Incremental Semantic LTL Bounded Model Checking

Rui Wang Wanwei Liu Xiaoguang Mao Tun Li
School of Computer Science
National University of Defense Technology
Changsha, Hunan
rui.wang@nudt.edu.cn

***Abstract*—Bounded model checking has proven to be an efficient method for finding bugs in system designs. In this paper, we present an incremental semantic translation for Bounded model checking and give an incremental algorithm. We implement this method in NuSMV model checker and report encouraging results.**

Keywords-bounded model checking; semantic encoding; incremental; NuSMV

I. INTRODUCTION

Bounded Model Checking (BMC) [1] is an efficient method for finding bugs in system designs. The basic idea of BMC is to convert the model checking problem to a Boolean satisfiability problem, i.e., given a Kripke structure M representing the system, an LTL formula φ and a bound k, a propositional formula $[\![M, \varphi]\!]_k$ is created and it is satisfiable if and only if M involves a counterexample of φ, and the length of such counterexample is bounded by k.

BMC has established itself as a complementary method to BDD [2] based approach. There are some advantages for BMC:

- First, it leverages the impressive achievements in SAT solver technology and several industry applications have proven that BMC seems to be able to solve certain problems that are not feasible for BDDs [4].
- Counterexample produced by most BMC encodings is minimal and that counterexample is immediately available [3]. In comparison, producing short counterexample with BDD-based method is much more expensive than with BMC.
- Lastly, Boolean formulas are a more compact representation than using BDDs for many Boolean functions. Bryant pointed that there are some Boolean functions whose corresponding BDDs are exponential large in the number of propositional variables that still have polynomial circuits [2].

Although BMC has been successfully applied in practice, verifications cannot always be accomplished within desired resources. One of the key factors that affect the efficiency of BMC is the size of the inputting of the SAT solver. If the encoding produces large formulas that the solver can be overwhelmed, then BMC will not be able to proceed deep enough to find all the bugs in the given model. As a result, it is still a meaningful work to improve the scalability of BMC.

There are two major ways to improve the efficiency of BMC. One is to develop better encodings, and the other is to equip with a more powerful SAT solver. In detail, the first approach is to develop a better approach to encoding $[\![M, \varphi]\!]_k$, the more compact the encoding is, the faster for solving it by SAT solvers.

The second way is to utilize new SAT solver technology 0. Incremental SAT solving is a promising technique to improve the performance of BMC. When a solver is faced with a sequence of related problems, learning clauses from the previous problem can drastically improve the solution time for the next problem and thus for the whole sequence 0. In BMC, when the SAT solver does not find bug with the present bound k, then it would repeat this process by augmenting k, unless k reaches some specific value (called completeness threshold). The fact that the encodings for $[\![M, \varphi]\!]_k$ and $[\![M, \varphi]\!]_k$ are very similar makes BMC a natural candidate for incremental solving.

In this paper, we present a BMC encoding technique, which is the incremental version of semantic encoding proposed in [5]. We adopt the new encoding to utilize the incremental SAT solver as a back-end. And, we give an implementation based the NuSMV [7]. Experimental results show that the incremental semantic encoding makes a satisfactory improvement in BMC efficiency.

Some work has also considered improving the efficiency of BMC encoding [9, 10]. Cimatti et al. [9] analyze the original encoding [1] and suggest several optimizations. Frisch et al. [10] propose a fixpoint based encoding which uses a normal form of LTL and it takes advantage of the properties of such normal form. Both 0 and [11] propose encodings with linear complexity, and we will do a comparison with them in section IV.

This paper is structured as follows: Section II gives the basic definitions and terminology used throughout the paper. Section III will give the optimized semantic encoding for BMC. Experimental results will be given in Section IV. Finally, Section V will conclude the whole paper.

II. PRELIMINARIES

Let AP be a finite set of atomic propositions. A *Kripke structure* M is a tuple $\langle S, I, T, L \rangle$ where S is a finite set of states; $I \subseteq S$ is the set of the initial states; $T \subseteq S \times S$ is a total transition relation; and a mapping $L : S \to 2^{AP}$ is the labeling function. Labeling is a way to attach observations to the system: For a state $s \in S$, the set $L(s)$ contains exactly those atomic propositions that hold in s.

Published by Atlantis Press, Paris, France.

We use $p(s)$ to denote $p \in L(s)$. The initial state I and the transition relation T are given as functions in terms of AP. This kind of representation, frequently called *functional form*, can be exponentially more succinct in comparison to an explicit representation of the states. An *infinite path* derived from M is a sequence $\pi = s_0 s_1 \ldots \in S^{\omega}$ for which $s_0 \in I$ and $(s_i, s_{i+1}) \in T$ $(i \geq 0)$ for each i. We say that π is a (k, l)-loop if $\pi = (s_0 s_1 \ldots s_{l-1})(s_l \ldots s_k)^{\omega}$ such that $0 \leq l \leq k$. We use the notation $\pi(i)$ to denote the i-th state s_i. We denote by $L(M)$ the set of all the infinite paths derived from M.

LTL is an extension of propositional logic. It can be defined inductively over a set of atomic propositions AP as follows:

- Both $\top$ and $\bot$ are LTL formulae.
- Each proposition $p \in AP$ is an LTL formula.
- If both f and g are LTL formulae, then both $\neg f$ and $f \wedge g$ are LTL formulae.
- If f is an LTL formula, then $X f$ is an LTL formula.
- If both f and g are LTL formulae, then $f U g$ is also an LTL formula.

Semantics of an LTL formula is given w.r.t an infinite path π of M and a position $i \in N$:

- $\pi, i \models p$ iff $p \in L(\pi(i))$
- $\pi, i \models \neg f$ iff $\pi, i \not\models f$
- $\pi, i \models f \wedge g$ iff $\pi, i \models f$ and $\pi, i \models g$
- $\pi, i \models X f$ iff $\pi, i+1 \models f$
- $\pi, i \models f U g$ iff there is some $j \geq i$, s.t. $\pi, j \models g$ and $\pi, t \models f$ for each $i \leq t < j$

As usual, we write $\pi \models \varphi$ in place of $\pi, 0 \models \varphi$.

We will also use the derived Boolean operators such as $\vee$, $\rightarrow$ and $\leftrightarrow$, defined as usual. Some derived temporal operators defined as follows are also used:

$$F f \equiv \top U f$$
$$G f \equiv \neg F \neg f$$
$$f R g \equiv \neg(\neg f U \neg g)$$

Given a Kripke structure M and an LTL formula φ, we denote $M \models \varphi$ if $\pi \models \varphi$ for each $\pi \in L(M)$.

Theorem 1: If $M \not\models \varphi$, then there exists a (k, l)-loop $\pi \in L(M)$ violating φ for some k and l, where $l \leq k$.

III. INCREMENTAL SEMANTIC ENCODING FOR BMC

In this section, we introduce the incremental semantic encoding for LTL BMC. But before that, we will revisit the original syntactic and semantic encoding, and then give the improved approach.

A. The basic syntactic/semantic encoding

Given a model M and an LTL formula φ, the syntactic encoding [1] approach for BMC generates the Boolean formula $[\![M, \varphi]\!]_k$ for M and φ with bound k separately. The whole encoding can be expressed as follows:

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

where the left two conjuncts are usually denoted as $[\![M]\!]_k$ which represents all the paths with bound k in M, and $[\![\varphi]\!]_k$ represents the paths of length k that violates φ. There are several known methods for generating $[\![\varphi]\!]_k$ [1, 5, 9, 10, 11].

Recall the method proposed in [8], LTL model checking problem could be done as follows: Given an LTL formula φ, we can construct its *tableau* $T_{\neg\varphi}$, which accepts exactly all paths that violate φ. Then, the model checking problem of LTL is boiled down to check whether $L(M \times T_{\neg\varphi})$ is empty. To this end, one need to show that there is no infinite fair path in $M \times T_{\neg\varphi}$. Hence, the so-called semantic encoding for LTL BMC is given in [5].

The fair path detection problem can be straightforward adapted to a SAT-based BMC procedure. Given a model M and an LTL formula φ, we can do this by verifying the property $G \top$ under the fairness constraint $\bigwedge_{F_i \in F_M} F_i$ in the product model $M \times T_{\neg\varphi}$ instead of the standard BMC translation. Thus, the semantic encoding can be formulated as follows:

$$I_M(s_0) \wedge \bigwedge_{i=0}^{k-1} T_M(s_i, s_{i+1}) \wedge \bigvee_{l=0}^{k-1} ((s_l = s_k) \wedge \bigwedge_{F_i \in F_M} \bigvee_{j=l}^{k} F_i(s_j))$$

In comparison to syntactic encoding, the semantic encoding has a big advantage in terms of the size of resulting formula [5].

B. Optimization of the semantic encoding

In order to apply the incremental SAT technique in the semantic encoding, we need to do some optimization on the basic encoding. We separate the whole encoding into three parts, the first part is the model's k-step unrolling, the second part is to handle the loop constraints and the third part is the fairness constraints.

The model's k-step unrolling can be encoded as follows:

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

which captures all the paths of length k in the product model. For loop constraints, we introduce $k+1$ fresh loop selector variables $\ell_0, \ldots, \ell_k$ which constraints the path in the model to be a (k, i)-loop if the variable ℓ_i is true and all other ℓ_j variables is false. Then, the loop constraints can be captured as follows:

$$[\![loopconstraints]\!]_k = \begin{pmatrix} \bigvee_{0 \leq i \leq k} (\ell_i \wedge (s_{i-1} = s_k)) \\ \wedge \bigwedge_{1 \leq i \leq k} (\bigwedge_{0 \leq j \leq i-1} (\ell_j \rightarrow \neg \ell_i)) \end{pmatrix}$$

From the formula above, we can see that $[\![loopconstraints]\!]_k$ is true if and only if there is a (k, l)-loop. However, the $[\![loopconstraints]\!]_k$ above is k-dependent. In order to make use of the k step information to compute the $k+1$ step, we would introduce a new proxy state s_p to denote the endpoint of the path. Then, the $[\![loopconstraints]\!]_k$ will be separated into two parts: k-dependent and k-invaraint. We use $[\![loopconstraints]\!]_{kd}$ to denote the k-dependent part and $[\![loopconstraints]\!]_{ki}$ to

denote the k-invaraint part. The new loop constraints encoding should be:

$$[\![loopconstraints]\!]_{ki} = \begin{pmatrix} \bigwedge_{0 \le i \le k} ((\neg l_i) \rightarrow (s_{i-1} = s_p)) \\ \bigwedge_{1 \le i \le k} (\bigvee_{0 \le j \le i-1} \neg l_j) \end{pmatrix}$$

$$[\![loopconstraints]\!]_{kd} = (s_p = s_k)$$

$$[\![loopconstraints]\!]_k = [\![loopconstraints]\!]_{ki} \wedge [\![loopconstraints]\!]_{kd}$$

Thus, many k dependent information for loop constraints is eliminated by this new encoding. The fairness constraints will be formulated as usual. Then, the whole encoding about the optimized semantic encoding will be:

$$[\![M, \varphi]\!]_k = \begin{pmatrix} [\![M]\!]_k \wedge [\![loopconstraints]\!]_{ki} \\ \wedge [\![loopconstraints]\!]_{kd} \wedge \bigvee_{0 \le l \le k} \\ (l_i \wedge \bigwedge_{F_i \in F} \bigvee_{l \le j \le k} F(s_j)) \end{pmatrix}$$

The encoding above only introduces $k + 2$ variables without changing the complexity of the semantic encoding. However, this encoding will be more suitable to be solved with the incremental SAT solver and we will give the algorithm in the next section.

C. The incremental algorithm

From section B, we achieve the optimized semantic encoding. In this section, we will give the incremental algorithm for the optimized semantic encoding. The complete algorithm is depicted in Figure 1.

In the algorithm, we use the tableau construction method proposed in [8] to generate the LTL tableau. zChaff [12] is used as the incremental SAT solver. The solver has two instance groups: the permanent group and the additional group. The permanent group is to record the k steps' invariant information which is used to compute the next step. The additional group is to store the k-dependent information.

The function ConstructKInvaraint returns the k step invariant which is the conjunction of the k step unrolling of the product model and the k step loop condition invariant. ConstructKDependent returns the k step dependent information which is the disjunction of all the loop condition with finesses and it will be deleted at the end of the current step computation.

IV. Experimental Results

To justify our idea, we implement the algorithm in NuSMV2.5.4 [7]. We mostly use examples of nontrivial complexity. All the benchmarks come from [3]. Our implementation is compared against four bounded LTL model checking algorithms. Firstly we compare against the standard NuSMV encoding, which includes many of the optimizations of [9]. Secondly, we compare our encoding to [11], denoted as "VMCAI2005" in the table. Then, we also compare against the encoding of 0, denoted as"CAV2005" in the table. To the best of our knowledge, these two encodings are the most efficient method for bounded model checking. At last, we compare against the basic semantic encoding without using the optimization and the incremental SAT technology which is denoted as "BMC_TAB" in the table.

We ran the benchmarks on Linux OS and the memory for each run was set to 1.5GB and the time limit to 1 hour. For all BMC algorithms, we used zChaff as the SAT solver. Table 1 shows the experimental results. The "*a*" columns tell whether the property was found to be *true* (*t*) or *false* (*f*). The running times are given in seconds except that "TO" ("MO") means that the formula was not solved because of timeout (running out of memory). The "*k*" columns give the bound that was reached. From the results, we can see that our algorithm performs better than others in most cases.

```
Input: Model: M , an ltl formula: φ, bound: k.
Output: true if there is a counterexample, false otherwise.
Procedure BMC(M, φ, k)
 begin
1.  solver := CreateIncrementalSovler();
2.  T_φ := ConstructLTLTableau(φ);
3.  M := M ⊗ T_φ;
4.  AddToPermanentGroup(solver, I_M);
5.  for increasingK := 0 to k
6.    if(pre_increasingK < increasingK)
7.      k_invaraint:= ConstructKInvaraint(M,
                pre_increasingK, increasingK);
8.      AddToPermanentGroup(solver,
                                k_invaraint);
9.      pre_increasingK := increasingK;
10.   endif
11.   k_dependent := ConstructKDependent(M,
                increasingK);
12.   AddToAdditionalGroup(solver, k_dependent);
13.   if(Solve(solver) is satisfiable)
14.     return true;
15.   else
16.     coninute;
17.   endif
18.   DestroyAdditionalGroup(solver);
19. endfor
20. return false;
21. end
```

Figure 1. Algorithm: Bounded model checking ltl based on incremental semantic encoding

V. Conclusion

In this paper, we present an optimized semantic encoding for bounded model checking LTL. The new encoding uses an incremental SAT solver as the backend and we give an incremental algorithm. Experimental results show that our algorithm is more efficient.

TABLE I. EXPESRIMENTAL RESULTS

model	Prop.	NuSMV			VMCAI2005			CAV2005			BMC_TAB			BMC_INC		
		a	***k***	***t***	***a***	***k***	***t***	***a***	***k***	***t***	***a***	***k***	***t***	***a***	***k***	***t***
abp4	0	f	16	62	f	16	46	f	16	27	f	16	24	**f**	**16**	**2**
	: 0		47	TO		52	TO		354	TO		378	TO		**496**	**TO**
	1		30	TO		29	TO		45	TO		38	TO		**51**	**TO**
	2	f	17	70	f	17	36	f	17	39	f	17	5	**f**	**17**	**1**
	3		29	TO		30	TO		37	TO		33	TO		**45**	**TO**
brp	0		31	TO		241	TO		3040	TO		786	TO		**2798**	**TO**
	: 0	f	1	0	f	1	0	f	1	0	f	1	0	**f**	**1**	**0**
	: 0, nv		22	TO		21	TO		24	600	f	25	65	**f**	**25**	**19**
	1		25	TO		38	TO		196	TO		76	TO		**205**	**TO**
	: 1	f	1	0	f	1	0	f	1	0	f	1	0	**f**	**1**	**0**
counter	0		202	TO		1263	MO		11849	TO		1567	TO		**12116**	**TO**
	: 0	f	8	0	f	8	0	f	8	0	f	8	0	**f**	**8**	**0**
pci	0		17	TO	f	18	3092	f	18	1339	f	18	435	**f**	**18**	**102**
	: 0	f	0	0	f	0	0	f	0	0	f	0	0	**f**	**0**	**0**
	F 0		14	TO	f	18	1121	f	18	514	f	18	60	**f**	**18**	**3**
	1		16	TO		18	TO		20	TO		21	TO		**28**	**TO**
	: 1	f	1	0	f	1	0	f	1	0	f	1	0	**f**	**1**	**0**
dme3	0		27	MO		49	TO		48	TO	f	63	1021	**f**	**63**	**307**
	: 0	f	1	0	f	1	0	f	1	0	f	1	0	**f**	**1**	**0**
	: 0, nv		27	MO	f	59	2330	f	59	641	f	59	403	**f**	**59**	**170**
	1		42	TO		53	TO		58	TO		67	TO		**103**	**TO**
	: 1	f	1	0	f	1	0	f	1	0	f	1	0	**f**	**1**	**0**
srg5	0		13	TO		312	TO		805	MO		732	TO		**863**	**TO**
	: 0	f	1	0	f	1	0	f	1	0	f	1	0	**f**	**1**	**0**
	: 0, nv	f	6	8	f	6	0	f	6	0	f	6	0	**f**	**6**	**0**

ACKNOWLEDGMENT

This work was supported by the National Natural Science Foundation of China under Grant No.90818024, 91118007 and 61133001, the National High Technology Research and Development Program of China (863 program) under Grant No.2011AA010106 and Program for New Century Excellent Talents in University.

This work is partially supported by NSFC project under Grant No. 61103012, No. 61133007 and No. 61272335.

REFERENCES

[1] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu, “Symbolic model checking without bdds”, Proc. International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), volume 1579 of Lecture Notes in Computer Science, Springer-Verlag, 1999, pp 193-207.

[2] R. E. Bryant, “Graph-based algorithms for boolean function manipulation”, IEEE Transactions on Computers, C-35(8), 1986, pp. 677-691.

[3] A. Biere, K. Heljanko, T. Junttila, T. Latvala, and V. Schuppan, “Linear encodings of bounded ltl model checking”, Logical Methods in Computer Science, 2006.

[4] N. Amla, X. Du, A. Kuehlmann, R. Kurshan, and K. McMillan, “An analysis of satbased model checking techniques in an industrial environment”, Correct Hardware Design and Verification Methods, 2005, pp. 254-268.

[5] K. Heljanko, T. Junttila, and T. Latvala, “Incremental and complete bounded model checking for full pltl”, Proc. Computer Aided Verification, Springer, 2005, pp. 517-527.

[6] E. Clarke, D. Kroening, J. Ouaknine, and O. Strichman, “Completeness and complexity of bounded model checking”, Proc. VMCAI’04, volume 2937 of Lecutre Notes in Computer Science, pp. 85-96, Springer-Verlag, 2004.

[7] R. Cavada, A. Cimatti, C. A. Jochim, G. Keighren, E. Olivetti, M. Pistore, M. Roveri, and A. Tchaltsev, “Nusmv 2.5 user manual”, http://nusmv.fbk.eu/NuSMV/userman/v25/nusmv.pdf, Apr. 2010.

[8] E.M. Clarke, O. Grumberg, and K. Hamaguchi, “Another look at LTL model checking”, Proc. CAV’94, volume 818 of Lecture Notes in Computer Science, pp. 415-427, Springer-Verlag, 1994.

[9] Cimatti, A., Pistore, M., Roveri, M., Sebastiani, R, “Improving the encoding of LTL model checking into SAT”, Proc. Verification, Model Checking, and Abstract Interpretation (VMCAI’2002). Volume 2294 of LNCS., Springer (2002), pp. 196-207.

[10] A. Frisch, D. Sheridan, T. Walsh, “A fixpoint encoding for bounded model checking”, Proc. FormalMethods in Computer-Aided Design (FMCAD’2002), Volume 2517 of LNCS., Springer (2002), pp. 238-255.

[11] T. Latvala, A. Biere, K. Heljanko, and T. Junttila, “Simple is better: Efficient bounded model checking for past LTL”, Proc. VMCAI’05, volume 3385 of LNCS, pp. 380-395, Springer, jan. 2005.

[12] M.W. Moskewicz, C.F. Madigan, Y. Zhao, and S. Malik, “Chaff: Engineering an efficent sat solver”, Proc. Design Automation Conference (DAC2001), volume 208, pp. 530-535. Springer-Verlag, 2001.