改进的以 SMT 为基础的实时系统限界模型检测

徐亮

1(中国科学院软件研究所 计算机科学国家重点实验室,北京，100190)
2(中国科学院软件研究所，北京，100190)

Improved SMT-Based Bounded Model Checking for Real-Time Systems

XU Liang1,2*

1(State Key Laboratory of Computer Science, Institute of Software, The Chinese Academy of Sciences, Beijing 100190, China)
2(Institute of Software, The Chinese Academy of Sciences, Beijing 100190, China)
+ Corresponding author: E-mail: xul1981@126.com


Abstract: SAT-Based bounded model checking (BMC) has high complexity in dealing with real-time systems. Satisfiability modulo theories (SMT) solvers can generalize SAT solving by adding the ability to handle arithmetic and other decidable theories. This paper uses SMT in BMC for real-time systems instead of SAT. The clocks can be represented as integer or real variables directly and clock constraints can be represented as linear arithmetic expressions. These make the checking procedure more efficient. TCTL (timed computation tree logic) is used to specify the properties of real-time systems and improvement of the encodings has been done.

Key words: bounded model checking; satisfiability modulo theories; real-time system; timed automata; timed Kripke structure; TCTL (timed computation tree logic)

摘 要: 基于 SAT 的限界模型检测在处理实时系统时具有很高的复杂度。SMT 求解器在计算可满足性的同时，还能处理算术和其他可判定性理论。在对实时系统进行检测时，用 SMT 求解器代替 SAT 求解器，系统里的时钟就可以用整型或实型变量表示，时钟约束则可以直接表示成线性算术表达式，从而使整个检测过程更加高效。带时间参数的计算树逻辑 (timed computation tree logic, 简称 TCTL) 被用来描述实时系统里的性质，同时还对检测方法作了相应的改进。

关键词: 限界模型检测; 可满足性模块理论; 实时系统; 时间自动机; 时间 Kripke 结构; 带时间参数的计算树逻辑

中图法分类号: TP301 文献标识码: A

∗ Supported by the National Natural Science Foundation of China under Grant Nos.60721061, 60833001 (国家自然科学基金); the CAS Innovation Program of China (中国科学院知识创新工程领域前沿项目); the Knowledge Innovation Key Directional Program of the Chinese Academy of Sciences under Grant No.KGCX2-YW-125 (中国科学院知识创新工程重要方向项目)

Received 2008-12-12; Accepted 2010-01-25
1 Introduction

In symbolic model checking, Binary Decision Diagrams (BDD’s)\cite{1} are used to represent system states. The technique is used in many model checking tools, such as NuSMV\cite{2} and TSMV\cite{3}. But the size of BDD may grow significantly as the number of variables increase. Checking a system with a large number of variables remains a difficult problem for BDD-based model checking tools. On the other hand, Bounded Model Checking (BMC) based on Boolean Satisfiability (SAT) has been introduced as a complementary technique to BDD’s model checking\cite{4-6} for combating the state explosion problem. The basic idea of BMC presented in Refs.\cite{4-6} is to restrict the general model checking problem to a bounded one. Instead of finding out whether the system $M$ violates the property $\psi$, we only need to know whether the system $M$ has some counterexamples of length $k$ to $\psi$. The problem is encoded into a propositional formula, and then a SAT solver is used to check the formula, in order to see whether it is satisfiable. The method’s efficiency is based upon the fact that if there is a counterexample, then it may be found only in a small portion of its state space\cite{4,5}.

In recent years, people’s interest in automated verification has shifted to real-time systems. The verification of real-time systems becomes a very important and challenging problem. References \cite{7,8} transform the Timed Automaton (TA) to a Region Graph (RG)\cite{9}, which is based on dense-time approach\cite{10-12}, then encode a Timed CTL (TCTL)\cite{9} formula to a propositional formula and use SAT solvers to check it. TSMV\cite{3}, improved from NuSMV for verifying timed systems, uses Timed Kripke Structure (TKS) as its model, which is based on discrete-time approach\cite{13}, and solves problems with BDD-based method. Either SAT-based or BDD-based method for real-time systems, needs to encode clocks and clock constraints into boolean formulae. After this encoding, the clocks’ characteristics disappear which means the whole checking process has no time information to use. In order to overcome this disadvantage, we use Satisfiability Modulo Theories (SMT) instead of SAT to do the check.

The SMT problem is a generalization of the SAT problem where Boolean variables are replaced by predicates from various background theories, such as linear real and integer arithmetic. So, we can use real or integer variables to represent clocks and linear arithmetic expressions to represent clock constraints instead of Boolean formulae, which preserve the time characteristics in the checking process. There are some related works for SMT-based BMC, such as CBMC\cite{14} which is dealing with programming languages and SA1\cite{15} which can check finite state systems based on SAT solvers and check infinite state systems based on SMT solvers. By doing so, the encoding of clock variables and clock constraints has less effect on BMC’s efficiency. A preliminary version of SMT-based BMC for real-time systems is mentioned in Ref.\cite{16} and the improvement to simplify the encodings in order to improve its efficiency is given in this paper.

The rest of this paper is organized as follows. Some preliminaries are introduced in the next section. The SMT-based BMC approach and the improvement are given in Section 3. The experimental results are summarized in Section 4. Concluding remarks are given in Section 5.

