4. Alur, R. and D. L. Dill, "A Theory of Timed Automata." Theoretical Computer Science 126, pp. 183-235, 1994.
5. Amla, N., Kurshan, R., McMillan, K. and Medel, R. K., "Experimental Analysis of Different Techniques for Bounded Model Checking." In Proc. of TACAS'03, LNCS, Warsaw, Poland, 2003.
6. Asarin, E., Bozga, M., Kerbrat, A., Maler, O., Pnueli, A. and Rasse, A., "Data-structures for the Verification of Timed Automata." In Proc. of HART'97, LNCS 1201, pp. 346-360, Springer-Verlag, 1997.
7. 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. Springer-Verlag, November 2002.
8. Beer, I., Ben-David, S. and Landver, A., "On-the Fly Model Checking of RCTL Formulas." In Proc. of the 10^{th} CAV, LNCS 818, 1998.
9. Biere, A., Cimatti, A., Clarke, E.M., Fujita, M., Zhu, Y., "Symbolic Model Checking Using SAT Procedures Instead of BDDs." In Proc. of the 36th Design Automation Conference, pp. 317-320, 1999.

10. Bjesse, P. and Claessen, K., "SAT-based verification without state space traversal." In Proc. 3rd Int. Conf. On Formal Methods in Computer Aided Design, LNCS 1954, pp. 372, 2000.
11. 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.
12. Bryant, R. E., "Graph-Based Algorithms for Boolean Function Manipulation." IEEE Trans. Computers 35(8), pp. 677-691, 1986.
13. Clarke, E., Biere, A., Raimi, R.,and Zhu, Y., "Bounded Model Checking Using Satisfiability Solving." In Formal Methods in System Design, July 2001.
14. Clarke, E., Kroening, D., Yorav, K., "Behavioral Consistency of C and Verilog Programs using Bounded Model Checking." In Proc. of the 40th Design Automation Conference, Session 23.3, Anaheim, CA, 2003.
15. Déharbe, D. and Moreira, A., "Symbolic model checking with fewer fixpoint computations." In Proc. of FM '99. LNCS 1708:272-288, 1999.
16. Göllü, A., Puri, A., and Varaiya, P., "Discretization of timed automata." In Proc. of the 33rd IEEE conference on decision and control, pp. 957-958, 1994.
17. 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.
18. Huang, Y.-W., Yu, F., Hang, C., Tsai, C.-H., Lee, D.-T. and Kuo, S.-Y., "Verifying Web Applications Using Bounded Model Checking." To appear: Proceedings of the 2004 International Conference on Dependable Systems and Networks, pages 199-208, Florence, Italy, Jun 28-Jul 1, 2004.
19. Josephson, W., Sirer, E. G and Schneider, F. B., "Peer-to-Peer Authentication with a Distributed Single Sign-On Services." In Proc. of IPTPS04, 2004.
20. Laroussinie, F., Larsen, K. G. and Weise, C., "From timed automata to logic - and back." In Proc. of MFCS'95, LNCS 969, pp. 529-539, 1995.
21. Larsen, K. G., Pettersson, P., and Wang, Y., "Compositional and Symbolic Model Checkingof Real-time System." In Proc. RTSS'95, Pisa, Italy, 1995.
22. 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.
23. Lu, F. Wnag, Li-C., Cheng, Kwang-Ting, Huan, Ric C-Y, "A Circuit SAT Solver with Signal Correlation Guided Learning." In Proc. of DATE'03, March, 2003.
24. McMillan, K. L., "Interpolation and SAT-Based Model Checking." In Proc. of CAV'03, LNCS 2725, pp. 1-13, 2003.
25. Moller, M. O., Rue, H., and Sorea, M., "Predicate Abstraction for Dense Real-time Systems." In Proc. of Theory and Practice of Timed Systems (TPTS'2002), 2002.
26. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L. and Malik, S., "Chaff: Engineering an Efficient SAT Solver." In Proc. of the 38th Design Automation Conference (DAC'01), June 2001.
27. de Moura, L., Rueß, H. and Sorea, M., "Lazy Theorem Proving for Bounded Model Checking over Infinite Domains." In Proc. of CADE'02, LNCS 2392, pp. 438-455, 2002.
28. de Moura, L., Rueß, H. and Sorea, M., "Bounded Model Checking and Induction: from Refutation to Verification." In Proc. of CAV'03, LNCS 2725, pp. 14-26, 2003.
29. de Moura, L., Owre, S., Rueß, H., Rushby, J., Shanker, N. and Sorea, M., "SAL 2", accepted for publication, CAV'2004.
30. Niebert, P., Mahfoudh, M., Asarin, E., Bozga, M., Jain, N., and Maler, O.,"Verification of Timed Automata via Satisfiability Checking." In Proc. of FTRTFT'02, LNCS 2469, pp. 225-244, Springer-Verlag, 2002.

31. Penczek, W., Wozna, B. and Zbrzezny, A., "Towards Bounded Model Checking for the Universal Fragment of TCTL." In Proc. of FTRTFT'02, LNCS 2469, pp. 265-288, Springer-Verlag, 2002.
32. Seshia, S. A. and Bryant, R., "Unbounded, Fully Symbolic Model Checking of Timed Automata using Boolean Methods." In Proc. of CAV'03. LNCS 2725, Springer-Verlag, 2003.
33. Sheeran, M., Singh, S. and Stålmarck, G., "Checking Safety Properties Using Induction and a SAT-solver." In Proc. of FMCAD 2000. LNCS, 1954:108, 2000.
34. Sorea, M., "Bounded Model Checking for Timed Automata." CSL Technical Report SRI-CSL-02-03, 2002.
35. Wang, F., "Efficient Verification of Timed Automata with BDD-like Data-Structures." VMCAI'03, LNCS 2575, Springer-Verlag, 2003.
36. Wozna, B., Penczek, W. and Zbrzezny, A., "Checking Reachability Properties for Timed Automata via SAT." Fundamenta Informaticae, Vol. 55(2), pp. 223-241, 2003.
37. Yovine, S., "Model-checking Timed Automata." Embedded Systems, LNCS 1494, pp. 114-152, 1998.
38. Yu, F., Wang, B.-Y. and Huang, Y.-W., "Bounded Model Checking for Region Automata." In Proc. of FORMATS and FTRTFT 2004, Grenoble, France, Sep. 2004.