ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.19, No.1, January 2008, pp.39−47
DOI: 10.3724/SP.J.1001.2008.00039

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

变量极小不可满足在模型检测中的应用*

陈振宇[1], 陶志红[2+], KLEINE BÜNING Hans[3], 王立福[2]

[1](东南大学 计算机科学与工程学院,江苏 南京 210096)

[2](北京大学 软件与微电子学院,北京 100871)

[3](Department of Computer Science, University of Paderborn, Paderborn 33095, Germany)

Applying Variable Minimal Unsatisfiability in Model Checking

CHEN Zhen-Yu[1], TAO Zhi-Hong[2+], KLEINE BÜNING Hans[3], WANG Li-Fu[2]

[1](School of Computer Science and Engineering, Southeast University, Nanjing 210096, China)

[2](School of Software and Microelectronics, Peking University, Beijing 100871, China)

[3](Department of Computer Science, University of Paderborn, Paderborn 33095, Germany)

+ Corresponding author: Phn: +86-10-62767117, E-mail: tzh21001@ss.pku.edu.cn

Chen ZY, Tao ZH, Kleine Büning H, Wang LF. Applying variable minimal unsatisfiability in model checking. *Journal of Software*, 2008,19(1):39−47. http://www.jos.org.cn/1000-9825/19/39.htm

Abstract: This paper presents a framework combining variable abstraction with bounded model checking, in order to prove the counterexamples' absence or establish the counterexamples' existence. A mathematical definition of variable minimal unsatisfiability (VMU) is introduced to drive this abstraction refinement process. The set of variables of VMU formula is a minimal one guaranteeing its unsatisfiability. Furthermore, the authors prove that VMU-driven refinement is valid and minimal by mathematical reasoning. Although the determining problem of VMU is as hard as the well-known problem called minimal unsatisfiability (MU), i.e. D^P-complete, the case study has shown that VMU could be more effective than MU in variable abstraction refinement process.

Key words: minimal unsatisfiability; abstraction refinement; model checking

摘 要: 提出一个结合变量抽象和有界模型检测(BMC)的验证框架,用于证明反例不存在或输出存在反例.引入变量极小不可满足(VMU)的数学概念来驱动抽象精化的验证过程.一个 VMU 公式 F 的变量集合是保证其不可满足性的一个极小集合.严格证明了 VMU 驱动的精化满足抽象精化框架中的两个理想性质:有效性和极小性.虽然 VMU 的判定问题和极小不可满足(MU)一样难,即 D^P 完全的,该案例研究表明,在变量抽象精化过程中,VMU 比 MU 更为有效.

关键词: 极小不可满足;抽象精化;模型检测

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.60425206, 60773104, 60403016, 60633010 (国家自然科学基金); in Part by the Jiangsu Planned Projects for Postdoctoral Research Funds of China under Grant No.0701003B (江苏省博士后科研资助计划)

Received 2006-05-25; Accepted 2006-11-03

1 Introduction

Model checking[1−3] has emerged as a promising and powerful approach of automatic verification, which has been successfully applied to both hardware and software systems. Since model checking entails the exploration of a potentially very large state space, the alleviation of the so-called state explosion problem has been the object of much research.

In the past years, SAT solvers have been found to be quite efficient at producing counterexamples for systems that are too large to allow standard model checking. Bounded Model Checking (BMC)[4] applies SAT techniques in model checking to reduce the state explosion problem. The basic idea of BMC is to look for counterexamples of maximum length k. It generates a propositional formula, which is satisfiable if and only if such counterexamples exist. BMC has come to be regarded as an excellent debugging (as opposed to verification) technique. That is, classical BMC is particularly adept at finding counterexamples, but ill-suited to prove their absence unless an upper bound is known on the depth of the state space. Unfortunately, it is hard to be computed in general.

On the other hand, conservative abstractions[5] have been used to allow the BDD-based model checker to draw conclusions on the original, concrete model by examining a simpler, abstract one. Conservative abstractions benefit from a preservation theorem, which states that the correctness of any universal fragment formula on an abstract model automatically implies the correctness of the formula on the concrete model. However, a counterexample on an abstract model may not correspond to any real path, in which case it is called a spurious counterexample. Therefore, conservative abstractions are suit for proving the counterexamples' absence, but suffer from the spurious counterexamples.

The opportunity of combining conservative abstraction and BMC is that the former proves the counterexamples' absence and the latter establishes the counterexamples' existence. In this paper, we introduce variable abstraction[6] to generate a small abstract model. If it fails in a spurious counterexample of length k, then we use BMC to look for counterexamples of maximum length k in the concrete model. If it fails in finding the real counterexamples, that is, the corresponding propositional formula is unsatisfiable, then we extract a small set of variables, which guarantees unsatisfiability of the formula. The refined abstract model based on these variables rules out all spurious counterexamples of maximum length k. This abstraction refinement process repeats until proving the counterexamples' absence or existence, or model checking is infeasible.

A main challenge of this iterative abstraction refinement process is how to select variables for refinement. Insufficient variables result in failure verification but over-many ones produce a large abstract model. To address it, a mathematical definition of variable minimal unsatisfiability (VMU)[7] is introduced in this paper. It bridges a gap between variable abstraction and BMC to drive this abstraction refinement process, and it gains an insight into applications of a well-known problem called minimal unsatisfiability (MU)[8]. We present some theoretical results and show that the deciding problem of VMU is as hard as MU, i.e., the complexity is D^P-complete[9]. Moreover, we use a two-bit counter example to explain that the VMU-driven abstraction refinement process could work effectively.

A number of variations on iterative abstraction refinement framework have appeared[10−13]. The earlier methods[10,11] refute one spurious counterexample in each iteration. Recently, BMC is introduced to generate an abstraction sufficient to refute all counterexamples within a given length bound in each iteration[12]. Combined with variable abstraction, a robust and powerful approach of abstraction refinement is presented in Ref.[13]. The authors used conflict analysis to identify important variables for refinement. However, all existing efforts described so far lack mathematical definitions and rigorous discussions. The key contribution of our work is the formalizations of refinement requirements and introduction to VMU.

The paper is organized as follows. We brief some basic concepts of SAT notations in Section 2 and introduce variable minimal unsatisfiability in Section 3. A motivation example and variable abstraction are described in Section 4. In Section 5, we present VMU-driven abstraction refinement framework and its two requirements. Furthermore we rigorously prove that VMU meets these two requirements exactly. The conclusion is drawn in the last section.

2 Notation

A propositional formula is a string that represents a Boolean function, which involves some atomic Boolean variables and logical connectives $\neg$(*not*), $\wedge$(*and*), $\vee$(*or*), and etc. A propositional formula is said to be satisfiable if there is at least one satisfying assignment, otherwise the formula is called unsatisfiable. In this paper, *SAT* is the class of satisfiable formulas and *UNSAT* is the class of unsatisfiable formulas.

A literal l is an atomic variable x or its negation $\neg x$. A clause C_i is a disjunction of literals $l_1\vee\ldots\vee l_m$. A cube D_i is a conjunction of literals $l_1\wedge\ldots\wedge l_m$. A formula is in Negation Normal Form (NNF) if the only connectives in it are $\wedge$, $\vee$, and $\neg$, where $\neg$ is only applied to atomic variables. A formula is in Conjunctive Normal Form (CNF) if it has the conjunction form of clauses $C_1\wedge\ldots\wedge C_n$. A formula is in Disjunctive Normal Form (DNF) if it has the disjunction form of cubes $D_1\vee\ldots\vee D_n$. A clause or a cube could be regarded as a set of literals. A CNF formula could be regarded as a set of clauses. A DNF formula could be regarded as a set of cubes.

Given two formulas F and F', if each satisfying assignment of F is also a satisfying one of F', then we call F entails F', denoted by $F|=F'$. F is equivalent to F', denoted by $F\approx F'$, if and only if $F|=F'$ and $F'|=F$. F is satisfiable equivalent to F', denoted by $F\approx_s F'$, if and only if $F\in SAT\Leftrightarrow F'\in SAT$. For an arbitrary formula, there exists a standard procedure to produce a satisfiable equivalent CNF formula in polynomial time[14]. In this paper, $var(F)$ denotes the set of variables of F.

3 Variable Minimal Unsatisfiability

There are many applications that can benefit from extracting a small unsatisfiable core (UC) from an unsatisfiable formula. When a propositional formula is shown unsatisfiable, a need arises to identify the causes of its unsatisfiability in order that a feasible design may be obtainable by revising its model specifications. Smaller unsatisfiable cores would be helpful to localize the reasons of the unsatisfiability.

A well-known problem relevant to unsatisfiable core is called minimal unsatisfiability (MU)[8]. A CNF formula is minimal unsatisfiable if and only if the formula is unsatisfiable and removing an arbitrary clause will result in a satisfiable formula, that is

$$MU:=\{F\in UNSAT\cap CNF|\forall F'\subset F:F\in SAT\}.$$

More existing work on *MU* could be found in Ref.[8]. Intuitively, the set of clauses of an *MU* formula is a minimal set, which guarantees unsatisfiability. That is, an *MU* formula F is characterized by the condition that every clause of F is used in every resolution refutation of F. Another interesting characterization is that every variable of F is used in every resolution refutation of F. That is, the set of variables of F is a minimal set, which guarantees unsatisfiability. It is called variable minimal unsatisfiability (VMU)[7].

Firstly, we define a simple operation for NNF formulas. $F[V]$ is a formula obtained from F by substituting each literal l, where its variable is not in V, by true. Intuitively, $F[V]$ is a resulting formula by hiding the variables not in V. For a *CNF* formula F, $F[V]$ is obtained from F by removing all clauses with some variables not in V. Given an NNF formula F and a set of variables $V\subseteq var(F)$, obviously $F|=F[V]$. Therefore, for any unsatisfiable formula F, there is $V\subseteq var(F)$, such that $F[V]$ is satisfiable. Please notice that, if V is empty, then $F[V]$ is true trivially. F is

variable minimal unsatisfiable, if and only if $var(F)$ is a minimal set of variables which guarantees unsatisfiability of F. The mathematical definition is described as follows.

Definition 1. Variable Minimal Unsatisfiability

$$VMU^*:=\{F\in UNSAT|\forall V\subset var(F):F[V]\in SAT\}.$$

In the following section, we restrict VMU formula to CNF formula and define $VMU:=VMU^*\cap CNF$. Following the definition of VMU, if $F\in VMU$, then $F\in UNSAT$. Furthermore, hiding an invisible variable will lead to removing at least one clause. Thus if $F\in MU$, then $F\in VMU$. Therefore, $MU\subset VMU\subset UNSAT$[7]. In Ref.[9], MU is shown to be D^P-complete, where D^P is the class which can be described as the difference between two NP problems. A D^P-complete problem is equivalent to solving a SAT-$UNSAT$ problem defined as: given two formulas F and F', is it the case that F is satisfiable and F' is unsatisfiable? D^P-complete problems are both NP-hard and $coNP$-hard. In Ref.[7], it is shown that the deciding problem of VMU is as hard as MU, i.e., D^P-complete.

Theorem 1[7]. VMU is D^P-complete.

4 Model and Abstraction

Considering a system with a set of Boolean variables $V=\{v_1,\ldots,v_n\}$ over {false,true}. The system is modeled by a concrete model $M=(S,T,I)$, where:

1. S is a set of (concrete) states.
2. $T\subseteq S\times S$ is a transition relation.
3. $I\subseteq S$ is a set of initial states.

Each state S can be represented as a cube $(l_1,\ldots,l_n)$. Each transition $T_j(s,s')$ can be considered as a cube $(l_1,\ldots,l_n,l'_1,\ldots,l'_n)$, in which l_i is a literal of v_i in current state s, and l'_i is a literal of v_i in next state s'. The transition relation of system is the disjunction of all $T_j(s,s')$, i.e. $T(s,s')=\vee T_j(s,s')$, thus $T(s,s')$ is in DNF. Similarly, I can be considered as a DNF formula $l_1\wedge\ldots\wedge l_n$. We say that the current state of system is s if the assignment of system variables satisfies s. State s is an initial state if and only if $s|=I$, i.e. $I(s)=true$. There is a transition from s to s′ if and only if $(s\wedge s')|=T$, i.e., $T(s,s')$=true.

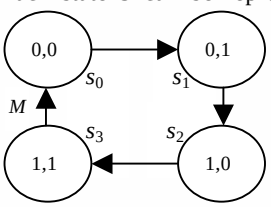

Fig.1 A two-bit counter

An example of a two-bit counter is shown in Fig.1. $V=\{x_1,x_2\}$, in which x_2 is the left bit variable and x_1 is the right bit variable. $S=\{s_0,s_1,s_2,s_3\}$, in which $s_0=(\neg x_2,\neg x_1)$, $s_1=(\neg x_2,x_1)$, $s_2=(x_2,\neg x_1)$, $s_3=(x_2,x_1)$; $I(s)=(\neg x_2\wedge\neg x_1)$; and

$$T(s,s')=(\neg x_2\wedge\neg x_1\wedge\neg x'_2\wedge x'_1)\vee(\neg x_2\wedge x_1\wedge x'_2\wedge\neg x'_1)\vee(x_2\wedge\neg x_1\wedge x'_2\wedge x'_1)\vee(x_2\wedge x_1\wedge\neg x'_2\wedge\neg x'_1).$$

4.1 Variable abstraction

For a realistic system, the number of variables is usually more than hundreds even thousands. In order to reduce the state space, we use variable abstraction to construct a conservative abstract model. We extract a set of variables from V (called *visible* variables), denoted by V^a. V^a corresponds to the part of the system that is currently believed to be important for verifying the desired property. $V^I=V-V^a$ is the set of *invisible* variables.

In this paper, we adopt a minor revision of variable abstraction[6] in model checking. An abstraction of s on V^a, denoted by $s[V^a]$, is defined by removing all literals of invisible variables for s represented as a cube. A variable abstraction function h over V^a for the system is defined as $h(s)=s[V^a]$. Given two concrete states s and s', they are equivalent over h if and only if $h(s)=h(s')$. An abstraction function h could be considered as a surjection h: $S\rightarrow S^a$, which maps a concrete state in S to an abstract state in S^a. The abstraction of system is modeled by an abstract model $M^a=(S^a,T^a,I^a)$, where:

1. $S^a=\{s^a|\exists s\in S:h(s)=s^a\}$ is a set of abstract states.

2. $T^a=\{(s^a,s'^a)|\exists s,s'\in S.\ T(s,s')\wedge h(s)=s^a\wedge h(s')=s'^a\}$ is an abstract transition relation.

3. $I^a=\{s^a|\exists s\in I.\ h(s)=s^a\}$ is a set of initial abstract states.

$s[V^a]$ could be considered as s^a. Similarly, $T(s,s')[V^a]=T^a(s[V^a],s'[V^a])=T^a(s^a,s'^a)$, $T[V^a]=T^a$, $I[V^a]=I^a$, and $M[V^a]=M^a$. M^a is a conservative abstraction, that is, all behaviors in M are preserved in M^a. It is formalized as the following preservation theorem.

Theorem 2[5]. If M is a concrete model and M^a is a corresponding abstract model, then for each universal fragment formula f,

$$M^a|=f\Rightarrow M|=f.$$

Fig.2 shows two variable abstract models. For the left one, the set of visible variables is $\{x_1\}$, i.e. hiding the invisible variable x_2. Therefore,

$$I^a(s)=\neg x_1, T^a(s,s')=(x_1\wedge\neg x_1')\vee(\neg x_1\wedge x_1').$$

For the right one, the set of visible variables is $\{x_2\}$, i.e. hiding the invisible variable x_1. Therefore,

$$I^a(s)=\neg x_2, T^a(s,s')=(x_2\wedge x_2')\vee(x_2\wedge\neg x_2')\vee(\neg x_2\wedge x_2')\vee(\neg x_2\wedge\neg x_2')$$

(a) $M_1^a: V^a=\{x_1\}$ (b) $M_2^a: V^a=\{x_2\}$

Fig.2 Two abstract models

5 VMU-Driven Abstraction Refinement Framework

Many existing efforts describe iterative abstraction refinement process in model checking[10−13]. In this paper, we present an abstraction refinement process driven by unsatisfiable cores (UC). This framework is shown in Fig.3.

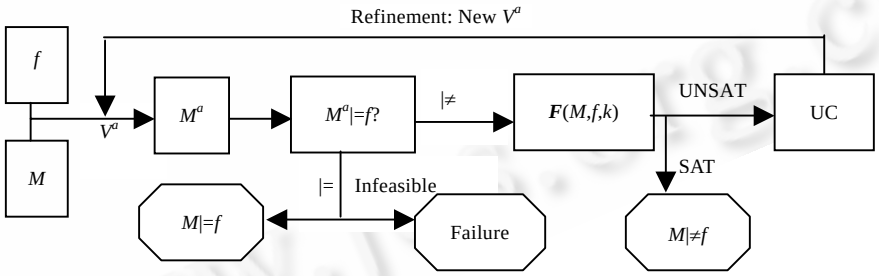

Fig.3 Abstraction refinement framework

Initially, the set of visible variables V^a includes the variables involved in the desired property f. Then, an abstraction function h is given. To construct an abstract model M^a, it can be computed directly from the description language of the original system, by hiding the invisible variables not in V^a.

When an abstract model M^a is generated, in each iteration, we check whether the abstract model satisfies the property f with a BDD-based model checker[15]. If M^a satisfies f, then M satisfies f. However, if M^a does not satisfy f, we could not draw any conclusion, since the abstract counterexample may be spurious. A spurious path is an abstract path, which does not have any corresponding concrete path. For example, suppose $f_1=AF(x_1==\text{true})$ and $f_2=AF(x_2==\text{true})$. We do model checking M_1^a (Fig.2(a)) with f_1, and the answer is "yes", i.e., $M_1^a|=f_1$, then $M|=f_1$ (M is the concrete model in Fig.1). On the other hand, we do model checking M_2^a (Fig.2(b)) with f_2, and the

answer is "no", i.e., $M_2^a \not\models f_2$, then it produces an abstract counterexample $\pi^a = s_0^a\ s_0^a\ s_0^a \ldots$. Actually, this counterexample is spurious and we will explain later.

An excellent technique of verifying counterexamples is bounded model checking (BMC)[4], which focuses on the search for counterexamples of bounded depth k. Effectively, the problem is translated into a propositional formula $\boldsymbol{F}(M,f,k):=\boldsymbol{F}(M,k)\wedge\boldsymbol{F}(f,k)$, such that the formula is satisfiable if and only if there exists a counterexample of maximum length k. $\boldsymbol{F}(M,k):=I(s_0)\wedge\wedge_{i=0}^{k-1}T(s_i,s_{i+1})$ is a propositional formula corresponding to a path in M, $\boldsymbol{F}(f,k)$ is a propositional formula of translation of a property f. More details could be found in Ref.[4].

Consider an example of a two-bit counter (M) in Fig.2. $\boldsymbol{F}(M,k)=I(s_0)\wedge\wedge_{i=0}^{k-1}T(s_i,s_{i+1})$, in which, $I(s_0)=(\neg x_2^0 \wedge \neg x_1^0)$, $T(s_i,s_{i+1})=(\neg x_2^i \wedge \neg x_1^i \wedge \neg x_2^{i+1} \wedge x_1^{i+1})\vee(\neg x_2^i \wedge x_1^i \wedge x_2^{i+1} \wedge \neg x_1^{i+1})\vee(x_2^i \wedge \neg x_1^i \wedge x_2^{i+1} \wedge x_1^{i+1})\vee(x_2^i \wedge x_1^i \wedge \neg x_2^{i+1} \wedge \neg x_1^{i+1})$. For the property specification $f=\neg f_2=EG(x_2==\text{false})$, $\boldsymbol{F}(f,k)=\wedge_{i=0}^{k-1}\neg x_2^i$. For the property of a loop path, it is required to conjunct with $\boldsymbol{F}_{loop}^k=\vee_{j=0}^{k-1}(s_k=s_j)$. Thus, $\boldsymbol{F}(M,f,k)=\boldsymbol{F}(M,k)\wedge\boldsymbol{F}(f,k)\wedge\boldsymbol{F}_{loop}^k$. It is not difficult to check that $\boldsymbol{F}(M,f,k)$ is unsatisfiable with $k\leq 3$ and it is satisfiable with $k=4$. $M\not\models f$, thus $M\not\models f_2$.

When M^a does not satisfy f, an abstract counterexample π_k^a of length k is reported from model checker. We use BMC to verify whether the corresponding propositional formula $\boldsymbol{F}(M,f,k)$ is satisfiable. If $\boldsymbol{F}(M,f,k)$ is satisfiable, then there exist some real counterexamples of maximum length k. If $\boldsymbol{F}(M,f,k)$ is unsatisfiable, then π_k^a is spurious. That means the abstract model based on V^a is too coarse and it fails to prove the desired property. An unsatisfiable core (UC) needs to be extracted from $\boldsymbol{F}(M,f,k)$ to produce a new set of visible variables V^a, which is used to refine the abstract model. Furthermore, in order to generate a small abstract model, V^a is required to be minimal ideally. Thus V^a must meet the following requirements:

- ***R*1** (**valid refinement**): The refined abstract model M^a based on V^a rules out all spurious counterexamples of maximum length k, i.e., $\boldsymbol{F}(M^a,f,k)$ is unsatisfiable.
- ***R*2** (**minimal refinement**): For any $V'^a\subset V^a$, the refined abstract model M' based on V'^a contains at least one spurious counterexample of maximum length k, i.e., $\boldsymbol{F}(M',f,k)$ is satisfiable.

This abstraction refinement process repeats until proving the counterexamples' absence or existence, or model checking is infeasible (e.g. memory is overflow). In principle, minimal refinement is not necessary. Though it could result in better performance, we have to weigh this against the cost of additional work. This comparison would not be discussed in this paper. In our abstraction refinement process, it is required that V^a meets ***R*1** and ***R*2** ideally.

5.1 VMU-Driven refinement meets *R*1 and *R*2

When BDD-based model checker[15] verifies the abstract model M^a, if $M^a\not\models f$ (otherwise verification is finished), then an abstract counterexample of length k is produced, thus $\boldsymbol{F}(M^a,f,k)$ is satisfiable. If this counterexample is spurious (otherwise verification is finished), then $\boldsymbol{F}(M,f,k)$ is unsatisfiable.

For example, we use a BDD-based model checker[15] to verify whether M_2^a (Fig.2(b)) satisfies the property $f_2=AF(x_2==\text{true})$, and it returns the answer "no", i.e., $M_2^a\not\models f_2$, then it produces an abstract counterexample $\pi_3^a = s_0^a\ s_0^a\ s_0^a$. Then we check whether $\boldsymbol{F}(M,f,2)$ is satisfiable by a SAT solver, in which M is the model in Fig.1 and $f=\neg f_2=EG(x_2==\text{false})$. For more details, It is not difficult to see that $\boldsymbol{F}(M,f,2)$ is unsatisfiable, thus π_3^a is a spurious counterexample.

Intuitively, M^a is a conservative abstraction of M, that is, any behavior in M will be preserved in M^a. Please notice that $\boldsymbol{F}(M,f,k)=I(s_0)\wedge\wedge_{i=0}^{k-1}T(s_i,s_{i+1})\wedge\boldsymbol{F}(f,k)$, $\boldsymbol{F}(M^a,f,k)=I^a(s_0^a)\wedge\wedge_{i=0}^{k-1}T^a(s_i^a,s_{i+1}^a)\wedge\boldsymbol{F}(f,k)$, $I\Rightarrow I^a$ and $T\Rightarrow T^a$. Therefore, $\boldsymbol{F}(M,f,k)\Rightarrow\boldsymbol{F}(M^a,f,k)$. If $\boldsymbol{F}(M,f,k)$ is unsatisfiable and $\boldsymbol{F}(M^a,f,k)$ is satisfiable, then M^a must contain some spurious counterexamples of maximum length k.

When BMC verifies the counterexamples of maximum length k, a propositional formula $\boldsymbol{F}(M,f,k)$ is generated. And it is always converted into a *CNF* formula to be input in a SAT solver. For simplicity, we consider $\boldsymbol{F}(f,k)$ as a *CNF* formula. For $\boldsymbol{F}(M,k)$, the initial predicate I and the transition relation T are represented in DNF in this paper. For a DNF formula, there exist two popular approaches to generate *CNF* formula. One is distribution law. A DNF formula F could be converted into a logical equivalent *CNF* formula, denoted by $F\approx CNF_d(F)$. However, the length of $CNF_d(F)$ formula may grow exponentially. The other is substitution method. There is a standard procedure (**TREE-CNF** in page 31 of Ref.[14]) to produce a satisfiable equivalent *CNF* formula, denoted by $F\approx_s CNF_s(F)$. The length of $CNF_s(F)$ only grows polynomially. This construction will be introduced some new variables, denoted by V^n in this paper. The basic idea of this method will be described here. For example, a DNF formula $F=(l_1\wedge l_2)\vee C$, in which C represents other clauses. We introduce a new variable v to substitute $(l_1\wedge l_2)$. Thus, $(\neg v\vee l_1)\wedge(\neg v\vee l_2)\wedge(v\vee C)$. Such substitution repeats until a *CNF* formula $CNF_s(F)$ is generated. Moreover, F is satisfiable if and only if $CNF_s(F)$ is satisfiable[14].

For the formula of bounded model checking, we convert it into a logical equivalent *CNF* formula as follows:

$CNF_d(\boldsymbol{F}(M,k))=CNF_d(I(s_0))\wedge\wedge_{i=0}^{k-1}CNF_d(T(s_i,s_{i+1}))$, and $CNF_d(\boldsymbol{F}(M,f,k))=CNF_d(\boldsymbol{F}(M,k))\wedge\boldsymbol{F}(f,k)$.

The satisfiable equivalent translation $CNFs(F)$ is similar.

Lemma 1. Given a concrete model M and an abstract model M^a based on V^a, then

$$\boldsymbol{F}(M^a,k)\approx CNF_d(\boldsymbol{F}(M,k))[V^a],\ \boldsymbol{F}(M^a,k)\approx_s CNF_s(\boldsymbol{F}(M,k))[V^a\cup V^n].$$

Proof: Please notice that $\boldsymbol{F}(M^a,k)=I^a(s_0^a)\wedge\wedge_{i=0}^{k-1}T^a(s_i^a,s_{i+1}^a)=I(s_0)[V^a]\wedge\wedge_{i=0}^{k-1}T(s_i,s_{i+1})[V^a]$.

For CNF_d, we use distribution law to convert a DNF formula into a logical equivalent CNF formula. That is $CNF_d((a\wedge b)\vee c)=(a\vee c)\wedge(b\vee c)$. This distribution law works on each subformula of DNF formula until a *CNF* formula is generated. It is not difficult to see that removing a literal from a clause in DNF formula is logical equivalent to removing a clause, which contains this literal, in CNF_d formula.

For CNF_s, we use substitution law to convert a DNF formula into a satisfiable equivalent *CNF* formula. That is $CNF_s((l_1\wedge l_2)\vee C)=(\neg v\vee l_1)\wedge(\neg v\vee l_2)\wedge(v\vee C)$. This substitution law works on each subformula of DNF formula until a *CNF* formula is generated. It is not difficult to see that removing a literal from a clause in DNF formula is satisfiable equivalent to removing a clause, which contains this literal, in CNF_s formula. □

Now we draw a conclusion that the new set of variables, which is generated from VMU-driven refinement, meets ***R*1** and ***R*2**.

Theorem 3. Given a concrete model M and an abstract model M^a. If $\boldsymbol{F}(M,f,k)$ is unsatisfiable and $\boldsymbol{F}(M^a,f,k)$ is satisfiable, that is, M^a contains at least one spurious counterexample of length k, then there exists a $V_{\min}\subseteq var(\boldsymbol{F}(M,f,k))$, such that $CNF_d(\boldsymbol{F}(M,f,k))[V_{\min}]\in VMU$ (resp. to CNF_s), and $V_{\min}$ meets ***R*1** and ***R*2**.

Proof: The existence of $V_{\min}$ is obvious. Now, we prove that $V_{\min}$ meets ***R*1** and ***R*2**. Suppose $M^a_{\min}$ is the refined abstract model based on $V_{\min}$.

(1) $V_{\min}$ meets ***R*1**.

$CNF_d(\boldsymbol{F}(M,f,k))[V_{\min}]\in VMU$

$\Rightarrow CNF_d(\boldsymbol{F}(M,f,k))[V_{\min}]\in UNSAT$

$\Rightarrow CNF_d(\boldsymbol{F}(M^a_{\min},f,k))\in UNSAT$ (by lemma 1)

$\Rightarrow \boldsymbol{F}(M^a_{\min},f,k)\in UNSAT$

(2) $V_{\min}$ meets ***R*2**.

$CNF_d(\boldsymbol{F}(M,f,k))[V_{\min}]\in VMU$

$\Rightarrow \forall V'\subset V_{\min}: CNF_d(\boldsymbol{F}(M,f,k))[V']\in SAT$

$\Rightarrow CNF_d(\boldsymbol{F}(M'^a,f,k))\in SAT$ (by lemma 1)

$\Rightarrow \boldsymbol{F}(M'^{a},f,k)\in SAT$

(M'^{a} is the refined abstract model based on V')

The proof of CNF_s is similar. □

5.2 Comparison between *MU* and *VMU*

In general, there exists no efficient procedure to solve *MU* and *VMU* (D^P-complete problems are both *NP*-hard and *coNP*-hard). Many methods for finding small unsatisfiable cores have been developed in recent years. In Refs.[16,17], they use information from a SAT salver's resolution procedure to find unsatisfiable cores. Their experimental results show that their procedures are successful for finding small unsatisfiable cores for the instances tested. Although the theoretic results on *VMU* and *MU* are similar, we believe that extraction of *VMU* would be more practical than *MU* in our abstraction refinement framework, based on the following observations:

- In an unsatisfiable *CNF* formula, the variables are often far less than the clauses. Extraction based on variables would be easier than clauses.
- (a) $F\in UNSAT\Rightarrow\exists F''\subseteq F'\subseteq F$: $F'\in VMU\wedge F''\in MU$; (b) $MU\subset VMU\subset UNSAT$. *VMU* is a generalization of *MU*. That means, for a minimal set of variables which guarantees unsatisfiability, extraction of *VMU* would be potentially easier than *MU*.

For example, the transition relation in Fig.1 has nearly 50 clauses for $CNF_d(T(s_i,s_{i+1}))$ and 17 clauses for $CNF_s(T(s_i,s_{i+1}))$. Consequently, there are many potential subformulas that must be considered for *MU*. However, for the extraction of *VMU* subformulas, we only consider the subsets of $\{x_1^i, x_2^i, x_1^{i+1}, x_2^{i+1}\}$, actually the subsets of $\{x_1,x_2\}$.

6 Conclusion

In this paper, we describe a *VMU*-driven abstraction refinement framework, which uses variable abstraction to prove the counterexamples' absence and uses BMC to establish the counterexamples' existence. Two requirements of this framework, valid refinement and minimal refinement are introduced. A novel contribution of this paper is that we introduce *VMU* in abstraction refinement and prove that *VMU* meets the two requirements in a rigorous way. Furthermore, a case study and some observations are shown that *VMU* could be more practical than *MU* in this abstraction refinement framework.

Acknowledgement The authors would like to thank Prof. DING De-Cheng and Prof. XU Bao-Wen for their valuable discussions and thank the anonymous referees for their helpful comments and suggestions.

References:

[1] Clarke EM, Grumberg O, Peled DA. Model Checking. Cambridge: The MIT Press, 1999.

[2] Lin HM, Zhang WH. Model checking: Theories, techniques and applications. Acta Electronica Sinica, 2002,30(12A):1907−1912 (in Chinese with English abstract).

[3] Su KL, Luo XY, Lu GF. Symbolic model checking for CTL*. Chinese Journal of Computers, 2005,28(11):1798−1806 (in Chinese with English abstract).

[4] Biere A, Cimatti A, Clarke EM, Strichman O, Zhu Y. Bounded Model Checking. Advances in Computers. Academic Press, 2003. 58.

[5] Clarke EM, Grumberg O, Long DE. Model checking and abstraction. ACM Trans. on Programming Languages and Systems, 1994, 16(5):1512−1542.

[6] Kurshan RP. Computer-Aided Verification of Coordinating Processes. Princeton University Press, 1994.

[7] Chen ZY, Ding DC. Variable minimal unsatisfiability. In: Proc. of the Theory and Applications of Models of Computation. LNCS 3959, Springer-Verlag, 2006. 262−273.

[8] Zhao XS. Complexity results on minimal unsatisfiable formulas—A survey. In: Proc. of the 9th Asian Logic Conf. Novosbirsk, 2005. 301−319.

[9] Papadimitriou CH, Wolfe D. The complexity of facets resolved. Journal of Computer and System Science, 1988,37(1):2−13.

[10] Chauhan P, Clarke EM, Kukula J, Sapra S, Veith H, Wang D. Automated abstraction refinement for model checking large state spaces using sat based conflict analysis. In: Proc. of the Conf. on Formal Methods in Computer Aided Design. LNCS 2517, Springer-Verlag, 2002. 33−51.

[11] Chen ZY, Zhou CH, Ding DC. Automatic abstraction refinement for Petri nets verification. In: Proc. of the 10th IEEE Int'l on High-Level Design Validation and Test Workshop. 2005. 168−174.

[12] McMillan KL, Amla N. Automatic abstraction without counterexamples. In: Proc. of the Workshop on Tools for Algorithms for Construction and Analysis of Systems. LNCS 2619, Springer-Verlag, 2003. 2−17.

[13] Gupta A, Strichman O. Abstraction refinement for bounded model checking. In: Proc. of the 17th Int'l Conf. on Computer Aided Verification. LNCS 3576, Springer-Verlag, 2005. 112−124.

[14] Kleine Büning H, Lettmann T. Propositional Logic: Deduction and Algorithms. Cambridge University Press, 1999.

[15] Cimatti A, Clarke EM, Giunchiglia F, Roveri M. NuSMV: A new symbolic model checker. Int'l Journal on Software Tools for Technology Transfer, 2000,2(4):410−425.

[16] Zhang L, Malik S. Extracting small unsatisfiable cores from unsatisfiable Boolean formula. In: Proc. of the 6th Int'l Conf. on Theory and Applications of Satisfiability Testing (SAT 2003). 2003. http://research.microsoft.com/users/lintaoz/papers/SAT_2003_core.pdf

[17] Li XW, Li GH, Shao M. Formal verification techniques based on boolean satisfiability problem. Journal of Computer Science and Technology, 2006,20(1):38−47.

附中文参考文献:

[2] 林惠民,张文辉.模型检测:理论、方法与应用.电子学报,2002,30(12A):1907−1912.

[3] 苏开乐,骆翔宇,吕关锋.符号化模型检测 CTL*.计算机学报,2005,28(11):1798−1806.

CHEN Zhen-Yu was born in 1978. He is a post-doctor at the School of Computer Science and Engineering, Southeast University. His current research areas are formal verification and software testing, etc.

TAO Zhi-Hong was born in 1965. His research areas are software engineering and model checking, etc.

KLEINE BÜNING H was born in 1948. He is a professor at the Paderborn University. His research areas are SAT problems and software engineering, etc.

WANG Li-Fu is a professor at the Peking University. His research areas are software engineering, etc.