Minimizing Counterexample with Unit Core Extraction and Incremental SAT

ShengYu Shen, Ying Qin, and SiKun Li

Office 607,School of Computer Science, National University of Defense Technology
410073 ChangSha, China
{syshen,qy123,skli}@nudt.edu.cn
http://www.nudt.edu.cn

Abstract. It is a hotly researching topic to eliminate irrelevant variables from counterexample, to make it easier to be understood. K Ravi proposes a two-stages counterexample minimization algorithm. This algorithm is the most effective one among all existing approaches, but time overhead of its second stage(called BFL) is very large due to one call to SAT solver per candidate variable to be eliminated. So we propose a faster counterexample minimization algorithm based on unit core extraction and incremental SAT. First, for every unsatisfiable instance of BFL, we perform unit core extraction algorithm to extract the set of variables that are sufficient to lead to conflict, all variables not belong to this set can be eliminated simultaneously. In this way, we can eliminate many variables with only one call to SAT solver. At the same time, we employ incremental SAT approach to share learned clauses between similar instances of BFL, to prevent overlapped state space from being searched repeatedly. Theoretic analysis and experiment result show that, our approach is 1 order of magnitude faster than K Ravi's algorithm, and still retains its ability to eliminate irrelevant variables.

1 Introduction

Model checking technology is widely employed to verify software and hardware system. One of its major advantages in comparison to such method as theorem proving is the production of a counterexample, which explains how the system violates some assertion.

However, it is a tedious task to understand the complex counterexamples generated by model checker. Therefore, how to automatically extract useful information to aid the understanding of counterexample, is an area of hotly research [5, 6, 13, 14].

A counterexample can be viewed as an assignment to a variable set $Free$, There must be a variables subset $R \subseteq Free$, which is sufficient to lead to counterexample. Then for variables in $Free - R$,no matter what value do they take on, they can't prevent the counterexample. Thus we call R as **minimization set**, and call the process that extract R as **counterexample minimization**.

Now we demonstrate the concept of counterexample minimization with following example:

R. Cousot (Ed.): VMCAI 2005, LNCS 3385, pp. 298–312, 2005.

For AND gate $z = a\&b$, assume the assertion is "z always equal to 1", then there are three counterexamples: $\{a \Leftarrow 0, b \Leftarrow 0, z \Leftarrow 0\}$, $\{a \Leftarrow 1, b \Leftarrow 0, z \Leftarrow 0\}$, and $\{a \Leftarrow 0, b \Leftarrow 1, z \Leftarrow 0\}$.
However, from an intuitive viewpoint, b is an irrelevant variable when a equal to 0. At the same time, a is also an irrelevant variable when b equals to 0. Then we can minimize above three counterexamples, and obtain 2 minimization sets: $\{a \Leftarrow 0, z \Leftarrow 0\}$ and $\{b \Leftarrow 0, z \Leftarrow 0\}$.

Thus, a minimized counterexample is much easier to be understood.

K Ravi[5] proposes a two-stage counterexample minimization algorithm. In the first stage, an Implication Graph Based Lifting(IGBF) algorithm is performed to quickly eliminate some irrelevant variables. In the second stage, a highly expensive Brute Force Lifting algorithm(BFL) is performed to further eliminate more irrelevant variables.

In BFL, free variables set contains input variables at all cycle and the initial state variables. "free" means that they can take on any value independent of others. For every free variable v, BFL constructs a SAT instance SAT(v), to determine if some assignment to v can prevent the counterexample. If SAT(v) is unsatisfiable, then v can't prevent the counterexample from happening , thus v is irrelevant to counterexample and can be eliminated.

K Ravi[5] compares his approach with other counterexample minimization approaches, and concludes that his approach is the most efficient one, it can often eliminates up to 70% free variables. However, the run time complexity of his approach is much higher than all other existing approaches. At the same time, run time overhead of BFL is 1 to 3 orders of magnitude larger than that of IGBF. So the key to speedup K Ravi's two-stage approach is to reduce time overhead of BFL.

The reasons of BFL's large time overhead are:

1. It needs to call SAT solver for every free variable. But there are often thousands of free variables in a counterexample. This means BFL needs to call SAT solver thousands of times, it is a huge overhead;
2. It can't share learned clause between similar SAT instances, so overlapped state space may be searched repeatedly.

Accordingly, the keys to reduce time overhead of BFL are:

1. Eliminate as many as possible variables after every call to SAT solver;
2. Share learned clauses between similar SAT instances, to avoid searching overlapped state space repeatedly.

