2nd IFIP/IEEE International Symposium on Theoretical Aspects of Software Engineering

Improving Encoding Efficiency for Bounded Model Checking

Jinji Yang[1,2]
[1]School of Computer,
South China Normal University,
Guangzhou, China.
[2]Dept. of Computer Science,
Sun Yat-Sen University,
Guangzhou, China.
yangjj@scnu.edu.cn

Kaile Su[3+]
[3]Key laboratory of High Confidence Software Technologies (Peking University),
Ministry of Education.
Beijing , China.
sukl@pku.edu.cn

Qingliang Chen[4]
[4]Dept. of Computer Science,
Jinan university,
Guangzhou, China.
tsingliangchen@gmail.com

Abstract

Bounded Model Checking (BMC) has played an important role in verification of software, embedded systems and protocols. The idea of BMC is to encode Finite State Machine (FSM) and Linear Temporal Logic (LTL) verification specification into satisfiability (SAT) instances, and then to search for a counterexample via various SAT tools. Improving encoding technology of BMC can generate a SAT instance easy to solve, and therefore is essential to improve the efficiency of BMC. In this paper, we improve the encoding of BMC by combining the characteristic of FSM state transition and semantics of LTL, get a simple and efficient recursion formula which is useful to efficiently generate SAT instances. We present an efficient algorithm to encode the modal operator (safety formula) in BMC. The experiments for comparative analysis shows that this encoding algorithm is more powerful than the existing two mainstream encoding algorithms in both the scale of generated SAT instances and the solving efficiency. The methodology presented in this paper is also valuable for optimization of other modal operator encodings in BMC.

1. Introduction

Bounded Model Checking (BMC)[1] is an important branch in the field of Model Checking, which has played an important role in verification of software, embedded systems and protocols in recent years [2][3][5]. It is a new technology to compensate for the disadvantages of the typical symbolic model checking, such as the space explosion caused by bad orderings of the variables in the OBDD, which thus limit its scale for the problem. One of the advantages of BMC is that it encodes the verification into a SAT instance, and utilize the powerful and industrial level SAT solvers to solve it. As we know now, SAT solvers nowadays are much more powerful in symbolic computation than OBDD. So inducing SAT solvers into this problem may greatly promote the scale and efficiency to above some order of magnitude. Another advantage for BMC is that the counterexamples it outputs are always the shortest and simplest because of its breadth-first searching technology and thus makes it a great convenience for the system designer to rectify the bugged system. The empirical analysis shows that when the bound k is smaller than 60, BMC outperforms that typical symbolic Model Checking [4].

The main procedure of BMC is as follows. First we need to construct a Finite State Machine (FSM) for the system or model we are to verify,and thus the behaviors of the system or model are encoded into finite state transitions in this FSM. Secondly the expected specifications are expressed by the Linear Temporal Logic (LTL) such as G(p) and F(p), in their NNF forms. Thirdly we set the upper bound for the state transition of the system as K. Then the state transitions in FSM and the LTL specifications are combined by logical conjunctions as the derived BMC problem formulas. Finally we translate the BMC problem formula to a SAT instance, and then solve it by SAT solvers. It finds a counterexample with respect to the specification if a satisfiable truth assignment is found in the formula;or it justifies that system is correct with respect to the specifications up to the K steps in the state transition of the system if the formula is unsatisfiable.

Recently the improvement of BMC has been a hotspot in the formal method research community. Now the performance improvements mainly rely on three aspects. The first one is the optimization for the translation of the BMC formula from the system FSM and LTL specification [6][7][8];The second one is the optimization of variables and clauses for the translation from the BMC formula to the SAT instance [9][10];The last one is the optimization of the SAT

+: corresponding author

978-0-7695-3249-3/08 $25.00 © 2008 IEEE
DOI 10.1109/TASE.2008.23

solvers to exploit the special structure of the encoded SAT formulas so as to improve the efficiency [11]. The former two methods aim to reduce the number of variables and clauses as much as possible to produce a simple formula; the second way may potentially destroy the inherent structure of the problem while the third way may not work very well by optimizing the SAT solvers since the whole SAT solving process occupies only 20%~30% of the total time. The first method is to encode the problem into a logically equivalent -but with simple and easy structure- formulas to solve, and thus can produce a good SAT instance for further translation. This method can get a BMC formula in a short time and be mapped into a relatively small SAT instance, which results in an improvement in efficiency.