2 Preliminaries

Real-Time systems do not only contain discrete variables but also have dense-time clocks which have a real domain and continuously increase at a uniform rate. Clocks are usually set to zero at the beginning and can be reset at any time. Real-time systems can often be modeled as a TA, which is a popular approach and several model checkers exist (e.g. Refs.\cite{17,18}), or simply modeled as a TKS, which combines simplicity and efficiency (e.g. Ref.\cite{3}).
2.1 Timed automata

A TA is a finite-state machine equipped with a set of clocks. Hereafter, the set \( \mathcal{P} = \{ p_1, p_2, \ldots \} \) denotes atomic propositions, \( \mathcal{N} \) denotes the set of natural numbers, \( \mathcal{N}^* \) denotes the set of \( \{1,2,\ldots\} \), \( \mathbb{Z} \) denotes the set of integer numbers, \( \mathbb{R} \) denotes the set of real numbers and \( \mathbb{R}^+ \) denotes the set of non-negative real numbers.

Let \( \mathcal{X} \) be a finite set of variables called clocks. A clock valuation is a function \( v: \mathcal{X} \rightarrow \mathbb{R}^+ \), which assigns a non-negative real number \( v(x) \) to each clock \( x \in \mathcal{X} \). For a subset \( \mathcal{I} \) of \( \mathcal{X} \) by \( v[\mathcal{I}] = 0 \), we mean the valuation \( v' \) such that \( \forall x \in \mathcal{X}, v'(x) = 0 \) and \( \forall x \in \mathcal{I}, v'(x) = v(x) \). For \( \delta \in \mathbb{R}^+ \), \( v + \delta \) denotes the valuation \( v'' \) such that \( \forall x \in \mathcal{X}, v''(x) = v(x) + \delta \). The set \( \Psi_\mathcal{X} \) of clock constraints over the set of clocks \( \mathcal{X} \) is defined as follows:

\[
s_x := x < \epsilon \lor x = 0 \lor \lnot s_y \lor s_x \wedge s_y \rightarrow \lnot s_x,
\]

where \( x, x' \in \mathcal{X} \), \( \epsilon \in \{<, \leq, =, \geq, >\} \) and \( c \in \mathcal{N} \).

A clock valuation \( v \in \Psi_\mathcal{X} \) in which \( \Psi_\mathcal{X} \) denotes the set of all valuations, satisfies the clock constraint \( \psi \in \Psi_\mathcal{X} \) denotes:

\[
\begin{align*}
v &\models x < c & \text{iff } v(x) < c; \\
v &\models x - x' < c & \text{iff } v(x) - v(x') < c; \\
v &\models v \wedge \psi & \text{iff } v = v \land v = \psi; \\
v &\models \psi & \text{iff } v \models \psi.
\end{align*}
\]

We denote by \( [[\psi]] \) the set of valuations that satisfy \( \psi \), that is:

\[
[[\psi]] = \{ x \in \Psi_\mathcal{X} \mid v \models \psi \}.
\]

**Definition 1.** A TA is a tuple \( (\mathcal{S}, \mathcal{X}, S_0, \epsilon, \delta) \) where:

- \( \mathcal{S} \) is a finite set of locations.
- \( \mathcal{X} \) is a finite set of clocks.
- \( \Sigma \) is a finite set of labels.
- \( s_0 \in \mathcal{S} \) is an initial location.
- \( \epsilon \) is a finite set of transition relations, \( \epsilon = \mathcal{S} \times \Sigma \times \mathcal{X} \times 2^\mathcal{X} \times \mathcal{S} \).
- \( \delta: \mathcal{S} \rightarrow \Psi_\mathcal{X} \) is a state invariant function.

Each element \( e \in \epsilon \) is denoted by \( e := s \xrightarrow{\epsilon \in \epsilon} s' \). This represents a transition from location \( s \) to location \( s' \) on the input label \( \epsilon \in \Sigma \). The set \( \Psi_\mathcal{X} \) gives the clocks to be reset with this transition. \( \psi \in \Psi_\mathcal{X} \) is the enabling condition for \( e \). Figure 1 is a simple example of TA. Let \( c_{\text{max}} \) be the largest constant appearing in \( \Psi_\mathcal{X} \). For \( x \in \mathcal{X} \), \( \text{frac}(v(x)) \) denotes the fractional part of \( v(x) \), and \( \lceil v(x) \rceil \) denotes its integral part.

![Fig. 1 A simple example of TA](image_url)

**Definition 2.** For two clock valuations \( v \) and \( v' \), \( v =_{\psi_\mathcal{X}} v' \) if and only if for all \( x, y \in \mathcal{X} \) follows the conditions below:
We use \([v]\) to denote the equivalence class of the relation \(\equiv_{p_s}\) to which \(v\) belongs. Such a class is called a zone. The set of all the zones is denoted by \(\mathcal{Z}(n)\). A zone \([v]\) is final iff \([v]\supseteq c_{\text{max}}\) for all \(v\in[v]\) and \(x\in X\). A zone \([v]\) satisfies the clock constraint \(\psi \in \Psi_X\), if \([v]\supseteq B\psi\) iff \(\forall v'\in[v], v'\supseteq B\psi\),

**Definition 3.** The RG of a TA is a finite structure \(\langle Q, q_0, \rightarrow, L \rangle \):  
- \(Q\) is the set of states.
- \(q_0\in Q\) is the initial state.
- \(\rightarrow\) is defined as follows:
  - \((s, [v]) \rightarrow (s', [v'])\) iff \(\forall e: s \rightarrow L e \rightarrow s'\in \varepsilon\) such that \(s = \text{source}(e), s' = \text{target}(e), [v'] = [v][\Psi = 0]\) and \(v'\supseteq I(s')\).
  - \(L: S \rightarrow 2^\text{AP}\) is a labeling function that maps each state of \(S\) to a set of atomic propositions true in that state.

