A Faster Counterexample Minimization Algorithm Based on Refutation Analysis

ShengYu Shen
School of Computer Science
National University of
Defence Technology
syshen@nudt.edu.cn

Ying Qin
School of Computer Science
National University of
Defence Technology
qy123@nudt.edu.cn

SiKun Li
School of Computer Science
National University of
Defence Technology
skli@nudt.edu.cn

Abstract

It is a hot research topic to eliminate irrelevant variables from counterexample, to make it easier to be understood. The BFL algorithm is the most effective counterexample minimization algorithm compared to all other approaches. But its time overhead is very large due to one call to SAT solver for each candidate variable to be eliminated. The key to reduce time overhead is to eliminate multiple variables simultaneously. Therefore, we propose a faster counterexample minimization algorithm based on refutation analysis in this paper. We perform refutation analysis on those UNSAT instances of BFL, to extract the set of variables that lead to UNSAT. All variables not belong to this set can be eliminated simultaneously as irrelevant variables. Thus we can eliminate multiple variables with only one call to SAT solver. Theoretical analysis and experiment result shows that, our algorithm can be 2 to 3 orders of magnitude faster than existing BFL algorithm, and with only minor lost in counterexample minimization ability.

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 active research [12, 8, 5, 7].

At the same time, many important verification algorithms need to analyze counterexample (or witness). Some of them are: abstraction-refinement model checking [4, 6] and SAT based image computation [2, 10, 9]. In these algorithms, if we can extract a subset of variables that are sufficient to lead to counterexample, then these variables can express a large number of counterexamples, not just the individual one generated by model checker. In the remainder of this paper, we call this variable subset as **minimization set**, and call the algorithm that extract minimization set as **counterexample minimization**.

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

> For AND gate $z = a\&b$, let the assertion be "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 $\{a \Leftarrow 0, b \Leftarrow 1, z \Leftarrow 0\}$. We need nine bits to store these counterexamples.
>
> However, from an intuitive viewpoint, $a \Leftarrow 0$ is sufficient to lead to counterexample. So b is an irrelevant variable in this case. At the same time, $b \Leftarrow 0$ is also a sufficient condition of counterexample. So a is also an irrelevant variable when b equal to 0. Then we can minimize above three counterexamples, and obtain 2 minimization sets: $\{a \Leftarrow 0, z \Leftarrow 0\}$ and $\{b \Leftarrow 0, z \Leftarrow 0\}$. We only need four bits to store them.

McMillan [10] and Chauhan et al.[2] point out that , directly storing and processing the minimization set, can be exponent efficient than storing and processing the corresponding set of counterexamples.

Thus, a minimized counterexample is easier to be understood, and will significant boost the performance of many verification algorithms.

Ravi and Somenzi [12] propose a counterexample minimization algorithm called Brute Force Lifting algorithm. We will refer to it as BFL in the remainder of this paper. For every free variable v, BFL construct a SAT instance SAT(v), to determine if v can prevent the counterexample. If result of SAT(v) is UNSAT, then v is irrelevant to counterexample, and can be eliminated. Ravi compares BFL with other counterexample minimization approaches, and concludes that BFL is the most efficient one, it can often eliminate up to 70% free variables.

1530-1591/05 $20.00 © 2005 IEEE

However, the time complexity of BFL is much larger than all other existing approaches due to one call to SAT solver per candidate variable to be eliminated.

So the key to reduce time overhead of BFL is to eliminate multiple variables after every call to SAT solver

Therefore, we propose a faster counterexample minimization algorithm based on refutation analysis in this paper, which employ refutation analysis for UNSAT instances of BFL, to extract all relevant variables and eliminate all irrelevant variables simultaneously.

We implement our algorithm based on zchaff [11] and NuSMV [3], and perform experiment on ISCAS89 benchmark suite. Theoretical analysis and experiment result shows that, our algorithm can be 2 to 3 orders of magnitude faster than BFL algorithm, and with only minor lost in counterexample minimization ability.

The remainder of this paper is organized as follows. Section 2 presents background material. Section 3 presents the counterexample minimization algorithm based on refutation analysis. Section 4 presents experiment result of our algorithm and compare it to that of BFL. Section 5 reviews related works. Section 6 concludes with a note on future work.

2. Preliminaries

2.1. Bounded Model Checking

We first define the Kripke structure:

Definition 1 Kripke structure is a 6-tuple $M = (S, I, W, T, A, L)$, with a finite set of states S, the set of initial states $I \subseteq S$, the input variable set W, transition relation between states $T : S \times W \times S \rightarrow \{0, 1\}$, and the labeling of the states $L : S \rightarrow 2^{AP}$ with atomic propositions set AP.