So we propose a faster counterexample minimization algorithm based on unit core extraction and incremental SAT. First, for every unsatisfiable instance of BFL, we perform unit core extraction algorithm to extract the set of variables that are sufficient to lead to conflict, all variables not belong to this set can be eliminated simultaneously. In this way, we can eliminate many variables with only one call to SAT solver. At the same time, we employ incremental SAT

approach to share learned clauses between similar instances of BFL, to prevent overlapped state space from being searched repeatedly.

We implement our algorithm based on zchaff[10] and NuSMV[9], and perform experiment on ISCAS89 benchmark suite[11]. Experiment result shows that, our approach is 1 order of magnitude faster than K Ravi's algorithm[5], and without any lost in its ability to eliminate irrelevant variables.

The remainder of this paper is organized as follows. Section 2 presents background material. Section 3 presents the counterexample minimization approach based on unit core extraction. Section 4 presents the incremental SAT approach. Section 5 presents experiment result of our approach and compares it to that of K Ravi's approach[5]. Section 6 reviews related works. Section 7 concludes with a note on future work.

2 Preliminaries

2.1 Satisfiability Solvers

Basic Notions of Satisfiability Solvers

Given a Boolean formula F, the SAT problem involves finding whether a satisfying assignment for F exists. A SAT solver typically computes a total satisfying assignment for F, if one exists, otherwise returns an UNSATISFIABLE answer. In a SAT solver a Boolean formula F is usually represented in CNF. For instance,

$$f = (a \vee b) \wedge (\neg c \vee d) \tag{1}$$

A CNF formula is a conjunction of clauses. A clause is a disjunction of literals. A literal is a variable or its negation. As shown by equation (1), formula f contains two clauses:$(a \vee b)$ and $(\neg c \vee d)$. Clause $(\neg c \vee d)$ include two literals:$\neg c$ and d. $\neg c$ is a negative phase literal of variable c, and d is a positive phase literal of d.

A *total* satisfying assignment for f is $\{(a,0),(b,1),(c,0),(d,0)\}$. "total" means that it contains assignments to all variables. $\{(b,1),(c,0)\}$ is a *partial* satisfying assignment because it contains only assignments to a subset of variables.

It is also convenient to use literals to designate variable-value pairs. For example, the assignment $\{(a,0),(b,1),(c,0),(d,0)\}$ can be denoted by $\{\neg a, b, \neg c, \neg d\}$.

Implication Graph

According to *unit clause rule*, when a clause contains only one unassigned literal, and all other literals are rendered false, then the variable of this unassigned literal can take on its value according to its phase. This mechanism is called *implication*. For instance, for clause $(\neg c \vee d)$ and assignment $\{\neg d\}$, variable c must take on value 0 to make this clause true.

With this implication mechanism, we can construct an Implication Graph $G = (V, E)$. The nodes set V represents the literals of the assignments made by implications or decisions. Each directed hyperedge $E \subseteq 2^V \times V$ represents an implication, caused by an antecedent clause.

Conflict Learning

Conflict learning[8] can significantly boost performance of modern SAT solver.

While solving SAT instance, when a conflict arises, SAT solver will analyze implication graph to construct learned clauses, and insert these clauses into clause database. These clauses record the information of searched state space, to prevent them from being searched again.

So after SAT solver terminates,there are two types of clauses in clause database:

1. **Origin clauses** are those clauses inserted into clause database before SAT solver start running.
2. **Learned clauses** are those clauses generated by conflict analysis.

2.2 Bounded Model Checking

We first define the Kripke structure:

Definition 1 (Kripke structure). *Kripke structure is a tuple M=(S,I,T,L), with a finite set of states S, the set of initial states $I \subseteq S$, a transition relation between states $T \subseteq S \times S$, and the labeling of the states $L : S \to 2^{AP}$ with atomic propositions set AP.*

Bounded Model Checking (BMC)[7] is a model checking technology that consider only limited length path. We call this length k as the bound of path. Let S_i and S_{i+1} be the state of the i-th and (i+1)-th cycle, and $T(S_i, S_{i+1})$ represents the transition relation between them.

Assume q is a boolean proposition, and the safety assertion under verification is "$G\ q$", then the goal of BMC is to find a state S that violates q, that is to say, $\neg q \in L(S)$. In the remainder of this paper, $\neg q$ will be denoted by P, and we will not refer to q any more.

Let P_i be P at i-th cycle ,then BMC problem can be expressed as:

$$F := I(S_0) \wedge \bigwedge_{i=0}^{k-1} T(S_i, S_{i+1}) \wedge \bigwedge_{i=0}^{k-1} \neg P_i \wedge P_k$$

BMC always searches for shortest counterexample, so $\bigwedge_{i=0}^{k-1} \neg P_i$ always holds true. Thus, we can remove it from above equation, and obtain following equation:

$$F := I(S_0) \wedge \bigwedge_{i=0}^{k-1} T(S_i, S_{i+1}) \wedge P_k \tag{2}$$

Reduce equation (2) into propositional satisfiability problem, and solve it with SAT solver, then a counterexample can be found if it exists.