A state \(q\) is deadlock if there is no delay \(\delta \in \mathbb{N}^+\) and an action \(l\in\Sigma\) such that \(q\rightarrow l q' \rightarrow q''\), for some \(q', q''\in \mathcal{L}\). For simplicity of presentation, we consider only progressive TA\(^7\).

### 2.2 Timed Kripke structure

TKS is an extension of Kripke Structure (KS) where each transition is labeled by a nonnegative integer.

**Definition 4.** A TKS is a tuple \(\langle S, s_0, R, L \rangle \) where  
- \(S\) is a finite set of states.
- \(s_0 \in S\) is an initial state.
- \(R \subseteq S \times \mathbb{N} \times S\) is a finite set of transitions labeled by a natural number, called the duration of the transition.
- \(L: S \rightarrow 2^\text{AP}\) is a labeling function that maps each state of \(S\) to a set of atomic propositions true in that state.

A path \(\pi\) in TKS is an infinite sequence \(s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \ldots\), where \(s_n, s_{n+1}, \ldots \in S\). For each path, and for \(n\in\mathbb{N}^+_0\), \(\pi(n)\) denotes the \(n\)th state \(s_n\), and \(\pi^n\) denotes the \(n\)th suffix \(s_n \rightarrow s_{n+1} \rightarrow \ldots\). Finally, for \(n\leq m\in\mathbb{N}^+_0\), \(\pi[n \ldots m]\) denotes the finite sequence \(s_n \rightarrow s_{n+1} \rightarrow \ldots \rightarrow s_m\) with \(m-n\) transitions and \(m-n+1\) states. The duration \(D_{\pi[n \ldots m]}\) of such a finite sequence is \(d_n + d_{n+1} + \ldots + d_m\) \((D_n\) when \(m=n\)). Fig.2 is a simple example of TKS.

![Fig.2 A simple example of TKS](image)

### 2.3 TECTL

*Computation tree logic (CTL)* is a propositional branching-time temporal logic introduced by Ref.[19] as a
specification language for finite state systems. TECTL\cite{9} is ECTL with timing constraints. TECTL formulae are built according to the following syntax:

\[ p \in \mathcal{AP} \quad | \quad \alpha \land \beta \quad | \quad \alpha \lor \beta \quad | \quad EF_\varphi \quad | \quad E (a U_\varphi) \]

where \( p \in \mathcal{AP} \) and \( \varphi \) is an interval in \( \mathbb{N} \) with integer bounds of the form \([n, m],[n, m],[n, m],[n, m],[n, \infty],[n, \infty],[n, \infty],[n, \infty] \), for \( n, m \in \mathbb{N} \), or \( \varphi \) called the timing constraint, is a predicate on durations. Typically, for all constraints of the form \( \leq n \), \( = n \), \( \geq n \), \( n \) is in the relevant domain (here \( \mathbb{N} \)). Let \( I_\varphi \) be the predicate that the timing constraint \( \varphi \) is satisfied and \( I^i_\varphi \) be the predicate that state \( i \) satisfies the timing constraint \( \varphi \) where \( i \in \mathbb{N} \).

Here, we give a translation of a TECTL formula \( \psi \) to an ECTL formula \( \psi' \) as follows:

\[ p \in \mathcal{AP} \quad \rightarrow \quad \neg \alpha \quad \land \quad \alpha \land \beta \quad \lor \quad \alpha \lor \beta \quad EF_\alpha \quad \rightarrow \quad E (a U_\varphi) \]

\[ p \in \mathcal{AP} \quad \rightarrow \quad \neg \alpha \quad \land \quad \alpha \land \beta \quad \lor \quad \alpha \lor \beta \quad EF_\alpha \quad \rightarrow \quad E (a U_\varphi) \]

**Definition 5 (semantics).** Let \( s_i, p \in \mathcal{S}, p \in \mathcal{AP} \). The semantics of TECTL formulae are defined as follows:

- \( s_i \models p \iff p \) is satisfied in state \( s_i \)
- \( s_i \models \neg \alpha \iff s \not\models \alpha \)
- \( s_i \models \alpha \land \beta \iff (s_i \models \alpha) \land (s_i \models \beta) \)
- \( s_i \models \alpha \lor \beta \iff (s_i \models \alpha) \lor (s_i \models \beta) \)
- \( s_i \models EF_\alpha \iff \exists \pi . (s_0 = s_i \land \exists k \geq 0 (s_k \models I^k_\varphi \land \alpha)) \)
- \( s_i \models EG_\alpha \iff \exists \pi . (s_0 = s_i \land \forall k \geq 0 (s_k \models I^k_\varphi \rightarrow \alpha)) \)
- \( s_i \models E (a U_\varphi) \iff \exists \pi . (s_0 = s_i \land \exists k \geq 0 ((s_k \models I^k_\varphi \land \beta) \land 0 \leq j < k (s_j \models \alpha))) \)

As an example, the TECTL formula \( EF_\alpha \) is true in state \( s_i \) means that there is a path \( \pi \) with its first state equal to \( s_i \) and can reach a state which satisfies \( \alpha \) and the timing constraint \( \varphi \).