Bounded Model Checking (BMC) [1] is a model checking technology that considers only limited length path. We call this length as the bound of path. We denote the state of the i-th and i+1-th cycle as S_i and S_{i+1}, and transition relation between them as $T_i(S_i, W_i, S_{i+1})$, with input variable set of i-th cycle denoted by W_i.

Then we can unfold the transition relation k times, and obtain the following equation:

$$[[M]]_k = I \wedge \bigwedge_{0 \leq i < k} T_i(S_i, W_i, S_{i+1}) \qquad (1)$$

Let the safety assertion under verification be $ASSERT$, the goal of BMC is to find a state S that violate $ASSERT$, that is to say, $\neg ASSERT \in L(S)$. In remainder of this paper, we denote $\neg ASSERT$ as P, and will not refer to $ASSERT$ any more.

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

$$F = [[M]]_k \wedge \bigwedge_{0 \leq i < k} \neg P_i \wedge P_k$$

Because BMC always searches for shortest counterexample, so $\bigwedge_{0 \leq i < k} \neg P_i$ always holds true. Therefore we can reduce above equation into following equation (2):

$$F = [[M]]_k \wedge P_k \qquad (2)$$

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

2.2. BFL Algorithm

BFL algorithm proposed by Ravi and Somenzi [12] can eliminate much more free variables than all existing algorithm, often up to 70% free variables can be eliminated.

We give some terminology below:

Definition 2: Assume the bound of counterexample is k, and denote the set of free variables as $Free = I \cup \bigcup_{0 \leq i \leq k} W_i$. The assignment to variable v in counterexample is denoted by $Assign(v)$, the assignment to variable set V in counterexample is denoted by $Assign(V) = \{Assign(v) | v \in V\}$.

Obviously, the set $Free$ includes input variables at all cycle and initial state variables.

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

Definition 3 Irrelevant Variable: for $v \in Free$, v is an irrelevant variable iff:

$$\forall c \in \{0, 1\}.[[M]]_k \wedge (v \Leftarrow c) \wedge A \rightarrow P_k$$

which is equal to

$$\neg \exists c \in \{0, 1\}.[[M]]_k \wedge (v \Leftarrow c) \wedge A \wedge \neg P_k \qquad (3)$$

where

$$A = Free - \{v\} \Leftarrow Assign(Free - \{v\}) \qquad (4)$$

From equation (3) and (4) we can conclude that: v is irrelevant variable iff SAT instance $[[M]]_k \wedge A \wedge \neg P_k$ is Unsatisfiable.

Thus, the BFL algorithm [12] that extracts minimization set from counterexample is show below:

Algorithm 1:BFL Algorithm

1 $F" = [[M]]_k \wedge \neg P_k$

2 foreach $v \in Free$
  a) $F' = F" \wedge (Free - \{v\} \Leftarrow Assign(Free - \{v\}))$
  b) if(SAT_Solve(F')==UNSAT)
    i $Free = Free - \{v\}$
3 $Free$ is the minimization set

To make it more distinct, we give the following two definitions:

Definition 4 Model Clause Set: all clauses generated from F" in step 2a) of algorithm 1.

Definition 5 Assignment Clause Set: all clauses generated from $(Free - \{v\} \Leftarrow Assign(Free - \{v\}))$ in step 2a) of algorithm 1

We call the former as model clause set, because $F"$ represents inverted model checking problem of equation (2). We call the latter as assignment clause set because they are used for assigning to all variables their value in counterexample, except v. For every $v' \in Free - \{v\}$, its assignment clause contain only one literal. If $Assign(v') == 1$, then assignment clause of v' is $\{v'\}$, otherwise it is $\{\neg v'\}$. SAT solver will assign these values to them by BCP when solving this instance.

3. Counterexample Minimization with Refutation Analysis

As stated before, the key to reduce time overhead of BFL is to eliminate multiple variables after every call to SAT solver.

In algorithm 1, when SAT instance F' is Unsatisfiable, a variable subset $R \subseteq Free$ can be extract from it by refutation analysis, which is the sufficient condition of counterexample. Then all variables in $Free - R$ can be eliminated immediately. Thus we can eliminate multiple variables in this way.

In this section, we first describe the overall algorithm flow in subsection 3.1, and then describe the most important part - refutation analysis 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

Overall flow of our algorithm is shown by algorithm 2.