2.3 BFL Algorithm and Its Shortcoming

BFL algorithm proposed by K Ravi[5] can eliminate much more free variables than all existing algorithms, often up to 70% free variables can be eliminated.

Lets first define some terminology below:

Definition 2 (Assignment Clause). *Assume the value of variable v in counterexample is denoted by $Value(v) \in \{0,1\}$, then the Assignment Clause of v is a unit clause which contain only one literal, which is defined as:*

$$Assign(v) := \begin{cases} \{v\} & \text{if } Value(v)==1 \\ \{\neg v\} & \text{otherwise} \end{cases} \tag{3}$$

Definition 3 (Free Variables Set). *Assume the set of input variables is W, then input variables set of i-th cycle is denoted by W_i. Assume the set of state variables is X, then state variables set of i-th cycle is denoted by X_i.*

Assume the bound of counterexample is k, then the set of free variables is $Free := X_0 \cup \bigcup_{i=0}^{k} W_i$.

Obviously,*Free* includes input variables at all cycle and initial state variables. "Free" means that they can take on any value independent of others.

For a free variable $v \in Free$, v is an irrelevant variable if and only if the following statement holds true: "no matter what value do v take on, it can't prevent the counterexample from happening. That is to say, it can't prevent P_k of equation (2) from equal to 1". Formal definition of irrelevant variable is presented below:

Definition 4 (Irrelevant Variable). *for $v \in Free$, v is irrelevant variable iff:*

$$\neg\exists c \in \{0,1\}.(\bigwedge_{i=0}^{k-1} T(S_i, S_{i+1}) \wedge (v \Leftarrow c) \wedge \bigwedge_{v' \in Free \setminus v} Assign(v') \wedge \neg P_k) \tag{4}$$

Convert $\bigwedge_{i=0}^{k-1} T(S_i, S_{i+1}) \wedge \bigwedge_{v' \in Free \setminus v} Assign(v') \wedge \neg P_k$ into SAT instance, then v is irrelevant variable iff this SAT instance is unsatisfiable.

Thus, the BFL algorithm that extracts minimization set from counterexample is shown below:

Algorithm 1: BFL Counterexample Minimization Algorithm

1. $F" = \bigwedge_{i=0}^{k-1} T(S_i, S_{i+1}) \wedge \neg P_k$
2. for each $v \in Free$
3. $\quad F' = F" \wedge \bigwedge_{v' \in Free \setminus v} Assign(v')$
4. $\quad$ if(SAT_Solve(F')==UNSATISFIABLE) $Free = Free \setminus v$
5. $Free$ is the minimization set

We introduce 2 definitions here to make it easier to describe our algorithm:

Definition 5 (Model Clause Set). *In step 3 of algorithm 1, the clauses set generated from F" is called Model Clause Set.*

Definition 6 (Assignment Clause Set). *In step 3 of algorithm 1, the clauses set generated from $\bigwedge_{v' \in Free \setminus v} Assign(v')$ is called Assignment Clause Set.*

We call *F" Model Clause Set* because it represents inverted model checking problem of equation (2). We call $\bigwedge_{v' \in Free \setminus v} Assign(v')$ *Assignment Clause Set* because it is used to assign to all variables their value in counterexample, except v. SAT solver will assign these values to them by performing BCP.

3 Counterexample Minimization with Unit Core Extraction

As stated before, the key to reduce time overhead of BFL is to eliminate multiple variables after every call to SAT solver. So we present our key ideas below:

> In algorithm 1, when SAT instance F' is unsatisfiable, a variables set R that is sufficient to lead to conflict can be extracted from it by unit core extraction. Then $F" \wedge \bigwedge_{v' \in R} Assign(v')$ is an unsatisfiable clause subset of F'. Thus $Free - R$ can be eliminated immediately. In this way, we can achieve our goal of eliminating multiple variables simultaneously.

In this section, we first describe the overall algorithm flow in subsection 3.1, and then describe the most important part– unit core extraction algorithm in subsection 3.2. We will prove its correctness in subsection 3.3. At last, we will analyze the complexity of this algorithm in subsection 3.4.

3.1 Overall Algorithm Flow

Run time overhead of BFL is very large due to one call to SAT solver per candidate variable to be eliminated. Therefore, it is very important to reduce the number of calls to SAT solver.