### 2.4 Satisfiability modulo theories

Recent SMT solvers\cite{20-22} closely integrate theory-specific solvers with a Davis-Putnam-Loveland-Loveland (DPLL) approach to SAT. These types of SMT solvers are often referred to as DPLL(T)\cite{22}. In this type of architecture, the DPLL-based SAT solver passes conjunctions of predicates belonging to theory \( T \), such as linear real and integer arithmetic, uninterpreted functions, and the theories of various data structures\cite{22-24}, to a specialized solver. The specialized solver is then responsible for deciding feasibility of those predicates.

A Satisfiability Modulo Theories Library (SMT-LIB) benchmark declaration is as follows:

```
benchmark b begin
  logic = L
  assumption = \varphi_1
  \vdots
  assumption = \varphi_n
  formula = \varphi
  status = \sigma
end
```

where the value of \textbf{logic} coincides with the name of a logic \( L \) for some logic in SMT-LIB. The formulae in the \textbf{assumption} and \textbf{formula} are in the language of \( L \) and they together constitute the benchmark. The attribute \textbf{status}
declaration records whether the benchmark’s formula is known to be satisfiable in the associated background theory.

For example, in checking real-time systems, the tools need to support linear real or integer arithmetic. The SMT-LIB theory of Ints or Reals or Reals_Ints is needed as its background theory. By using SAT solvers, multiple Boolean variables are used as a bit representation for integers and the necessary integer theories are specified as Boolean operations on those individual bit variables. This can result in extremely large SAT instances. By using SMT solvers, with logic=QF_LRA or QF_LIA, real or integer variables, linear arithmetic and Boolean combinations of inequations between linear polynomials over real or integer variables can be used in assumption and formula. This preserves the time information in the system and with simplified instances.

Yices’20, the winner of the 2006 SMT competition, includes an incremental simplex algorithm for the theory of linear arithmetic that is tightly integrated within the DPLL framework. Yices’ great ability to work with the theory of linear arithmetic made it particularly well suited for real-time systems model checking. For these reasons, we use Yices as the SMT solver in this paper.

3 Improved SMT-Based Bounded Model Checking

3.1 Bounded semantics of TECTL

In order to define the bounded semantics of TECTL and the encodings of BMC to a SMT problem, we have to give some notions. Let \( M=\langle \mathcal{S}, T, s_0, \mathcal{A} \rangle \) be a model with different semantics in different models. If TA is used as the model, we have to transform the TA to a corresponding RG. Then the elements in \( M \) stand for the corresponding ones in Definition 3 and \( \text{dead}(w_i) \) means the state \( w_i \) is a deadlock. (If the model is represented by TKS, and the elements in \( M \) stand for the corresponding ones in Definition 4, then \( \text{dead}(w_i) \) is set true if \( D_0 [0, k] \geq n, k \in \mathbb{N} \), and the timing constraint is in the form of "\( \leq n \)" or "\( = n \)", and else \( \text{dead}(w_i) \) is set false.) A k-path of \( M \) is a finite sequence \( s_0, \ldots, w_i \) of states such that \( (w_i, w_i+1) \in T \) for \( i=0, \ldots, k-1 \). The k-model for \( M \) is a structure \( M_k=\langle \mathcal{S}, \mathcal{A}, \mathcal{A}, \mathcal{A}_k \rangle \), where \( \mathcal{A}_k \) is the set of all different k-paths of \( M \). We use loop(\( \pi \)) to denote \( \{ s \leq k \land (w_i, w_i+1) \in T \} \).

Definition 6 (bounded semantics). Let \( M_k \) be a k-model of \( M, s, p \in \mathcal{A}, \alpha, \beta \) TECTL formulae in Negation Normal Form (NNF), \( M_k, s, p, \alpha, \beta \) denotes that \( \alpha \) is true in state \( s \) in \( M_k, s, p, \alpha, \beta \) is defined as follows:

\[
M_k, s, p, \alpha, \beta \iff \begin{cases} 
M_k, s, p & \text{if } p \in L(s) \\
M_k, s, \neg p & \text{if } p \notin L(s) \\
M_k, s, \alpha \land \beta & \text{if } (M_k, s, \alpha) \land (M_k, s, \beta) \\
M_k, s, \alpha \lor \beta & \text{if } (M_k, s, \alpha) \lor (M_k, s, \beta) \\
M_k, s, \exists \pi. (w_i = s \land 0 \leq i \leq k(M_k, w_i, w_i + 1) \land \alpha) & \text{if } (M_k, s, \pi, w_i) \land 0 \leq i \leq k(M_k, w_i, w_i + 1) \land \alpha \\
M_k, s, \forall \pi. (w_i = s \land 0 \leq i \leq k(M_k, w_i, w_i + 1) \land \alpha) & \text{if } (M_k, s, \pi, w_i) \land 0 \leq i \leq k(M_k, w_i, w_i + 1) \land \alpha \\
\end{cases}
\]

In TA model, \( k'=k \). In TKS model, \( k'=k \land \text{if the timing constraint is in the form of "\( \leq n \)" or "\( = n \)"}, else \( k'=2k-l' \) where \( l'=\min(\text{loop}(\pi)) \). A TECTL formula \( \psi \) is true in k-model \( M_k \), denoted \( M_k, s, \psi \), if \( \psi \) is true in the initial state of the model \( M_k \). Similar to Ref [7], we have the following theorem.