Algorithm 2 BFL algorithm with refutation analysis
1 $F" = [[M]]_k \wedge \neg P_k$
2 foreach $v \in Free$
  a) $F' = F" \wedge (Free - \{v\} \Leftarrow Assign(Free - \{v\}))$
  b) if(SAT_Solve(F')==UNSAT)
    i $\mathbf{R} = \mathbf{Refutation\_Analysis}()$
    ii $\mathbf{Free} = \mathbf{Free} \cap \mathbf{R}$
    iii **goto step 3**
3 $Free$ is the minimization set

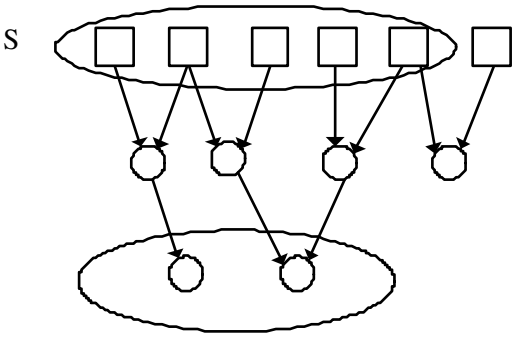

Figure 1. implication graph root at unit clauses and target at all literals of conflict clause c

Compare it to algorithm 1, steps in 2b) of algorithm 2 are newly inserted steps, which are highlighted with bold font. In step 2b)i, we perform refutation analysis to extract the variables set R that leads to UNSAT. And then in step 2b)ii, we eliminate all variables not belong to R. After that we terminate the whole algorithm by going to step 3 directly.

3.2. Refutation Analysis

As stated by last subsection, we perform refutation analysis to extract the variable set R that leads to UNSAT.

For SAT instance F' of algorithm 2, we denote its model clause set by $F"$, and its assignment clause set by A. After SAT solver finished running, denote its conflict clause set as C.

Refer to last paragraph of section 2.2 of L.Zhang's famous paper about unsatisfiable core extraction [13], 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.

With this theorem, it is obvious that there must be an implication graph root at unit clauses and target at all literals of conflict clause c. We show this implication graph in figure 1.Every rectangle denote a unit clause, and S is the set of unit clauses that makes all literals of conflict clause c to be 0.

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 set of variables that lead to con-

flict is $R = \{v|\{v\} \in S \cap A\} \cup \{v|\{\neg v\} \in S \cap A\}$. This is the key idea of our refutation analysis algorithm.

Now we present the refutation analysis algorithm below.

Algorithm 3 Refutation Analysis

1 set $S = \emptyset$
2 queue $Q = \emptyset$
3 foreach literal $l \in c$
  a) push antecedent clause of l into Q
  b) mark antecedent clause of l as visited
4 while(Q is not empty)
  a) cls=pop first clause from Q
  b) if(cls is a unit clause)
    i $S = S + \{cls\}$
    ii If(cls is a learned unit clause)
      return $R = Free - \{v\}$
  c) else
    i foreach literal $l \in cls$
      1 assume $ante(l)$ is antecedent clause of l
      2 if($ante(l)$ has not being visited)
        a) push $ante(l)$ into Q
        b) mark $ante(l)$ as visited
5 $R = \{v|\{v\} \in S \cap A\} \cup \{v|\{\neg v\} \in S \cap A\}$ are variables that lead to UNSAT

There is a special case in step 4b)ii, when cls is a learned unit clause, we can't backtrack further because the SAT solver has not record the clauses involved in resolution to construct cls. In this case, we abandon the effort to extract R, and simply return $R = Free - \{v\}$. This means that we can eliminate only one variable v in this case.

Fortunately, we have not met with this special case in experiments. I suspect that this is because of the 1UIP conflict learning mechanism, which seldom generate learned unit clause.

3.3. Correctness Proof

We prove the correctness of algorithm 2 and 3 with following theorems:

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 it is Unsatisfiable.

Assume $C' \subseteq C$ is the set of learned clauses met with by algorithm 3 while traversing implication graph. Thus, $F" \wedge \bigwedge_{cls \in S} cls \wedge \bigwedge_{cls \in C'} cls$ 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.

It is obvious that $NU(cls) \subseteq F"$. And according to [13], unit clauses never involve in resolution, so $U(cls)$ is empty set. So we can remove $\bigwedge_{cls \in C'} cls$ from $F" \wedge \bigwedge_{cls \in S} cls \wedge \bigwedge_{cls \in C'} cls$, and still retain its unsatisfiability

Thus this theorem is proven.

Theorem 3: $F" \wedge \bigwedge_{v' \in R}(v \Leftarrow Assign(v'))$ is an Unsatisfiable clause subset of F'

Proof: it is obvious that $F" \wedge \bigwedge_{v' \in R}(v \Leftarrow Assign(v'))$ is an clause subset of F'. Thus we only need to prove that $F" \wedge \bigwedge_{v' \in R}(v \Leftarrow Assign(v'))$ is Unsatisfiable.