Overall flow of our algorithm is shown by algorithm 2:

Algorithm 2 BFL with unit core extraction

1. $F" = \bigwedge_{i=0}^{k-1} T(S_i, S_{i+1}) \wedge \neg P_k$
2. for each $v \in Free$
3. $\quad F' = F" \wedge \bigwedge_{v' \in Free \setminus v} Assign(v')$
4. $\quad$ if(SAT_Solve(F')==UNSATISFIABLE)
5. $\quad\quad$ **R=Unit_Core(v)**
6. $\quad\quad$ **Free = Free $\cap$ R**
7. $Free$ is the minimization set

Compare it to algorithm 1, step 5 and 6 of algorithm 2 are newly inserted steps, which are highlighted with bold font. In step 5, we perform unit core extraction to extract the variables set R that lead to UNSATISFIABLE. And then in step 6, we eliminate all variables not belong to R, then we don't need to call SAT solver for them any more. Thus, the number of calls to SAT solver is significantly decreased in our approach compared to BFL.

3.2 Unit Core Extraction Algorithm

As stated by last subsection, we perform unit core extraction algorithm to extract the variable set R that lead to UNSATISFIABLE. Main idea of our unit core extraction algorithm are presented below:

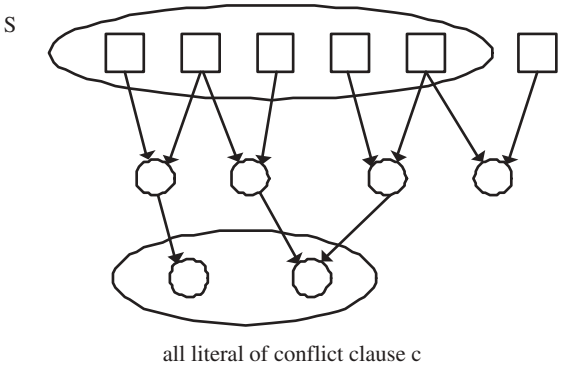

Fig. 1. Implication graph starting from the unit clauses at the leaves and ending with the conflict clause c at the root.

For SAT instance F', let F" be its model clause set, and A be its assignment clause set. After SAT solver finished running, let C be its learned clause set.

Refer to last paragraph of section 2.2 of L.Zhang's famous paper about unsatisfiable core extraction [16],we have the following theorem 1.

Theorem 1. *If F' is unsatisfiable, then there must be a conflict clause at decision level 0, we denote it by c. Because the decision level is 0, so there are no decided variables, any variables can only take on their value by implication.*

According to this theorem,there must be an implication graph starting from the unit clauses at the leaves and ending with the conflict clause c at the root. We show this implication graph in figure 1.Every rectangle is an unit clause, and S is the set of unit clauses that make all literals of conflict clause c to be false. Staring from clause c, we can traverse the implicate graph in reverse direction, to obtain the set of unit clauses S that lead to conflict. We denote the assignment clauses in S by $S \cap A$, then the variables set that lead to conflict is $R = \{v | Assign(v) \in S \cap A\}$. This is the key idea of Unit Core Extraction.

Now we present the unit core extraction algorithm below.

Algorithm 3 Unit Core Extraction Unit_Core(v)

1. set $S = \phi$;
2. queue $Q = \phi$;
3. for each literal $l \in c$
4. push antecedent clause of l into Q
5. mark antecedent clause of l as visited
6. while(Q is not empty)
7. cls=pop first clause from Q
8. if(cls is unit clause)
9. $S = S + \{cls\}$
10. if(cls is a learned unit clause) return $R = Free \setminus v$

11. else
12. for each literal $l \in cls$
13. assume $ante(l)$ is antecedent clause of l
14. if($ante(l)$ has not being visited before)
15. push $ante(l)$ into Q
16. mark $ante(l)$ as visited
17. return $R = \{v | Assign(v) \in S \cap A\}$

There is a special case in step 10 of algorithm 3. When cls is a learned clause with only one literal, we can't backtrack further because the SAT solver has not record the clauses involved in resolution to generate learned clause cls. In this case, we abandon the effort to extract unit core, and just return $R = Free \setminus v$. This means that we can eliminate only one variable v in this case.

Fortunately, we has not met with this special case in our experiments. But we just can't prove its absence in theory. Currently, I think it is because of the 1UIP conflict learning mechanism of zchaff, which may never generate learning clause with only one literal.

The unsatisfiable core extraction approaches[16,17] do provide a mechanism to record the clauses involved in resolution to generate learned clause. This mechanism do help to eliminate more irrelevant variables, but it imposes very large time overhead, which will outweigh the benefit of unit core extraction.

3.3 Correctness of Algorithm

Theorem 2. *$F" \wedge \bigwedge_{cls \in S} cls$ is an unsatisfiable clause subset of F'*

Proof. It is obvious that $F" \wedge \bigwedge_{cls \in S} cls$ is a clause subset of F', so we only need to prove that $F" \wedge \bigwedge_{cls \in S} cls$ is unsatisfiable.

Assume $C' \subseteq C$ is the set of conflict clauses met with by algorithm 3 while traverse the implication graph. Then according to algorithm 3 and figure 1, $F" \wedge \bigwedge_{cls \in S} cls \wedge \bigwedge_{cls \in C'} cls$ is unsatisfiable. Then if we can remove $\bigwedge_{cls \in C'} cls$ from it, and still retain its unsatisfiability?

For every learned clause $cls \in C'$, assume $NU(cls)$ and $U(cls)$ are non-unit clauses set and unit clauses set that involved in resolution to construct cls. Then it is obvious that $F" \wedge \bigwedge_{cls \in S} cls \wedge \bigwedge_{cls \in C'} (\bigwedge_{cls' \in U(cls)} cls' \wedge \bigwedge_{cls' \in NU(cls)} cls')$ is unsatisfiable.

It is obvious that $NU(cls) \subseteq F"$. And according to [8], unit clauses never involve in resolution, so $U(cls)$ is empty set. Thus we can remove $\bigwedge_{cls \in C'} (\bigwedge_{cls' \in U(cls)} cls' \wedge \bigwedge_{cls' \in NU(cls)} cls')$ from $F" \wedge \bigwedge_{cls \in S} cls \wedge \bigwedge_{cls \in C'} (\bigwedge_{cls' \in U(cls)} cls' \wedge \bigwedge_{cls' \in NU(cls)} cls')$, and still reatin its unsatisfiability.

So $F" \wedge \bigwedge_{cls \in S} cls$ is unsatisfiable.

Thus this theorem is proven.

Theorem 3. *$F" \wedge \bigwedge_{v' \in R} Assign(v')$ is an unsatisfiable clause subset of F'*

Proof. It is obvious that $R \in Free \setminus v$, so $F" \wedge \bigwedge_{v' \in R} Assign(v')$ is clause subset of $F' = F" \wedge \bigwedge_{v' \in Free \setminus v} Assign(v')$.

So we only need to prove that $F" \wedge \bigwedge_{v' \in R} Assign(v')$ is unsatisfiable.

According to step 17 of algorithm 3, $\bigwedge_{v' \in R} Assign(v')$ is equal to $\bigwedge_{cls \in S \cap A} cls$.

So we only need to prove that $F" \wedge \bigwedge_{cls \in S \cap A} cls$ is unsatisfiable.

According to theorem 2, $F" \wedge \bigwedge_{cls \in S} cls$ is unsatisfiable, and it can be rewritten as $F" \wedge \bigwedge_{cls \in S \cap A} cls \wedge \bigwedge_{cls \in S - A - F"} cls$.

We discuss it in 2 aspects:

1. if $S - A - F"$ is an empty set, then $F" \wedge \bigwedge_{cls \in S} cls$ and $F" \wedge \bigwedge_{cls \in S \cap A} cls$ are of the same, then $F" \wedge \bigwedge_{cls \in S \cap A} cls$ is unsatisfiable.
2. otherwise, $S - A - F"$ isn't an empty set. In this case, algorithm 3 will meet with a learning clause with only one literal. According to step 10 of algorithm 3,it will abandon the effort to extract unit core, and eliminate only one variable v. According to step 3 of algorithm 2, it is obvious that $F" \wedge \bigwedge_{v' \in R} Assign(v')$ is unsatisfiable.

Thus this theorem is proven.

According to theorem 3, Assigning to all variables in R their value in counterexample can make F' unsatisfiable. Thus according to definition 3, variables in $Free - R$ are all irrelevant variables. No matter what value do they take on, they can't prevent the counterexample. Thus, we can eliminate $Free - R$ simultaneously in step 6 of algorithm 2.

3.4 Complexity Analysis

Because our algorithm depends heavily on SAT solver, so we don't analyze its complexity directly. Instead, we compare our algorithm with SAT solver.

We first analyze space complexity of our algorithm. Comparing algorithm 2 and 1, the only difference is that algorithm 2 add an unit core extraction step. Therefore, difference of space complexity between them resides in unit core extraction algorithm. We know that the space overhead of unit core extraction mainly resides in set S and queue Q. Lets analyze them as below:

- We add a tag to each clause in clause database of SAT solver, to indicate that if this clause belongs to set S. Therefore, space overhead of S is linear to size of clause database.
- For queue Q, it may contain learned clauses. Because conflict analysis algorithm of SAT solver also need to perform similar implicate graph traversing, so space overhead of Q is not larger than that of SAT solver. We will present the peak size of Q in table 3 of experiment result, it is obvious that its size are much smaller than clause database.

Next, we will analyze the time complexity of our algorithm.

In algorithm 3, the most complex part is the if statement in step 14. For every clauses that has been in Q, this if statement will be run once. Because the size of Q is much smaller than clause database, so time overhead of algorithm 3 is much smaller than that of BCP in SAT solver.

In algorithm 2, one call to unit core extraction algorithm will eliminate many irrelevant variables, thus prevent them from calling SAT solver. This will significantly decrease the number of calling SAT solver and time overhead.

We will present the number of calls to unit core extraction algorithm and SAT solver in table 3 of experiment result.

4 Incremental SAT

From step 3 of algorithm 2, it is obvious that F' of two consecutive iterations are very similar. This suggests a good chance to share learned clause between them by employ incremental SAT approach.

In last paragraph of section 6, K Ravi[5] has mentioned that BFL's performance can be improved by Incremental SAT. But he hasn't presented how to achieve this. And all his experiments are based on non-incremental SAT. So we present here a novel approach to improve BFL's performance further by incremental SAT.

For two consecutive iterations, assume the two variables to be eliminated are $v1$ and $v2$. Then for the first iteration, $F' = F" \wedge \bigwedge_{v' \in Free \setminus v1} Assign(v')$. For the second iteration, $F' = F" \wedge \bigwedge_{v' \in Free \setminus v2} Assign(v')$. After we have finished solving the first F', to obtain the second one, we only need to delete $Assign(v1)$ and insert $Assign(v2)$ into clause database.

N. Een[12] concludes that: when delete a unit clause that contains only one literal, all learned clauses can be safely kept in clause database.

So when we delete $Assign(v1)$, we don't need to delete any learned clauses. Thus all learned clauses can be shared between consecutive iterations.

Therefore, we modify algorithm 2 into the following algorithm 4. All new steps are highlight with bold font.

Algorithm 4 BFL with unit core extraction and Incremental SAT

1. $F" = \bigwedge_{i=0}^{k-1} T(S_i, S_{i+1}) \wedge \neg P_k$
2. **Insert** $F' = F" \wedge (\bigwedge_{v' \in Free} Assign(v'))$ **into clause database**
3. for each $v \in Free$
4. **delete** $Assign(v)$ **from clause database**
5. if(SAT_Solve(F')==UNSATISFIABLE)
6. R=Unit_Core(v)
7. $Free = Free \cap R$
8. **For each** $v' \in Free - R$
9. **delete** $Assign(v')$ **from clause database**
10. **else**
11. **Insert** $Assign(v)$ **into clause database**
12. $Free$ is the minimization set

In step 8 and 9 of algorithm 4, according to N. Een's conclusion [12] stated above, we can simply delete all such $Assign(v')$, and no need to delete any learned clauses.

Thus, This mechanism improves performance in 2 aspects:

1. Learned clauses can be share between similar instances, to avoid searching overlapped state space repeatedly.
2. No need to waste time on deleting learned clauses.

5 Experiment Result

K Ravi[5] only presents the circuits that used to generate counterexample, but has not presented the assertion used. Therefore, we can't compare our result with his one directly. So we implement K Ravi's two-stages algorithm and ours in zchaff[10], such that we can compare them with same circuits and assertions.

We use NuSMV[9] to generate deep counterexample in the following way:

1. Perform a symbolic simulation to generate a state sequence $S_0, ..., S_k$.
2. Use "S_k can not be reach" as an assertion, and put it into BMC package of NuSMV[9] to obtain a counterexample shorter than k.

We perform counterexample minimization with K Ravi's two-stages algorithm[5] and our algorithm. The timeout limit is set to 10000 seconds.

5.1 Experiment Result of K Ravi's Two Stages Approach

Because K Ravi's approach includes two stages, so we first present its result in table 1. The 1st column are the circuits used to generate counterexample. The 2nd column presents the length of counterexample. The 3rd column presents number of free variables.

The 4th column is the number of variables eliminated by first stage of K Ravi's approach, the 5th column is the run time overhead of first stage.

The 6th column is the number of variables eliminated by BFL, the second stage of K Ravi's approach, The 7th column is run time overhead of BFL.

According to table 1, most irrelevant variables are eliminated by first stage, with little run time overhead. But to further eliminate more irrelevant variables, the highly expensive BFL must be called. The run time overhead of BFL is 2 to 3 orders of magnitude larger than that of first stage.

In the last 2 rows of table 1, K Ravi's approach run out of time limit. To obtain the data in the 6th column, we incorporate incremental SAT into BFL, but without unit core extraction.

5.2 Comparing Result of Our Approach and That of K Ravi

Experiment result of our approach and that of K Ravi are presented in table 2. The 1st column are the circuits used to generate counterexample. The 2nd column presents the length of counterexample. The 3rd column present number of free variables.

Table 1. Experiment Result of K Ravi's Two Stage Approach.

Circuits	CE length	Free Vars	first stage		second stage	
			Eliminated Vars	Run time	Eliminated Vars	Run time
s1512	21	667	601	0.07	5	5.097
s1423	24	483	325	0.07	75	92.443
s3271	15	507	398	0.911	34	38.946
s3384	13	743	596	0.08	19	29.01
s3330	6	373	279	0.03	16	2.613
s5378	10	530	344	0.08	72	12.698
s9234	7	362	169	0.09	57	16.984
s13207	22	1352	977	0.581	132	4080.34
s38584	14	1621	1008	2.592	61	>10000
s38417	14	2029	909	1.365	71	>10000

Table 2. Experiment Result.

Circuits	CE length	Free Vars	Result of K Ravi[5]		Result of our approach		
			Eliminated Vars	Run time	Eliminated Vars	Run time	Speedup
s1512	21	667	606	5.167	606	3.45	1.50
s1423	24	483	400	92.513	397	7.12	12.99
s3271	15	507	432	39.857	432	6.11	6.52
s3384	13	743	615	29.09	615	9.61	3.02
s3330	6	373	295	2.643	295	1.77	1.49
s5378	10	530	416	12.778	411	8.21	1.56
s9234	7	362	226	17.074	226	10.36	1.65
s13207	22	1352	1109	4080.921	1093	153.92	26.51
s38584	14	1621	1069	>10000	1069	682.19	>10
s38417	14	2029	980	>10000	981	947.84	>10

The 4th column is the number of irrelevant free variables eliminated by the two stages of K Ravi's algorithm[5]. run time of the two stages of K Ravi's algorithm is shown in 5th column.

The 6th column is the number of irrelevant free variables eliminated by our approach. run time of our algorithm is shown in 7th column. The speedup compared to K Ravi's algorithm is shown in last column.

5.3 Run Time Statistics of Our Approach

In table 3, we present some run time statistics of our algorithm:

The first column is the name of circuits. The variables and clauses number of CNF files are presented in 2nd and 3rd column. Their number of free variables are presented in 4th column, the variable eliminated by unit core extraction in step 7 of algorithm 4 are presented in 5th column. The numbers of UNSATISFIABLE

Table 3. Run Time Statistics.

Circuits	Vars	Clauses	Free Vars	Eliminated by Unit_Core	Number of UNSAT	Number of SAT	Peak size of Q
s1512	14858	39735	667	601	5	61	397
s1423	16565	44248	483	369	29	85	736
s3271	21769	59656	507	416	16	75	448
s3384	19452	50353	743	596	19	128	458
s3330	6935	17322	373	278	17	78	321
s5378	16180	42415	530	387	24	119	484
s9234	18291	49555	362	173	53	136	581
s13207	107079	284839	1352	1025	68	259	1999
s38584	237756	661828	1621	1005	64	552	5119
s38417	211653	576324	2029	910	71	1048	7703

instances are presented in the 6th column. The numbers of SAT instances are presented in 7th column. The peak size of Q is presented in last column.

Relationship between these columns is:

4th column=5th column+6th column+7th column

5.4 Conclusion About Experiment Result

From these tables, we can conclude that:

1. In most case, our approach run much faster than K Ravi's algorithm;
2. According to last column of table 2, it is obvious that the more complex the counterexample, the higher the speedup. For the three most complex counterexample:s13207, s38584 and s38417, our approach is 1 order of magnitude faster than K Ravi's algorithm.
3. Our approach achieves this speedup without any lost in its ability to eliminate irrelevant variables;
4. From 5th column of table 3, most variables are eliminated by unit core extraction, and don't need to run SAT solver for them any more;
5. Compare last column of table 3 to 3rd column, the size of Q are much smaller than that of clause database.

6 Related Works

Our work are somewhat similar to SAT solution minimization of SAT-based image computation[2–4].

Ken.McMillan's approach [3] needs to construct an alternating implication graph rooted at input variables. With this graph, he eliminates irrelevant variables from SAT solution.

Hyeong-Ju Kang[4] assigns lower decision priority to next state variables, such that when the transition relation is satisfied, as many as possible next state variables are undecided.

P Chauhan[2] employs an ATPG-like approach to analyze the dependence relation between input variables and transition relation. And try to eliminate as many as possible next state variables from final solution.

Minimization of counterexamples is useful in the context of abstraction-refinement[1, 15]. Refinement is often more effective when it is based on the simultaneous elimination of a set of counterexamples rather than on elimination of one counterexample at a time.