Theorem 1. Let \( M=\langle \mathcal{S}, \mathcal{A}, \mathcal{A}, \mathcal{A}_k \rangle \) be a k-model, and \( \psi \) a TECTL formula. \( M, \psi \) iff there is \( k \in \{0, \ldots, |M| \} \) such that \( M_k, s, \psi \).

Proof (sketch). Follows from the following facts:

- TECTL formula can be translated to ECTL formula which was given in Section 2.3.
- Since we have assumed that we deal with progressive TA and the TKS is an extension of TK, the above

* More details about the currently supported background theories and logics in SMT-LIB, see http://goedel.cs.uiowa.edu/smtlib/.
translation can be translated to the model checking problem for ECTL.

By Theorem 4.1 of Ref.[6] we can get the conclusion.

With this theorem, we are able to reduce a model checking problem $M \models \phi$ to a bounded model checking problem $M_k \models \phi$ or not.

### 3.2 SMT-Based bounded model checking for TECTL

Given a TECTL formula $\phi$, a model $M$ and a bound $k$, SMT-based BMC approach for TECTL needs to generate and solve a semi-propositional formula $[[M, \phi]]_k \models [[M^+]]_k \land [[\psi]]_{\delta_k}$, where $[[M^+]]_k$ represents the transition relations of the $k$-model $M_k$ and $[[\psi]]_{\delta_k}$ specifies which $k$-paths satisfy $\phi$. The satisfiability of the semi-propositional formula on some $k \leq |M|$ implies that $M$ satisfies $\phi$. Otherwise, $\phi$ do not hold in $M$.

In order to construct $[[M, \phi]]_k$, we first give some useful notions. Let $w$ be a vector state variable, $w = (w[1], \ldots, w[n])$, where $w[i]$ for $i=1, \ldots, n$ are propositional variables and $n$ depends on the size of the model, $n = \log_2(|M|)$. A state can be represented by a truth assignment to $(w[1], \ldots, w[n])$. When we talk about a state $w$, we mean the state represented by $w$ with a given assignment. Let $k \geq 0$, $w_j$ represents the $j$th state on the $k$th path and $\phi_{w_j}$ means that $w_j$ satisfies the time constraint $\phi$. So we use $w_0, \ldots, w_k$ to represent the $k+1$ states of the $i$th $k$-path for each $i \in \{0, \ldots, N_k(\phi) - 1\}$, where $N_k(\phi)$ is the number of different $k$-paths needed for checking the formula $\phi$, and it is defined as follows:

$N_k(\phi) = N_k(\neg \phi) = 0$, where $p \in \mathcal{A}$

$N_k(\phi \land \psi) = N_k(\phi) + N_k(\psi)$

$N_k(\phi \lor \psi) = \max\{N_k(\phi), N_k(\psi)\}$

$N_k(EF_{\phi}) = N_k(\phi) + 1$

$N_k(EG_{\phi}) = (k+1) \cdot N_k(\phi) + 1$

$N_k(E(AU_{\phi})) = k \cdot N_k(\phi) + N_k(\phi) + 1$

**Definition 7 (translation of the $k$-model).** Let $M_k = \langle \text{paths}^k, S, A \rangle$ be a $k$-model of $M$ and $\phi$ a TECTL formula.

The semi-propositional formula $[[M, \phi]]_k$ is defined as follows:

$[[M, \phi]]_k := I(s) \land \bigwedge_{j=0}^{N_k(\phi)-1} \bigwedge_{j=0}^{k} \bigwedge (w_{j+1})$

where $I(s)$ is true when $s$ is the initial state, $w_j$, for $j = 0, \ldots, k$ and $i = 0, \ldots, N_k(\phi) - 1$ are vectors of state variables.

According to the bounded semantics of TECTL, we get the following definition.

**Definition 8 (translation of TECTL formulae).** Let $p \in \mathcal{A}$, $\alpha, \beta$ TECTL formulae, $m, n$ represent states’ number and paths’ number respectively. The rules of a TECTL formula $\phi$ translated to a semi-propositional formula are defined as follows:

$[[p]]^{m,n}_k := p(w_{m,n})$

$[[\neg p]]^{m,n}_k := \neg p(w_{m,n})$

$[[\alpha \land \beta]]^{m,n}_k := [[\alpha]]^{m,n}_k \land [[[\beta]]^{m,n}_k$

$[[\alpha \lor \beta]]^{m,n}_k := [[\alpha]]^{m,n}_k \lor [[[\beta]]^{m,n}_k$

$[[EF_{\alpha}]]^{m,n}_k := \bigvee_{i=0}^{N_k(\alpha)} ((w_{m,n} \equiv w_i) \land \bigvee_{j=0}^{k} (I_{w_i}^{(i)} \land [[\alpha]]^{m,n}_{(j)})$