Our work in this paper belongs to the first category of the above method. we introduces a concise recursive formula to improve the encoding of modal operator G(p) , which makes use of the state transitions of a FSM and the semantics of G(p), and then we justify the correctness of this formula. Based on this formula, we give its related algorithm of encoding for G(p), followed further by the complexity analysis. Furthermore we analyze the characteristics of clauses in its SAT instances. And finally we show the validity of our work by experimental results.

This paper focuses on safety property (G(p)) in the specifications for BMC by LTL. G(p), which are very useful in verifications of software etc. It turns out that verification of safety properties which is difficult for BDD based model checking can be done with remarkable efficiency with BMC. In fact, more than 90% of errors in real system are violation of safety properties [14]. A very important type of safety property is an invariant expressed by G(p) in LTL. Most safety properties can be reduced to G(p) form [15], a property that must hold in all reachable states. Obviously, if a sequence of states can be found that begins at an initial state and ends in a state where the supposed invariant is false, that property is not an invariant.

The structure of the paper is as follows. The coming section gives some background knowledge on Bounded Model Checking and the relevant theorem. The section 3 presents the verification and optimization of the LTL specification G(p)based on the relationship of their semantics and the state transitions in FSM induced by the BMC, gets the relative algorithm, and analyzes the characteristic of the clauses in the SAT instance produced by our method. The following section 4 is the discussion of related work. The section 5 shows the experimental results and the comparisons between our skills and several existing ones. Finally we will conclude the paper in section 6.

2. Bounded Model Checking (BMC) and the relevant theorem

The specifications BMC is to check are in Linear Temporal Logic (LTL) so we need to introduce first the syntax and semantics of the LTL. And then we present the reduction details of Finite State Machine together with the expected corresponding specifications in LTL to BMC formulas. At last, some existing optimization skill for this problem is also shown.

2.1. Syntax and Semantics of LTL

Definition 1. (syntax) Let A be a set of propositional atoms , the syntax of the LTL formulas can be defined recursively as follows:

1. If $\varphi \in A$, then φ is an LTL formula;
2. If φ and ϕ are LTL formulas, then, $\neg\varphi$, $\varphi\wedge\phi$, $\varphi\vee\phi$, $G\varphi$, $F\varphi$, $X\varphi$, $\varphi U\phi$, $\varphi R\phi$ are all LTL formulas, where X, G, F, U, R are temporal modalities of next, global, eventually, until, release and $\neg$, $\wedge$, $\vee$ are logical connectives.

Because BMC is mainly used to find counterexamples, the expected LTL specifications for BMC are indeed in NNF (Negative Normal Form), in which negative only appears before the propositional atoms. If f is an LTL formula, depth(f)is defined to be the depth of f, the nested number of the modalities in f.

Definition 2. (Semantics) The Kripke structure of the BMC is the tuple M=(S,I,T,I), where S is the set of possible states in the affiliated FSM ; I is the set of initial states and $I\subseteq S$; T is the transition relation of the states and $T\subseteq S\times S$; I: $S\rightarrow P(A)$ is the evaluating function that attaches each state with a set of propositional atoms which hold in that state.

Definition 3. The path of BMC can be defined as a $\pi=(s_1,s_2,s_3,\ldots)$where $s_i\in S$, $i\in N$; $\pi(i)=s_i$; $\pi^i=(s_i,s_{i+1},s_{i+2},\ldots)$.

Definition 4.(Semantics of LTL) Let M be a Kripke structure , π a path of BMC,and f an LTL formula, then the satisfaction relation $\pi \models f$ can be defined as follows:

$\pi \models p$ iff $p\in l(\pi(0))$
$\pi \models \neg p$ iff $p\notin l(\pi(0))$
$\pi \models f\wedge g$ iff $\pi \models f$ and $\pi \models g$
$\pi \models f\vee g$ iff $\pi \models f$ or $\pi \models g$
$\pi \models G(f)$ iff $\forall i$, $\pi^i \models f$
$\pi \models F(f)$ iff $\exists i$, $\pi^i \models f$
$\pi \models X(f)$ iff $\pi^1 \models f$

Some modalities irrelevant to our paper here are not listed here and can refer to [1].