There are also other approaches to minimize counterexample.

Jin[6] presents a game-based technique that partitions an error trace into fated segments, controlled by the environment attempting to force the system into an error, and free segments, controlled by the system attempting to avoid the error.

P. Gastin[13] proposes a length minimization approach for explicate state model checker SPIN, which tries to generate shorter counterexample.

Alex Groce[14] proposes a value minimization approach for C language. His approach tries to minimize the absolute value of typed variables of C language.

7 Conclusion

To make the counterexample easier to be understood, irrelevant variables must be eliminated. At the same time, minimized counterexamples can significantly improve the performance of many important verification algorithm.

K Ravi's algorithm is the most effective counterexample minimization algorithm. However, its time overhead is too large.

Therefore, we propose a faster counterexample minimization algorithm in this paper. Our algorithm is 1 order of magnitude faster than K Ravi's algorithm without any lost in its ability to eliminate irrelevant variables;

In this paper we only due with path like counterexample of safety assertion, we would also like to address minimization of loop-like counterexample in future work.

References

1. E. Clarke, A. Gupta, J. Kukula, and O. Strichman. SAT based abstraction-refinement using ILP and machine learning. In E. Brinksma and K. G. Larsen, editors, Fourteenth Conference on Computer Aided Verification (CAV 2002), pages 265-279. Springer-Verlag, July 2002.LNCS 2404.
2. Pankaj Chauhan, Edmund M. Clarke, Daniel Kroening, Using SAT based Image Computation for Reachability Analysis. technology report CMU-CS-03-151, School of Computer Science ,Carnegie Mellon University, September 2003
3. K. L. McMillan. Applying SAT methods in unbounded symbolic model checking. In E. Brinksma and K. G. Larsen, editors, Fourteenth Conference on Computer Aided Verification (CAV'02), pages 250-264. Berlin, July 2002. LNCS 2404.
4. Hyeong-Ju Kang and In-Cheol Park,SAT-Based Unbounded Symbolic Model Checking,In Proceeding of DAC 2003,Anaheim, California, USA, June 2-6, 2003.

5. Kavita Ravi and Fabio Somenzi. Minimal Assignments for Bounded Model Checking. In Tenth International Conference on Tools and Algorithms For the Construction and Analysis of Systems (TACAS'04),pages 31-45 , 2004. LNCS 2988.
6. H.Jin, K.Ravi,and F.Somenzi. "Fate and free will in error traces". In 8th International Conference on Tools and Algorithms For the Construction and Analysis of Systems(TACAS 2002), pages 445-458, 2002.LNCS 2280.
7. A. Biere, A. Cimatti, E.M. Clarke, M. Fujita, Y. Zhu . "Symbolic Model Checking using SAT procedures instead of BDDs".In Proceedings of the 36th Conference on Design Automation(DAC1999).pages 317-320, 1999.
8. L.Zhang, C.Madigan, M.Moskewicz, and S.Malik. Efficient conflict driven learning in a Boolean satisfiability solver. ICCAD 2001.
9. A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani and A. Tacchella. "NuSMV 2: An OpenSource Tool for Symbolic Model Checking". In 14th International Conference on Computer Aided Verification(CAV 2002),pages 359-364 , Copenhagen, Denmark, July 27-31, 2002.LNCS 2404
10. M. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In Proceedings of the Design Automation Conference, pages 530-535,Las Vegas, NV, June 2001.
11. http://www.cbl.ncsu.edu/CBL_Docs/iscas89.html
12. N. Een and N. Sorensson. Temporal Induction by Incremental SAT Solving. In Proc. of the First International Workshop on Bounded Model Checking, 2003.
13. P. Gastin, P. Moro, and M. Zeitoun. Minimization of counterexamples in spin.In SPIN Workshop on Model Checking of Software, pages 92-108, 2004.
14. Alex Groce , Daniel Kroening. Making the Most of BMC Counterexamples. the second international workshop on Bounded Model Checking(BMC 2004), to appear
15. Marcelo Glusman, Gila Kamhi, Sela Mador-Haim, Ranan Fraer, and Moshe Y. Vardi,Multiple-Counterexample Guided Iterative Abstraction Refinement: An Industrial Evaluation. In 9th International Conference on Tools and Algorithms For the Construction and Analysis of Systems(TACAS 2003)
16. L. Zhang and S. Malik. Validating sat solvers using an independent resolution-based checker: Practical implementations and other applications. In Proceedings of Design Automation and Test in Europe (DATE2003),2003.
17. E. Goldberg and Y. Novikov. Verification of proofs of unsatisfiability for cnf formulas. In Proceedings of Design Automation and Test in Europe (DATE2003),2003.