$[[EG_{\alpha}]]^{m,n}_k := \bigvee_{i=0}^{N_k(\alpha)} ((w_{m,n} \equiv w_i) \land \bigwedge_{j=0}^{k} (I_{w_i}^{(i)} \lor \text{dead}(w_{j+1})) \land \bigwedge_{j=0}^{k} (I_{w_i}^{(i)} \land [[\alpha]]^{m,n}_{(j)})$

$[[E(AU_{\phi})]]^{m,n}_k := \bigvee_{i=0}^{N_k(\alpha \lor \beta)} ((w_{m,n} \equiv w_i) \land \bigvee_{j=0}^{k} (I_{w_i}^{(i)} \land [[\beta]]^{m,n}_{(j)}) \land \bigwedge_{j=0}^{k} (I_{w_i}^{(i)} \land [[\alpha]]^{m,n}_{(j)})$
Let \( [[\psi]]_{M^i} \) denote \( [[\psi]]_{M^i}^{0,0} \) and use the arithmetic expressions to represent \( I_{\psi}^{i,j} \) directly. \( L_{i,j} := T(w_{i,j},w_{j}) \) encodes a backloop from the \( k \)th state to the \( l \)th state in the \( i \)th \(-k\)-path. \( k' \) in \( EG \) case is the same in Definition 6.

So, from the encodings of \( [[M,\psi]] \), we have the following theorem.

**Theorem 2.** Let \( M \) be a model, \( M_k \) be a \( k \)-model, and \( \psi \) a TECTL formula. Then \( M_k \models \psi \) iff \( [[M^y \wedge \psi]] \wedge [[\psi]]_{M_k} \) is satisfiable.

**Proof (sketch).** Follows from the following facts:

- TECTL formula can be translated to ECTL formula which was given in Section 2.3.
- Since we have assumed that we deal with progressive TA and the TKS is an extension of TK, the above translation can be translated to the model checking problem for ECTL. Our definition of the translation of TECTL problem is based on this translation.
- Theorem 6.1 of Ref. [6] shows that the translation of the model checking of ECTL problem to the SAT problem is correct. So our translation of TECTL problem to the SMT problem is correct.

### 3.3 Improved SMT-based encodings

From Definition 8, the translated formula is in the form of \( \phi_0 \lor \phi_1 \lor \ldots \lor \phi_n \), where \( n \in \mathbb{N} \) and \( \phi_0 \lor \ldots \lor \phi_n \) are propositional formulae with the same structure but different path numbers. According to Ref. [25], it’s a rotation symmetric formula. We say a formula is rotation symmetric if, for every permutation \( \sigma = \{k_0 \rightarrow 0, \ldots, k_n \rightarrow n\} \) of its components’ indexes, the permutation of a satisfiability assignment is a satisfiability assignment for the permuted formula. The rotating symmetry in the formula enables us to consider only one fixed order of paths such as the improvement in Ref. [26], yet not affecting its satisfiability.

**Definition 9 (improved SMT-based encoding).** Let \( p \in \mathcal{P}, w \) a state variable, \( \psi, \alpha, \beta \) TECTL formulae, \( k \equiv 0 \).

The encoding \( [[\psi, w]]_{N_k(\psi)}^{i} \) is defined as follows:

\[
\begin{align*}
[[p, w]]_{N_k(\psi)}^{i} & := p(w) \\
[[\neg p, w]]_{N_k(\psi)}^{i} & := \neg p(w) \\
[[\alpha \land \beta, w]]_{N_k(\psi)}^{i} & := [[\alpha, w]]_{k}^{i} \land [[\beta, w]]_{k}^{i} \\
[[\alpha \lor \beta, w]]_{N_k(\psi)}^{i} & := [[\alpha, w]]_{k}^{i} \lor [[\beta, w]]_{k}^{i} \\
[[\alpha \land \beta, w]]_{N_k(\psi)}^{i} & := [[\alpha, w]]_{k}^{i} \land [[\beta, w]]_{k}^{i} \\
[[\alpha \lor \beta, w]]_{N_k(\psi)}^{i} & := [[\alpha, w]]_{k}^{i} \lor [[\beta, w]]_{k}^{i} \\
[[\alpha \land \beta, w]]_{N_k(\psi)}^{i} & := [[\alpha, w]]_{k}^{i} \land [[\beta, w]]_{k}^{i} \\
[[\alpha \lor \beta, w]]_{N_k(\psi)}^{i} & := [[\alpha, w]]_{k}^{i} \lor [[\beta, w]]_{k}^{i} \\
[[\alpha \land \beta, w]]_{N_k(\psi)}^{i} & := [[\alpha, w]]_{k}^{i} \land [[\beta, w]]_{k}^{i} \\
[[\alpha \lor \beta, w]]_{N_k(\psi)}^{i} & := [[\alpha, w]]_{k}^{i} \lor [[\beta, w]]_{k}^{i} \\
[[\alpha \land \beta, w]]_{N_k(\psi)}^{i} & := [[\alpha, w]]_{k}^{i} \land [[\beta, w]]_{k}^{i} \\
[[\alpha \lor \beta, w]]_{N_k(\psi)}^{i} & := [[\alpha, w]]_{k}^{i} \lor [[\beta, w]]_{k}^{i} \\
\end{align*}
\]

\( i \) is the start path number, \( N_k(\psi) \) is the number of paths needed to check the formula \( \psi \), \( g_0 = N_k(\alpha) \) and \( g_1 = N_k(\beta) \). Let \( s \) be a TECTL formula. Then \( M_k \models s \) iff \( [[M^y \wedge s]]_{k} \wedge [[s]]_{M_k} \) is satisfiable.