2.2. BMC Formula Transform Principles

If M is a Kripke structure, f is the LTL specification in NNF form that BMC is to check, k is the bound of the number of the state transition, then we can construct a propositional formula $[[M,f]]_k$ (called transformed BMC formula)according to the semantics of LTL, and the path π=(s0,s1,s2,…sk)is the finite state transition series of $[[M,f]]_k$. The meaning of this transformed BMC formula $[[M,f]]_k$ is that $[[M,f]]_k$ is satisfiable in the state transition series s0,s1,s2,…sk if and only if f is valid in the path π.

Definition 5. (BMC formula transformation) Let M be a Kripke, and f be the LTL specification in NNF form that BMC is to check,k be the bound of the number of the state transition,the transformed BMC formula is:

$$[[M,f]]_k = [[M]]_k \wedge [[f]]_k \quad (1)$$

where $[[M]]_k = I(s_0) \wedge \bigwedge_{i=0}^{k-1} T(s_i, s_{i+1})$,

$[[f]]_k = (\neg L_k \wedge [[f]]_k^0) \vee \bigvee_{l=0}^{k} ({}_lL_k \wedge {}_l[[f]]_k^0)$,

${}_lL_k = T(s_k, s_l)$, $L_k = \vee_{l=0}^{k} {}_lL_k$,and the recursive definition of $[[f]]_k^0$ and ${}_l[[f]]_k^0$ can be summarized in Table 1.

$[[M,f]]_k$ encodes all the paths where f holds, and the paths start in initial state s_0 with the length of k. Practically the bound k is very hard to decide ahead of the computation [12] so usually we increase the values of $k=k_0,k_0+1,k_0+2,k_0+3,\ldots$,and so on. k_0 is called lowest bound which is set to 0 in most cases;We solve $[[M,f]]_k$ case by case for different values of k , if a counterexample is found or, the expected time bound has arrived, we can stop the computation, otherwise k will increase by one and the computation goes on to the next case. In every case, k is an integer i, we can turn the formula $[[M,f]]_i$ into a SAT instance, and solve it by a SAT solver. If it is satisfiable and then a counterexample has been found, otherwise we let k be i+1 and continue. As the LTL specification f has been encoded into part of the Boolean formulas$[[M,f]]_k$, the size of f decides the size of $[[M,f]]_k$ when translated into SAT instances, which is different from BDD-based approach [1]. At the same time, the size of the SAT instance also has a lot to do with the bound k. It will be larger if k becomes more. The first method mentioned above is to optimize $[[M,f]]_k$ by finding the simplest and equivalent formulas for it, which can lead to a smaller SAT instance. If we can determine an ideal lower bound k_0 for k, we can decrease the number of calling of the SAT solvers

f	$[[f]]_k^i$	${}_l[[f]]_k^i$
p	p_i	p_i
$\neg p$	$\neg p_i$	$\neg p_i$
$h \wedge g$	$[[h]]_k^i \wedge [[g]]_k^i$	${}_l[[h]]_k^i \wedge {}_l[[g]]_k^i$
$h \vee g$	$[[h]]_k^i \vee [[g]]_k^i$	${}_l[[h]]_k^i \vee {}_l[[g]]_k^i$
Xg	$[[g]]_k^{i+1}$ *if* $i<k$ $\perp$ *otherwise*	${}_l[[g]]_k^{i+1}$ *if* $i<k$ ${}_l[[g]]_k^{l}$ *otherwise*
Gg	$\perp$	$\wedge_{j=\min(i,l)}^{k} {}_l[[g]]_k^j$
Fg	$\vee_{j=i}^{k} [[g]]_k^j$	$\vee_{j=\min(i,l)}^{k} {}_l[[g]]_k^j$

Table1. Recursive definition of $[[f]]_k^i$ and ${}_l[[f]]_k^i$

2.3. Related optimization methods for BMC formulas

Here we introduce and recall the optimization methods for BMC formulas that are related to our method in this paper. Some will be discussed in the latter section. Paper [6] has given some interesting properties of $[[f]]_k$ in $[[M,f]]_k$, which can simplify $[[f]]_k$ by logical deduction. Some relevant theorems are the following two:

Theorem 1. Formula $[[f]]_k$ is logically equivalent to $[[f]]_k^0 \vee \bigvee_{l=0}^{k} ({}_lL_k \wedge {}_l[[f]]_k^0)$.

Theorem 2. If depth(f)≤1, then ${}_l[[f]]_k^0$ is independent of l. Specially if$[[f]]_k^0 = {}_l[[f]]_k^0$ then $[[f]]_k = [[f]]_k^0$.

The details of the proof can be found in [6].

3. The encoding optimization for G(p)

3.1. The existing encoding schemes

G(p)is the most frequently used LTL modalities in the protocol verification by BMC. G(p) can be used to specify the secrecy of the protocol. Also they can be applied to verify a lot of properties for multi-agent

systems [5]. Usually p is Boolean formula and the NNF forms of G(p) is F(¬p).

As depth(F(¬p))=1, we can see in Table 1 that, $[[F(\neg p)]]_k^0 =_l [[F(\neg p)]]_k^0$. So we can get the BMC for G(p) by Equation (1) and Theorem2:

$$[[M, F(\neg p)]]_k = [[M]]_k \wedge [[F(\neg p)]]_k^0 \quad (2)$$

Although Equation (2) in [6] are well optimized formulas and thus improve the efficiency, the optimization itself only consider $[[f]]_k$ and does not consider the $[[M]]_k$. If we take $[[M]]_k$ into account and introduce some corresponding optimization, we can get the equivalent and more concise formulas according to their increasing and recursive properties. In existing BMC tools (such as NuSMV and VIS), this problem is not yet implemented and not incorporated.

In the following section, we present the logically equivalent recursive formulas for $[[M, F(\neg p)]]_k$ in Theorem 3 and the corresponding proof. Through the recursive formulas in Theorem 3 , the size of the resulting SAT instances will be much smaller than those in Equations (2) and thus can lead to significant improvement in performance for the overall BMC process.

.

3.2. The optimization scheme

Theorem 3. If $[[M, F(p)]]_k$ is unsatisfiable, then $[[M, F(p)]]_{k+1}$ is logically equivalent to $[[M]]_{k+1} \wedge (p_{k+1}^{k+1})$.

Proof:

$$[[M, F(p)]]_{k+1} = [[M]]_{k+1} \wedge [[F(p)]]_{k+1}^0$$

$$= I(s_0) \wedge \bigwedge_{i=0}^{k} T(s_i, s_{i+1}) \wedge \bigvee_{i=0}^{k+1} p_{k+1}^i$$

$$= I(s_0) \wedge \bigwedge_{i=0}^{k} T(s_i, s_{i+1}) \wedge (\bigvee_{i=0}^{k} p_k^i \vee p_{k+1}^{k+1})$$

$$= (I(s_0) \wedge \bigwedge_{i=0}^{k-1} T(s_i, s_{i+1}) \wedge \bigvee_{i=0}^{k} p_k^i \wedge T(s_k, s_{k+1}))$$

$$\vee (I(s_0) \wedge \bigwedge_{i=0}^{k} T(s_i, s_{i+1}) \wedge p_{k+1}^{k+1})$$

$$= ([[M, F(p)]]_k \wedge T(s_k, s_{k+1})) \vee ([[M]]_{k+1} \wedge p_{k+1}^{k+1})$$

As we know that the precondition is that $[[M, F(p)]]_k$ is unsatisfiable, $[[M, F(p)]]_k \wedge T(s_k, s_{k+1})$ is unsatisfiable either. If $[[M, F(p)]]_{k+1}$ is satisfiable, then $[[M]]_{k+1} \wedge (p_{k+1}^{k+1})$ is also satisfiable. If, $[[M]]_{k+1} \wedge (p_{k+1}^{k+1})$ is satisfiable, $[[M, F(p)]]_{k+1}$ is also satisfiable. This justifies that $[[M]]_{k+1} \wedge p_{k+1}^{k+1}$ is logically equivalent to $[[M, F(p)]]_{k+1}$.

3.3. The algorithm

By Theorem 3, $[[M]]_{k+1} \wedge p_{k+1}^{k+1}$ is logically equivalent to $[[M, F(p)]]_{k+1}$, so the solution of the SAT instance for $[[M, F(p)]]_{k+1}$ is exactly that for the $[[M]]_{k+1} \wedge p_{k+1}^{k+1}$ and vice versa. If the SAT instance for $[[M]]_{k+1} \wedge p_{k+1}^{k+1}$ is unsatisfiable, neither is that for $[[M, F(p)]]_{k+1}$. Since the size of the SAT instance for $[[M]]_{k+1} \wedge p_{k+1}^{k+1}$ is much smaller than that for the $[[M, F(p)]]_{k+1}$, replacing $[[M, F(p)]]_{k+1}$ by $[[M]]_{k+1} \wedge p_{k+1}^{k+1}$ can induce potential efficiency.Fig1 show the encoding algorithm based theorem 3.

```
//K is the upper bound in bounded model checking
//Solution.out:{ UNSATISFIABLE, SATISFIABLE}
//Solution.Counterexample:{NULL, a solution to a SAT instance};
begin
  Solution.out⇐UNSATISFIABLE;
  Solution. counterexample ⇐NULL;
  k⇐0;
  While (k≤K)
  begin
   G_CNF_k⇐EncodingCNF([[M]]_k ∧ ¬p_k^k );
        Solution⇐SAT(G_CNF_k);
  if(Solution.out= UNSATISFIABLE)
        k⇐k+1;
  else
    goto OK;
                //G(p) is invalid, its counterexamples
                // in Solution. counterexample.
  end;
  OK;
end.
```

Figure 1. the algorithm encoding G(p)

Now we proceed to discuss the efficiency improvement of the SAT instance by our scheme.

First we discuss the size of the SAT instances. SAT instance is a conjunction of all clauses, while it is disjunction of variables inside the clauses. In LTL formulas, the modalities G, F, X, U and the implication → can be replaced by those formulas that only contain ¬ ,

$\wedge$ and $\vee$. The resulting BMC formulas ϕ can be encoded into a SAT instance with size n(ϕ) that can be recursively defined as follows:

ϕ	n(ϕ)	$\neg$n(ϕ)
$\neg\phi_1$	$\neg$n(ϕ_1)	n(ϕ_1)
$\phi_1 \wedge \phi_2$	n(ϕ_1)+n(ϕ_2)	n(ϕ_1)n(ϕ_2)
$\phi_1 \vee \phi_2$	n(ϕ_1)n(ϕ_2)	n(ϕ_1)+n(ϕ_2)

Table 2. Recursive definition of clause number of ϕ

By Table 2, the clause number of Equation (2) is:

$$n([[M]]_k)+n([[F(\neg p)]]_k^0)=n([[M]]_k)+n(\bigvee_{i=0}^{k}\neg p_k^i)$$

$$=n([[M]]_k)+\prod_{i=0}^{k} n(\neg p_k^i) \quad (3)$$

While by Table 2 and Theorem 3, the clause number of $[[M,F(p)]]_k$ is: $n([[M]]_k)+ n(\neg p_k^i)$ (4)

Equation (3) and(4)has a same part of n($[[M]]_k$), thus we can just compare the second part. By Table 1, $n(\neg p_k^i) = n(\neg p_k^j)$ and suppose $n(\neg p_k^i)$=m, where m is an integer, so the complexity of the second part in Equation (3) is the exponential $O(m^k)$. Paper [9] has introduced a method to transform the original formulas to a satisfiability equivalent one and at the same time reduce the number of the clauses to the order of tm where t>k, by means of adding 4 times new variables and renaming. And thus the complexity is linear O(t); The second part of the Equation (4) is the constant order of O(m), and we can see from the second parts of the Equation (3) and (4) that the size has become from linear complexity to constant complexity and no more new variables are added.

Conflict clauses are produced by SAT solver when SAT solvers process the SAT instances. The more conflict clauses are produced, the more time the SAT solvers spend when they process the SAT instances in most cases. In some times, these small scale of SAT instances with more conflict clauses need much more time than those large scale of SAT instances with less conflict clauses. Let $S1_k$ be set of clauses associated with Equation (2) $[[M]]_k \wedge [[F(\neg p)]]_k^0$ and $S2_k$ be set of clauses associated with $[[M]]_k \wedge \neg p_k^k$, it is easy to see that $S2_k$ is a subset of $S1_k$. the number of conflict clauses of $S2_k$ is less than that of $S1_k$ in most cases. This can be seen in Table 4. Certainly, this is only an empirical observation, not a theoretical result.

Let $S2_{k+1}$ be set of clauses associated with $[[M]]_{k+1} \wedge \neg p_{k+1}^{k+1}$ .although $S2_k$ is not a subset of $S2_{k+1}$,but there are many characteristic of BMC between them. As $[[M]]_{k+1} = [[M]]_k \wedge T(s_k, s_{k+1})$,so the set of clauses associated with $[[M]]_k$ is a subset of the set of clauses associated with $[[M]]_{k+1}$. The variables in the formula $\neg p_k^k$ don't equal to the variables in the formula $\neg p_{k+1}^{k+1}$, but both of them have the same clause structure .

4. Related Work

This work is a further step of that in [6]. Paper [6] points out the problem in the original SAT encoding for BMC. This paper is mainly to optimize $[[f]]_k$ in the transformation formulas for BMC and give some highly efficient storage structures as well. All these ideas are implemented in the tool NuSMV2.1.2 and perfected continuously in later version. Timo, etc [8] utilize the properties of lasso-shaped Kripke Structure, and apply the fixed point techniques for model checking CTL (Computation Tree Logic) [1] to BMC formulas. Empirical analysis shows that the way in [8] is more effective than those in [6] [7]. Frisch, etc [7] represent the BMC formulas in a fixed point-theoretic way by SNF and standard form of fixed point, respectively. The standard form of fixed point takes good advantages of the properties of the standard form, and utilizes similar symbolic tableau style method. Experiments shows that the resulting SAT instances for this form are smaller than those in [6], and are also better than those in SNF.

Sheridan, etc [9][10] try to optimize the encoded SAT instances for the BMC so as to reduce the size. But they introduce new variables and renaming during encoding and thus have a lot of redundancy. Later by the ideas of Boy de la Rour, they give a compact and optimal way to reduce the SAT instance size greatly but this will as well lose some inherent structure of the BMC during encoding.

Strichman, etc [11] exploit some properties of the SAT instances for the BMC such as: the order of the variables, the conflicting clauses, etc, to optimize the SAT solving process by the SAT solvers; Similarly, Gupta, etc [13] enhance the SAT solving efficiency by extracting some structure information hidden in the SAT instances. They accomplish this by using some results from the model checking process in a pure BDD way ahead of time before the BMC, then find and convey them to the SAT solvers.

5. Experimental Results

We implement our optimization ideas in NuSMV2.3.1. We here compare our results with the original method in NuSMV2.3.1 and another wonderful method in [8]. The original method in NuSMV2.3.1, denoted as AA_BMC, is based on that in [1] and optimized in [6]; Another method is by Timo, etc and implemented in NuSMV2.4.0. We denote it as Timo_BMC. Because Timo_BMC generally outperforms other optimization schemes [6][7], we do not need to compare ours with any other ones any more. We denote our schemes for optimizing G(f)as G_BMC in the following.

The experiments are carried out on a PC equipped with INTEL core 4300 CPU, 2G memory, Windows XP Professional with winGW32 complier for NuSMV. Our experimental model is mainly from [16].

BMC has the following properties: the needed account of time will be smaller if k becomes smaller;the needed account of time will be larger if k becomes larger. So if the bound of the model k is relatively smaller then the verification is more efficient than any other ones; In our experiments, we use simple models in order to shorten the running time. If the running time is still unbearable, we just choose a smaller k. All the models contain the examples with the specifications G(p). In the following experiments, most of the specifications in the models are in CTL, we have to translate to the equivalent LTL specifications. We can see that if the resulting SAT instance has solutions within bound k, we have to adjust the LTL specifications so that it will be unsatisfiable, and thus it is easy to validate the efficiency of the algorithm.

The following tables illustrate every item we check: the number of variables and clauses in the resulting SAT instances, the overall time needed to generate and verify the SAT instances in the bound of k(time unit: second).

From the data in Table 3 we can conclude that Timo_BMC does not outperform AA_BMC in verifying G(p), while our methods G_BMC obviously outperform both of them because of less number of variables and clauses. For the case of BRP model, it is a small improvement, but for the case of SEMAPHORE model and SYNCARB5 model, the improvement is significant. In general, the more complex the formula p in G(p) is ,the more significant improvement can be achieved.

The number of every conflict clause is produced by a well-known SAT tool MINISAT [17]. From the data in Table 4, we can conclude that MINISAT doesn't produce any conflict clause when processing the SAT instance about BRP model in the three methods; the number of conflict clauses in our method is less than that of the other two methods in most cases, but with a few exception such as DME2 model.

6. Conclusion and Future Work

We in this paper have implemented to optimize the SAT encoding phase of BMC for a very important LTL modalities G(p). We have deduced a very concise and recursive formula for it and at the same time prove their equivalence to the original formulas, by the semantics of these modalities and the state transition relation of FSM. We have also analyzed the complexity of the optimized SAT instance, and justified the effectiveness of our method by concrete models in experiments.

One of our future work will point to apply our recursive ideas in this paper for optimizing other LTL modalities in BMC. In another perspective, since the SAT instances produced by our method preserve many characteristics of BMC, another future work would be to utilize those methods such as variable order, sharing conflict clause etc proposed in [11] to process these SAT instances and further improve efficiency.

Acknowledgements

We gratefully acknowledge the financial support of the National Science Foundation for Distinguished Young Scholars of China under Grant No. 60725207, the National Natural Science Foundation of China under Grant No.60473004 ; the National Grand Fundamental Research 973 Program of China under Grant No.2005CB321900, the Research Foundation of Science and Technology Plan Project in Guangdong Province of China under Grant No.2007B010400068 and the Startup Research Fund for Talents in Jinan University.

We would also like to thank Biere , Cimatti for sharing their NuSMV implementation with us.

References

[1] A. Biere, A . Cimatti , E.M. Clarke, and Y .Zhu. Symbolic model checking without BDDs. In Tools and Algorithms for the Constructions and Analysis of Systems (TACAS'99), volume 1579 of LNCS. Springer, 1999. 193–207.

[2] G. Daniel, K.Ulrich, D.Rolf. HW/SW CoVerification of Embedded Systems using Bounded Model Checking Proceedings of the 16th ACM Great

Model	K	AA_BMC			Timo_BMC			G_BMC		
		Num of Var	Num ofClau	Time	Num of Var	Num ofClau	Time	Num of Var	Num ofClau	Time
Dme2	10	6384	14727	5	7053	16347	7	6325	14547	4
	20	12519	29277	22	13727	32517	29	12305	28917	18
	23	14379	33642	30	15750	37368	41	14099	33228	25
Brp	10	4597	13294	<1	5279	15677	<1	4543	13264	<1
	35	16072	46169	16	18479	54577	19	15443	46064	15
	50	23257	65894	39	26699	77917	51	21983	65744	37
Semaphore	10	1248	1959	2	1349	2329	2	534	1404	<1
	14	2188	2759	315	2329	3277	215	742	1956	7
Syncarb5	30	3229	7023	15	3530	7923	16	2718	3153	2
	50	5819	11603	55	6320	13103	58	4498	5153	7
	70	8809	16183	132	9510	18283	136	6278	7153	15

Table 3. Comparison of SAT Instances of Encoding for G(p)

Model	K	AA_BMC		Timo_BMC		G_BMC	
		literials	clauses	literials	clauses	literials	clauses
Dme2	10	4569	602	1916	395	2217	346
	20	20738	2341	44471	3474	31513	2950
	23	44357	4341	122255	9538	61697	4680
Semaphore	10	2678	479	2396	416	2048	430
	14	8527	1282	8682	1349	4755	865
Syncarb5	30	777	365	758	342	17	10
	50	1426	607	1425	634	9	9
	70	1909	862	1804	837	7	7

Table 4. Comparisons of Conflict Clauses in SAT Instances of Encoding for G(p)

Lakes symposium on VLSI .Philadelphia, PA, USA SESSION: CAD for embedded systems 2006 . 43 - 48

[3] F. Ivancic , Z. Yang , M. K. Ganai , P. Ashar .Efficient SAT-based bounded model checking for software verification , in International Symposium on Leveraging Applications of Formal Methods (ISoLA), November 2004. 168-179

[4] N .Amla, R .Kurshan, K .McMillan, and R. Medel. Experimental analysis of different techniques for bounded model checking. In TACAS, 2003. LNCS 2619, Springer 2003. 34–48,

[5] X.Y Luo, KL Su, J.J. Yang. Bounded Model Checking for Temporal Epistemic Logic in Synchronous Multi-Agent Systems. Journal of Software, 2006, 17(12):2485−2498 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/17/2485.htm

[6] A .Cimatti, M .Pistore, M .Roveri, and R .Sebastiani. Improving the encoding of LTL model checking into SAT. In Verification, Model Checking, and Abstract Interpretation (VMCAI'2002), volume 2294 of LNCS, Springer, 2002. 196–207.

[7] A. Frisch , D .Sheridan, and T .Walsh. A fixpoint encoding for bounded model checking. In Formal Methods in Computer-Aided Design (FMCAD'2002), volume 2517 of LNCS,. Springer, 2002. 238–255

[8] T .Latvala, A . Biere., K .Heljanko ., T .Junttila .: Simple bounded LTL model checking.In: Formal Methods in Computer-Aided Design (FMCAD 2004). Volume 3312 of LNCS., Springer, 2004. 186–200

[9] P .Jackson and D .Sheridan. Clause Form Conversions for Boolean Circuits. In Theory and Appl. of Sat. Testing, 7th Int. Conf. (SAT'04), volume 3542 of LNCS, Springer 2004. 183-198

[10] P .Jackson and D .Sheridan. The optimality of a fast CNF conversion and its use with SAT. Technical Report APES-82-2004, APES Research Group, March 2004.http://www.dcs.st-and.ac.uk/~apes/apesreports.html.

[11] O .Strichman. Accelerating bounded model checking of safety properties. Formal Methods in System Design, 24(1):5–24, 2004.

[12] E.M Clarke, D . Kroenig, J . Oukanine, and O .Strichman. Completeness and complexity of bounded model checking. In Verification, Model Checking, and Abstract Interpretation (VMCAI'2004), volume 2937 of LNCS, Springer, 2004. 85–96.

[13] A .Gupta, M .Ganai, C .Wang, Z .Yang, and P .Ashar. Learning from BDDs in SAT-based bounded model checking. In Proceedings of the 40th Conference on Design Automation, IEEE, 2003. 824–829.

[14] A. Podelski, B. Steffen, and L. Zuck. *Liveness Manifestos. Beyond Safety, International Workshop, Schloß Ringberg, Germany, April 25–28, 2004*.http://www.cs.nyu.edu/acsys/beyond-safety/liveness.htm.

[15] I. Beer, S. Ben-David, and A. Landver, "On-the-fly model checking of RCTL formulas," in A.J. Hu and M.Y.Vardi (Eds.), *Proc. 10th Intl. Conference on Computer Aided Verification (CAV'98)*, Vol. 1427 of Lect. Notes in Comp. Sci., Springer-Verlag, 1998, pp. 184–194.

[16]http://nusmv.irst.itc.it/examples/examples.html

[17]http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat/MiniSat.html