According to algorithm 3, $\bigwedge_{v' \in R}(v \Leftarrow Assign(v'))$ 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, which can be rewritten as $F" \wedge \bigwedge_{cls \in S \cap A} cls \wedge \bigwedge_{cls \in S - A - F"} cls$.

Lets discuss it in two aspects:

1. If $S - A - F"$ is empty set, then $F" \wedge \bigwedge_{cls \in S \cap A} cls$ is Unsatisfiable
2. Otherwise, $S - A - F"$ isn't empty set. In this case, algorithm 3 will meet with a learned unit clause. According to step 4b)ii of algorithm 3, it will abandon the effort to extract R, and eliminate only one variable v. In the case, $F" \wedge \bigwedge_{cls \in S \cap A} cls$ is Unsatisfiable

Thus this theorem is proven.

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.

Lets first analyze space complexity of our algorithm. Comparing algorithm 2 and 1, the only difference is that algorithm 2 adds a refutation analysis step. Therefore, difference of space complexity between them resides in refutation analysis algorithm. We know that the space overhead of refutation analysis 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 conflict clauses. Because conflict analysis algorithm of SAT solver also need to perform similar implicate graph traversing, so space overhead due to Q is not larger than that of SAT solver.

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

In algorithm 3, the most complex part is the if statement in step 4c)i2. for every clause that has been in Q, this statement will be run once. Because the size of Q is smaller than

clause database, so time overhead of algorithm 3 is smaller than that of conflict analysis.

In step 2b) of algorithm 2, one call to refutation analysis algorithm will eliminate many irrelevant variables, thus prevents them from calling SAT solver. This will significant reduce time overhead.

4. Experiment Result

Ravi and Somenzi [12] only presents the circuits that used to generate counterexample, but has not presented the assertion used. Therefore, we can't compare our algorithm with his one directly. So we implement his algorithm and ours in zchaff [11], such that we can compare them with same circuits and assertions.

We use NuSMV [3] to generate deep counterexample in the following way:

1. Perform reachability analysis to generate state sequence $\{S_0, \ldots, S_k\}$.
2. Use "S_k can't be reached" as an assertion, and put it into bounded model checking package of NuSMV [3], we can obtain a counterexample with length smaller than k.

We perform counterexample minimization with BFL [12] and ours. The timeout limit is set to 20000 seconds.

Experiment result is presented in table 1. The 1st column is 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 shows the numbers of irrelevant free variables eliminated by K Ravi's BFL [12]. Divide this column with the 3rd column, we can get the minimization rate shown in 5th column. Run time of BFL is shown in 6th column.

The numbers of irrelevant free variables eliminated by our algorithm are shown in the 7th column. Divide this column with the 3rd column, we can get the minimization rate shown in 8th column. Run time of it is shown in 9th column. The speedup compared to BFL is shown in last column.

From this table, we can conclude that:

1. As shown by last column, our algorithm can be 2 to 3 orders of magnitude faster than BFL;
2. Compare the 4th and 7th column, the number of irrelevant variables eliminated by BFL and our algorithm are of almost the same.

5. Related Works

Our work is somewhat similar to that of SAT-based image computation [2, 10, 9].

McMillan [10] propose an SAT solution minimization approach, to extract a partial assignment from SAT solution. His approach needs to construct an alternating implication graph root at input variables. With this graph, he eliminates many irrelevant variables from SAT solution.

Kang and Park [9] assign lower decision priority to next state variables, such that when the transition relation is satisfied, as many as possible next state variables are undecided.

Chauhan et al. [2] employ 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 [4, 6]. 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.

Gastin et al [5] propose an length minimization approach for explicate state model checker SPIN, which try to generate smaller counterexample with respect to their length.

Groce and Kroening [7] propose an value minimization approach for CBMC tools, which target at bounded model checking of C language. His approach tries to minimize the absolute value of typed variables of C language.

6. Conclusions

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

BFL is the most effective counterexample minimization algorithm. However, its time overhead is too large.

Therefore, we propose a faster counterexample minimization algorithm based on refutation analysis in this paper. Our algorithm can be 2 to 3 orders of magnitude faster than BFL, and with only minor lost in minimization ability.

In this paper we only due with safety assertion, we would also like to address minimization algorithm for loop like counterexample of liveness property in future work.

7. Acknowledgements

Supported by the National Natural Science Foundation of China under Grant No. 90207019; the National High Technology Development 863 Program of China under Grant No. 2002AA1Z1480

Circuits	CE length	Free Vars	Result of K Ravi[12]			Result of our approach			
			Eliminated Vars	Min rate	Run time	Eliminated Vars	Min rate	Run time	Speedup
s1512	21	667	606	90.85%	27.469	601	90.10%	0.05	549
s1423	24	483	397	82.19%	28.381	369	76.40%	0.07	405
s3271	15	507	432	85.21%	41.239	398	78.50%	0.88	47
s3384	13	743	615	82.77%	45.676	579	77.92%	0.08	570
s3330	6	373	295	79.08%	7.931	279	74.80%	0.03	264
s5378	10	530	416	78.49%	27.68	344	64.91%	0.06	461
s9234	7	362	226	62.43%	21.771	169	46.69%	0.07	331
s13207	22	1352	1109	82.03%	619.163	1017	75.22%	0.14	4422
s38584	14	1621	1069	65.95%	1830.82	1008	62.18%	0.43	4255
s38417	14	2029	980	48.29%	2096.06	909	44.80%	0.72	2911

Table 1. Experiment Result

References

[1] A. Biere, A. Cimatti, E. Clarke, M. Fujita, and Y. Zhu. Symbolic model checking using sat procedures instead of bdds. In *Proceedings of the 36th ACM/IEEE conference on Design automation(DAC1999)*, pages 317–320, New Orleans, LA, USA,, June 1999. ACM Press.

[2] P. Chauhan, E. M. Clarke, and D. Kroening. Using sat based image computation for reachability analysis. technology report CMU-CS-03-151, School of Computer Science ,Carnegie Mellon University, Pittsburgh, PA 15213, September 2003.

[3] A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebasti-ani, and A. Tacchella. Nusmv 2: An opensource tool for symbolic model checking. In E. Brinksma and K. G. Larsen, editors, *Proceeding of 14th International Conference on Computer Aided Verification(CAV 2002),LNCS 2404*, pages 359–364, Copenhagen, Denmark, July 2002. Springer-Verlag.

[4] 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, *Proceeding of Fourteenth Conference on Computer Aided Verification (CAV 2002),LNCS 2404*, pages 265–279, Berlin, July 2002. Springer-Verlag.

[5] P. Gastin, P. Moro, and M. Zeitoun. Minimization of counterexamples in spin. In S. Graf and L. Mounier, editors, *Proceeding of 11th International SPIN Workshop(SPIN2004),LNCS 2989*, pages 92–108, Barcelona, Spain, April 2004. Springer-Verlag.

[6] M. Glusman, G. Kamhi, S. Mador-Haim, R. Fraer, and M. Y. Vardi. Multiple-counterexample guided iterative abstraction refinement: An industrial evaluation. In H. Garavel and J. Hatcliff, editors, *Proceeding of 9th International Conference on Tools and Algorithms For the Construction and Analysis of Systems(TACAS 2003),LNCS 2619*, pages 92–108, Warsaw, Poland, April 2003. Springer-Verlag.

[7] A. Groce and D. Kroening. Making the most of bmc counterexamples. In *Proceeding of the second International Workshop on Bounded Model Checking(BMC2004)*, 2004.

[8] H.Jin, K.Ravi, and F.Somenzi. Fate and free will in error traces. In J.-P. Katoen and P. Stevens, editors, *Proceeding of 8th International Con-ference on Tools and Algorithms For the Construction and Analysis of Systems(TACAS 2002),LNCS 2280*, pages 445–458, Grenoble, France, April 2002. Springer-Verlag.

[9] H.-J. Kang and I.-C. Park. Sat-based unbounded symbolic model checking. In *Proceeding of the 40th Design Automation Conference, DAC 2003*, pages 840–843, Anaheim, CA, USA, June 2003. ACM Press.

[10] K. L. McMillan. Applying sat methods in unbounded symbolic model checking. In E. Brinksma and K. G. Larsen, editors, *Proceeding of Fourteenth Conference on Computer Aided Verifica-tion (CAV 2002),LNCS 2404*, pages 250–264, Berlin, July 2002. Springer-Verlag.

[11] M. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient sat solver. In *Proceedings of the 38th conference on Design automation(DAC2001)*, pages 530–535, Las Vegas, NV, June 2001. ACM Press.

[12] K. Ravi and F. Somenzi. Minimal assignments for bounded model checking. In K. Jensen and A. Podelski, editors, *Proceeding of Tenth International Conference on Tools and Algorithms For the Construction and Analysis of Systems (TACAS'04),LNCS 2988*, pages 31–45, Barcelona, Spain, March 2004. Springer-Verlag.

[13] 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)*, pages 530–535, Munich, Germany, March 2003. IEEE Computer Society.