**Proof (sketch).** Follows from the following facts:

- TECTL formula can be translated to ECTL formula which was given in Section 2.3.
- Since we have assumed that we deal with progressive TA and the TKS is an extension of TK, the above translation can be translated to the model checking problem for ECTL. Our definition of the translation of TECTL problem is based on this translation.
- Theorem 3.2 of Ref. [26] shows that the improved translation of ECTL problem to the SAT problem is correct. So our improved translation of TECTL problem to the SMT problem is correct.
4 Experiments

The experiments were run under RedHat Enterprise Linux AS release 4 (Nahant update 4) on a 2-processor Xeon 3.0GHz machine with 32GB RAM. We use “SMT-based BMC” to represent our methods (it is implemented in the tool[27]), “Improved” means we use the improved translation of TECTL formulae in Definition 9, “Unimproved” means we use the translation of TECTL formulae in Definition 8. We also use SAL 3.0[13], UPPAAL 4.0.6[18], TSMV 1.0[3], NuSMV 2.4.3[2] as comparable methods.

4.1 Fischer’s protocol

The system consists of \( n (n \geq 2) \) processes: \( Process_1, \ldots, Process_n \), which compete for an access to the critical region. Each process in Fischer’s protocol has one local clock \( x \) and can access the global pointer \( lock \) (\( lock := p \) to assign its process ID to the pointer). The TA of one process was shown in Fig.1. \( Process_i \)'s transitions can be described as follows:

\[
\begin{align*}
&i = idle_i \land lock = nil \rightarrow (i, x_i) := (ready_i, 0) \\
&i = ready_i \land x_i < B \rightarrow (i, lock, x_i) := (wait_i, p, 0) \\
&i = wait_i \land lock \neq x_i \lor A \rightarrow (i) := (idle_i) \\
&i = wait_i \land lock = x_i \lor A \rightarrow (i) := (critical_i) \\
&i = critical_i, \rightarrow (i, lock) := (idle_i, nil)
\end{align*}
\]

Experimental Results and Analysis. For \( n \) processes, we verify the negation of mutual exclusion property \( \psi_n : EF(\bigwedge_{i=1}^{n} \text{critical}_i) \). If \( A \geq B \), \( \psi_n \) is false, and if \( A < B \), \( \psi_n \) is true. We compare our method with SAL and UPPAAL, and TA is used as its model. The experimental results with \( A = 1 \) and \( B = 4000 \) are shown in Table 1.

<table>
<thead>
<tr>
<th>( n )</th>
<th>Improved</th>
<th>Unimproved</th>
<th>SAL</th>
<th>UPPAAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>k</td>
<td>Time</td>
<td>Var</td>
<td>Size</td>
<td>Time</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>0.006</td>
<td>102</td>
<td>4.08</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>0.008</td>
<td>179</td>
<td>4.21</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>0.016</td>
<td>278</td>
<td>4.35</td>
</tr>
<tr>
<td>5</td>
<td>12</td>
<td>0.032</td>
<td>399</td>
<td>4.47</td>
</tr>
<tr>
<td>6</td>
<td>14</td>
<td>0.052</td>
<td>542</td>
<td>4.60</td>
</tr>
<tr>
<td>7</td>
<td>16</td>
<td>0.074</td>
<td>707</td>
<td>4.86</td>
</tr>
<tr>
<td>8</td>
<td>18</td>
<td>0.112</td>
<td>893</td>
<td>5.05</td>
</tr>
</tbody>
</table>

The 1st column shows the number of processes. The 2nd column gives the bound of SMT-based BMC. The 3rd to 5th columns give the time, variables and memory needed for the improved SMT-based BMC method. The 6th to 8th columns are the same but for unimproved SMT-based BMC; the 9th and 10th columns display the time and memory needed for SAL. The 11th and 12th columns are the same but for UPPAAL.

From Table 1, we can see that, the SMT-based BMC method spend less time and memory than SAL and the time increasing trend of SMT-based BMC is much slower than SAL and UPPAAL with the increase of processes. Otherwise, the improved SMT-based BMC is much better than the unimproved one in the time, variables and memory they needed.

4.2 Bridge-Crossing problem

The bridge-crossing problem is a famous mathematical puzzle with time critical aspects[28]. A group of four people, called \( P_1, P_2, P_3 \) and \( P_4 \), have to cross a bridge at night. It is dark and they can only cross the bridge if they carry a lamp. Only one lamp is available and at most two of them can cross at the same time. Therefore any solution requires that, after the first two people cross the bridge, one of them returns, bringing back the lamp for the
remaining people. The four people have different maximal speeds: here \( P_1 \) crosses in 5 time units (t.u.), \( P_2 \) in 10 t.u., \( P_3 \) in 20 t.u. and \( P_4 \) in 25 t.u. When a pair crosses the bridge, they move at the speed of the slower one. Now, how much time is required before the whole group is on the other side (in the safe state)?

The details of this model can be found in Ref.[3]. We use TKS as the model in this problem.

**Experimental Results and Analysis.** We have tested two properties of this problem:

- \( \psi_2: EF_{=60T}(safe) \)
  
  The formula expresses the property that the whole group will be safe exactly at 60 t.u.. The property is true as the minimum time to cross the bridge is 60 \times T \text{ t.u.}.

- \( \psi_3: EF_{<59T}(safe) \)
  
  The formula expresses the property that the whole group will be safe at some time less than 60 \times T \text{ t.u..} It is obviously false.

<table>
<thead>
<tr>
<th>( \times T )</th>
<th>( k )</th>
<th>Time</th>
<th>Var</th>
<th>Size</th>
<th>Time</th>
<th>Var</th>
<th>Size</th>
<th>Time</th>
<th>Size</th>
<th>Time</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \times 1 )</td>
<td>5</td>
<td>0.012</td>
<td>280</td>
<td>4.21</td>
<td>0.023</td>
<td>442</td>
<td>4.34</td>
<td>0.006</td>
<td>1.82</td>
<td>0.084</td>
<td>6.17</td>
</tr>
<tr>
<td>( \times 10 )</td>
<td>5</td>
<td>0.014</td>
<td>280</td>
<td>4.21</td>
<td>0.026</td>
<td>442</td>
<td>4.34</td>
<td>0.023</td>
<td>1.93</td>
<td>2.618</td>
<td>41.94</td>
</tr>
<tr>
<td>( \times 20 )</td>
<td>5</td>
<td>0.015</td>
<td>280</td>
<td>4.21</td>
<td>0.024</td>
<td>442</td>
<td>4.34</td>
<td>0.044</td>
<td>2.06</td>
<td>8.945</td>
<td>45.48</td>
</tr>
<tr>
<td>( \times 50 )</td>
<td>5</td>
<td>0.016</td>
<td>280</td>
<td>4.21</td>
<td>0.022</td>
<td>442</td>
<td>4.34</td>
<td>0.162</td>
<td>2.34</td>
<td>58.351</td>
<td>58.54</td>
</tr>
<tr>
<td>( \times 100 )</td>
<td>5</td>
<td>0.017</td>
<td>280</td>
<td>4.21</td>
<td>0.026</td>
<td>442</td>
<td>4.34</td>
<td>0.466</td>
<td>2.76</td>
<td>236.662</td>
<td>110.71</td>
</tr>
<tr>
<td>( \times 200 )</td>
<td>5</td>
<td>0.016</td>
<td>280</td>
<td>4.21</td>
<td>0.026</td>
<td>442</td>
<td>4.34</td>
<td>1.274</td>
<td>3.60</td>
<td>2608.600</td>
<td>204.68</td>
</tr>
</tbody>
</table>

The results show in Table 2 and Table 3. The 1st column in Table 2 shows the magnifications of each one’s crossing time (e.g. "\( \times 10 \)" means replacing 5, 10, 20 and 25 with 50, 100, 200 and 250.). The 2nd to 8th columns are the same in Table 1. The 9th and 10th columns display the time and memory needed for TSMV method. The 11th and 12th columns show the time and memory consumed with NuSMV method. They are the same in Table 3.

For \( \psi_2 \), the bound is 5, which means in Step 5 the system is in the safe state and the property is true. For \( \psi_3 \), the bound is also 5, that means in or after Step 5, the system will never reach the safe state within 59 \times T \text{ t.u.}, and the property is false. This is because the minimal time that the system first in safe state is 60 \times T \text{ t.u.} at \( k=5 \), that means when \( k<5 \) the system cannot reach state safe and when \( k\geq 5 \) the system reaches safe state, as the duration of the system is monotonically increased, \( D_{[0...k]}\geq60\times T \), the timing constraint cannot be satisfied any more. So \( k=5 \) is enough for \( \psi_3 \).

From Table 2 and Table 3, we can see that SMT-based BMC are much faster than TSMV and NuSMV. The time SMT-based BMC used is almost the same with different magnifications, but TSMV and NuSMV are increased exponentially. Also, the performance of the improved SMT-based BMC is better than the unimproved one.

**5 Conclusions and Future Work**

We have considered BMC for real-time systems based on SMT solver Yices to verification TECTL properties.
We have chosen TECTL BMC as our main verification technique because of the advantage that the length of the symbolic paths needed in verifying a TECTL property could be much shorter. SMT-Based BMC can deal with two kinds of models: Timed Automata and Timed Kripke Structures. We have compared our method with some other real-time system model checkers by two well-known examples. The experimental results indicate that the efficiency of our method is better than others. We also did improvement to the SMT-based BMC and the improved one is better than the unimproved one.

For future work, stuttering studied in Ref.[29], linear counterexamples studied in Ref.[30] and some reduction methods can be incorporated into our method to simplify the $k$-model and to reduce the number of verification paths. Another direction is to work on efficient techniques for the verification of valid TECTL properties under the TA model without necessary to reach a high completeness threshold, such as Ref.[31] using $k$-induction or similar to that considered for LTL and ACTL properties in Refs.[32,33].

Acknowledgement We thank Prof. Wenhui Zhang for discussing some issues about this paper.

References:


附中文参考文献: