2013 International Symposium on Theoretical Aspects of Software Engineering

Optimizing the SAT Decision Ordering of Bounded Model Checking by Structural Information

Liangze Yin, Fei He and Ming Gu
Key Laboratory for Information System Security, Ministry of Education
Tsinghua National Laboratory for Information Science and Technology
School of Software, Tsinghua University, Beijing 100084, PR China
Emails: yinliangze@163.com, {hefei, guming}@tsinghua.edu.cn

***Abstract*—This paper considers bounded model checking for extended labeled transition systems. Bounded model checking relies on a SAT solver to prove (or disprove) the existence of a counterexample with a bounded length. During the translation of a BMC problem to a SAT problem, much useful information is lost. This paper proposes an algorithm to analyze the transition system model, and then utilize the structure information hidden in the model to refine the decision ordering of variables in SAT solving. The basic idea is to guide the search process of SAT solving by the structure of the transition system. Experiments with this heuristic on real industrial designs show 5-12 times speedup over standard bounded model checking.**

I. Introduction

Bounded model checking (BMC) [1] has been widely accepted as a complement to BDD-based model checking, and has achieved great successes in a large number of industrial cases. It limits the search to a bounded length, and reduces the model checking problem to a propositional satisfiability problem, which can be solved by a SAT solver. The performance of BMC depends heavily on the efficiency of the SAT solver.

Most works on BMC are conducted on the state machine model. However, for specifying sophisticated behaviors of embedded systems, an event-based formalism is a necessity. This paper considers BMC for extended labeled transition systems. Of course an extended labeled transition system can be translated to a state machine, and then the BMC problem can be encoded similarly as a SAT problem. However, during such translation, much useful information stored in the transition system is lost.

Consider an extended labeled transition system shown in Fig. 1, which consists of 4 locations and a data variable x ranging from 0 to 15. Assume that we are about to verify the property $AG(x! = 15)$, and the search length is 8, the BMC problem is formulated as a SAT problem and then fed to a modern SAT solver (minisat2-070221). It takes the SAT solver up to 156 variable decisions[1] to give the result. However, by looking at the model, only at location s_0 is there a fork. So we need only choose branch at location s_0. For any path of length 8, 3 decisions should be enough.

Encoding of the BMC problem takes 6 Boolean variables (2 for encoding the locations, and 4 for encoding the data). The SAT solver treats all these Boolean variables alike when

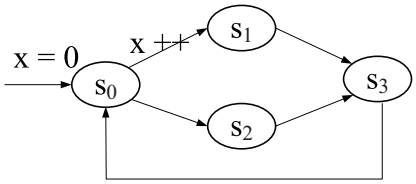

Fig. 1. A simple example for BMC

branches in the search tree. As a result, many unreachable states may be explored by the SAT solver. Consider the example in Fig. 1: given any path of length 8, the only reachable location is s_3. This is obvious by looking at the structure of the model. However, when a SAT solver makes decisions in the search tree, all this information is lost. As a result, the SAT solver may lead the search tree in the eighth step to any location, including s_0 to s_2. It will be very time-consuming for the SAT solver to resolve the unreachable locations.

To avoid such situation, the basic idea is to keep and utilize the structure information of the model in SAT solving. We define a *transition variable* for each transition in the model. During the SAT solving, the transition variable is assigned higher priority than other variables to be chosen as a decision variable. Among the many transition variables, their priorities are assigned following the structure of the transition system. In this way, the structure information is utilized to guide the search process of the SAT solver.

The strategies are realized in VCS[2], a model checker for component-based systems in the BIP language [2]. Experiments with this heuristic on real industrial designs show 5-12 times speedup over standard bounded model checking.

A. Related Work

There are some works on utilizing circuits' structure to speedup the SAT solving. In [3], Kuehlmann et al. uses circuit specific knowledge to guide the search of SAT solving. In [4], Gupta et al. explores implications learned from the circuit structure to help the SAT solving. For a CNF with no structure information, Ostrowski et al. [5] suggests recovering and exploiting structural knowledge to eliminate clauses and variables.

[1] Variable decision: when a SAT solver cannot deduce anymore, it selects a variable and then guesses its value.

[2] http://code.google.com/p/bip-vcs/

978-0-7695-5053-4/13 $26.00 © 2013 IEEE
DOI 10.1109/TASE.2013.11

There are also some works carried out to exploit the characteristics of BMC to refine the SAT decision ordering. In [6], the authors identifies important variables from previous unsatisfiable BMC instances, and makes decisions first on the important variables to solve the current instance. The work most related with ours is [7]. It suggests predetermining a static order, following either a forward or backward Breadth-First Search (BFS) on Variable Dependency Graph (VDG). But the strict backward or forward BFS will make the method to be explicit. So it suggests triggering the BFS with a set S of small number of variables from each cycle. In our method, as our algorithm is limited to transition systems, so transition variables can be an appropriate trade-off between explicit enumerations and reducing unreachable states.

The rest of the paper is organized as follows. Section II introduces the preliminaries. Section III-A defines the transition variables, Section III-B presents our decision heuristics, and Section III-C gives the BMC algorithm enhanced by our heuristics. Section IV reports the experimental results. Finally Section V concludes this paper.

II. Preliminaries

A. Extended Labeled Transition System

A *labeled transition system (LTS)* is a tuple $\langle \mathcal{L}, \mathcal{L}_0, \mathcal{P}, \mathcal{T} \rangle$, where $\mathcal{L}$ is a set of local states, $\mathcal{L}_0 \subseteq \mathcal{L}$ is a set of initial states, $\mathcal{P}$ is a set of labels, and $\mathcal{T} \subseteq \mathcal{L} \times \mathcal{P} \times \mathcal{L}$ is a set of transitions.

An *extended labeled transition system (ELTS)* $\mathcal{E}$ is a tuple $\langle \mathcal{L}, \mathcal{L}_0, \mathcal{P}, \mathcal{T}, \mathbf{v}, \{g_t\}_{t\in\mathcal{T}}, \{\alpha_t\}_{t\in\mathcal{T}} \rangle$ where $\langle \mathcal{L}, \mathcal{L}_0, \mathcal{P}, \mathcal{T} \rangle$ is a LTS; $\mathbf{v}$ is a set of local variables; $\{g_t\}_{t\in\mathcal{T}}$ is a set of predicates on $\mathbf{v}$, and for each transition $t \in \mathcal{T}$, g_t is the guard that enables t; $\{\alpha_t\}_{t\in\mathcal{T}}$ is a set of predicates on the *current* and *next-state* variables $\{\mathbf{v}, \mathbf{v}'\}$, and for each transition $t \in \mathcal{T}$, α_t is the function that updates the values of variables in $\mathbf{v}$.

Given a set of ELTSs $\{\mathcal{E}_1, \mathcal{E}_2, \cdots, \mathcal{E}_n\}$, denote $\mathcal{T}_i$ the set of transitions of $\mathcal{E}_i$. A *communication* γ is a set of transitions $trans(\gamma)$ which have to execute synchronously. $trans(\gamma) \subseteq \bigcup_{i=0}^{n} \mathcal{T}_i$, and for any ELTS $\mathcal{E}_i$, $trans(\gamma) \cap \mathcal{T}_i \leq 1$. In γ, there may be some data exchanges among the ELTSs in $dom(\gamma)$.

An *asynchronous system (AS)* $\mathcal{A}$ is a pair $\langle \mathcal{E}, \Gamma \rangle$, where $\mathcal{E} = \{\mathcal{E}_1, \mathcal{E}_2, \cdots, \mathcal{E}_n\}$ is a set of ELTSs, and $\Gamma = \{\gamma_1, \gamma_2, \cdots, \gamma_m\}$ is a set of communications on $\mathcal{E}$. In the concurrent semantics, each ELTS in $\mathcal{A}$ should execute separately, except that those transitions participating the same communication should execute at the same time. A single ELTS $\mathcal{E}$ can be seen as a special AS $\mathcal{A} = \{\{\mathcal{E}\}, \emptyset\}$.

Given an AS $\mathcal{A} = \langle \mathcal{E}, \Gamma \rangle$, the internal transitions of any $\mathcal{E}_i \in \mathcal{E}$ are those ones that do not join any communication of Γ. The set of global transitions of an AS $\mathcal{A}$ is denoted by $\mathbf{T}$. $\forall \mathbf{t} \in \mathbf{T}$, it can be either an internal transition t of any $\mathcal{E}_i \in \mathcal{E}$, or a communication $\gamma_i \in \Gamma$.

Denote $\mathbf{x}$ and $\mathbf{x}'$ the *current* and *next-state* variables of $\mathcal{A}$ respectively, and let $F_{\mathbf{t}}$ be the symbolic representation of the global transition $\mathbf{t}$, then the transition relation $\mathcal{R}$ of an AS $\mathcal{A}$ can be formulated as

$$\mathcal{R}(\mathbf{x}, \mathbf{x}') = \bigvee_{\mathbf{t}\in\mathbf{T}} F_{\mathbf{t}}(\mathbf{x}, \mathbf{x}') \tag{1}$$

The AS $\mathcal{A}$ can be formulated as a *Finite State Machine (FSM)* $M = (\mathbf{x}, \mathcal{I}(\mathbf{x}), \mathcal{R}(\mathbf{x}, \mathbf{x}'))$ where $\mathbf{x}$ is the set of state variables of $\mathcal{A}$, $\mathcal{I}(\mathbf{x})$ is the initial predicate, and $\mathcal{R}(\mathbf{x}, \mathbf{x}')$ is the transition relation.

B. Bounded Model Checking

Given a FSM $M = (\mathbf{x}, \mathcal{I}(\mathbf{x}), \mathcal{R}(\mathbf{x}, \mathbf{x}'))$ and a safety property $\phi(\mathbf{x})$, the existence of a counterexample of length k can be reduced to a SAT problem, i.e.

$$\Omega(k) = \mathcal{I}(\mathbf{x}_0) \wedge (\bigwedge_{i=0}^{k-1} \mathcal{R}(\mathbf{x}_i, \mathbf{x}_{i+1})) \wedge (\bigvee_{i=0}^{k} \neg\phi(\mathbf{x}_i)). \tag{2}$$

Note that $\Omega(k)$ is a Boolean formula, which is first converted to the *conjunctive normal form (CNF)*, and then solved by a SAT solver. Property ϕ is false iff there exists some k so that $\Omega(k)$ is satisfiable.

C. SAT Solving

Given any propositional formula f, it can be converted to CNF in linear time by introducing intermediate variables [8] [9]. In most of the CNF conversion algorithms, a new name is given to every sub-formulae of f, which is known as the *definitional* clause, and the resulting CNF is *equisatisfiable* to the original formula f. For example, formula $f = ((a \wedge b) \vee c) \wedge d$ can be represented by formula set $S = \{x \Leftrightarrow a \wedge b, y \Leftrightarrow x \vee c, z \Leftrightarrow y \wedge d, z\}$ where each subformula $t \in S$ can be converted to clauses trivially.

Most modern SAT solvers are variants of the DPLL algorithm [10], which is based on the Depth-First Search (DFS) on the variable space. In the beginning, the DPLL algorithm selects a variable and assigns it true or false. This process is called making a *decision*. A clause is a unit clause if it has only one free literal and all the other literals are false. To make a unit clause satisfiable, the free literal must be set to true, and then this literal is called a propagated literal. The process of propagating literals iteratively is called Boolean constraint propagation (BCP). A DPLL algorithm iteratively makes decisions and applies BCP, until either a satisfiable assignment is found or the search tree has been fully explored. For a DPLL algorithm, the number of decisions it makes and the number of literals it propagates reflect the size of the search tree and can be used to evaluate the efficiency of the SAT solver.

III. Our Approach

A. Transition Variable

To convert the transition relation $\mathcal{R}$ to a CNF, for each global transition $\mathbf{t}$, we need to add one intermediate variable, i.e.

$$\mathcal{R} = \bigvee_{\mathbf{t}\in\mathbf{T}} F_{\mathbf{t}} = \bigvee_{\mathbf{t}\in\mathbf{T}} v_{\mathbf{t}} \wedge \bigwedge_{\mathbf{t}\in\mathbf{T}} (v_{\mathbf{t}} \Leftrightarrow F_{\mathbf{t}}). \tag{3}$$

Definition 1: When converting the transition relation $\mathcal{R}$ to a CNF, the intermediate variable $v_{\mathbf{t}}$ added for each transition $\mathbf{t}$ is called a *transition variable*.

Note that these transition variables are needed to convert $\mathcal{R}$ to a CNF. So in our algorithm, we do not add any new variable.

B. Heuristics for SAT Decision Ordering

A SAT solver is essentially a DFS algorithm, which iteratively makes decisions on unassigned variables. When a SAT solver makes a decision, it prefers choosing the variable leading to the maximal propagations. Many heuristics have been proposed to evaluate variables dynamically [11]. Each time a SAT solver makes a decision, it chooses the variable with the greatest evaluation value. For example, DLIS is a heuristic strategy which prefers choosing the literal with the most occurrences in the unassigned clauses.

The basic idea here is that the decision variable should deduce as much as possible from other variables, so that assigning values to this variable can lead to more propagation than other variables. A transition variable in $\mathcal{R}(\mathbf{x}_k, \mathbf{x}_{k+1})$ bridges two big sets of system variables, that is $\mathbf{x}_k$ and $\mathbf{x}_{k+1}$. Intuitively, decision on transition variables may lead to a large number of unit clause propagations.

Rule 1: Transition variables have higher priority than other variables in the SAT decision ordering.

Formally, let $\preceq$ be the partial order on variables, $x_1 \preceq x_2$ means x_1 is preferred over x_2 to be selected as the decision variable. Rule 1 states that transition variables lead all other variables in the partial order.

Rule 2: The transition variables in the early system evolution have higher priorities than those in the later system evolution in the SAT decision ordering.

More formally, the transition variables in $\mathcal{R}(\mathbf{x}_k, \mathbf{x}_{k+1})$ has higher priorities than those in $\mathcal{R}(\mathbf{x}_{k+1}, \mathbf{x}_{k+2})$. According to the semantics of asynchronous systems, in each system evolution only one transition can be taken, and the transition taken in the next evolution must be a successor of the current one. Rule 2 attempts to limit the search following the structure of the transition system. In other words, Rule 2 makes the SAT search explicitly on locations, and it can prune a large number of unreachable branches in the search tree.

Consider the example shown in Fig. 1 again. Since a transition variable can be set true only if its preceding transition has already been taken, the search path of length 8 will never come to s_0, s_1 and s_2.

To realize the heuristics, we use an array $priority$ to store the priority of each variable. Given k the search length of BMC, Δ_i the set of transition variables in $\mathcal{R}(\mathbf{x}_k, \mathbf{x}_{k+1})$, we assign each transition variable with priority,

$$priority[v] = k + 1 - i, \text{ for } v \in \Delta_i, 0 \leq i \leq k \qquad (4)$$

and all other variables with priority 0. In our algorithm, the SAT solver always selects the unassigned variable with the highest priority to make the decision. If several variables have the same priority, it relies on the default heuristic, like DLIS and VSIDS, to make decision.

After a variable being selected, the next step is to determine its value. There are many heuristics on deciding the variable's value [11]. In our algorithm, if a transition variable (with priority greater than 0) is chosen, its value is constantly set to 1, since we want to take the corresponding transition; otherwise, the value is determined by the default heuristics of SAT solver.

C. Enhanced Bounded Model Checking

The standard BMC algorithm is enhanced with our variable decision heuristic, as shown in Algorithm 1. It accepts 4 parameters, i.e. the initial predicate $\mathcal{I}$, the transition relation predicates $\mathcal{R}$, the safety property ϕ and the maximal search length D. The search length for BMC increases from 0 to D. With each determined search length, the $cnfConvert$ function reduces the BMC problem to a SAT formula according to Eq. (2), and then converts it to CNF. The priorities of variables are also computed in the $cnfConvert$ function, according to Eq. (4). The satisfiability of the formula is then solved by a SAT procedure. Note that the array $priority$ is also provided to assist the SAT solving. The algorithm terminates with a counterexample ce if the SAT procedure returns SAT, otherwise it iterates the process with the search length increased by 1.

Algorithm 1 Enhanced BMC Algorithm bmc^*

Require: $\mathcal{I}$, $\mathcal{R}$, ϕ, D
1: **for** $k = 0$ to D **do**
2: $(\Omega, priority)$ = $cnfConvert(\mathcal{I}, \mathcal{R}, \phi, k)$
3: (ret, ce) = $satSolve(\Omega, priority)$
4: **if** ret == SAT **then**
5: **return** ce
6: **end if**
7: **end for**

IV. Experimental Results

Our enhanced bounded model checking algorithm is implemented in VCS, a verifier for component-based systems, which accepts models described in BIP language [2]. A small tool is implemented to extract SMV models from BIP models. The method is that, a FSM $M = (\mathbf{x}, \mathcal{I}(\mathbf{x}), \mathcal{R}(\mathbf{x}, \mathbf{x}'))$ can be easily written in SMV format using the {*VAR, INIT, TRANS*} operators. Our approach is compared to the existing bounded model checking algorithm implemented in NuSMV-2.5.3[3] (the base step of een-sorensson algorithm). The involved SAT solver in both VCS and NuSMV is minisat2-070221[4]. All experiments are conducted on a computer with a Intel 2.67GHz CPU and 4GB memory.

In this paper, we consider 3 examples taken from real industry. The first example is a data processing unit (DPU) used in a space vehicle [12], which continuously collects data from sensors, processes the data and then sends results to the master computer. The second example is a gate control system (GCS), which is used in LingShan Buddhist Palace in Jiangsu, China [13]. The third example is a real time protocol (RTP) used in the train communication network (IEC 61375). Additionally, two examples from BIP's website[4] are also considered: the ATM example and the dining philosophers problem (DPP).

All experimental results are listed in TABLE I, where "Example" column shows the name of the models, "Prop" column gives the property name it verifies, "Step" column lists the search length for the model checker to find a counterexample, "Time" column gives the run time (in seconds) for *NuSMV*

[3] http://nusmv.fbk.eu/
[4] http://minisat.se/MiniSat.html

TABLE I. EXPERIMENTAL RESULTS

Example	Prop	Step	Time(s)			Decisions($\times 10^3$)		Propagations($\times 10^6$)		Prop/Dec	
			NuSMV	bmc	bmc*	bmc	bmc*	bmc	bmc*	bmc	bmc*
DPU	p1	24	5.01	1.08	0.09	128.89	0.28	2.98	0.26	23	947
DPU	p2	26	6.59	1.46	0.15	164.52	0.33	3.81	0.34	23	1005
DPU	p3	32	14.4	4.00	0.39	354.35	0.66	10.74	0.93	30	1407
GCS	p1	46	36.72	24.80	4.61	849.74	2.81	169.45	33.79	199	12044
GCS	p2	54	224.31	119.50	11.59	1748.37	8.39	845.02	84.32	483	10055
RTP	p1	30	-	134.71	17.74	1167.93	56.31	157.96	73.78	135	1310
RTP	p2	30	-	195.87	16.93	1348.60	57.12	184.58	72.27	137	1265
RTP	p3	33	-	184.64	30.35	1549.05	95.97	204.72	129.36	132	1348
RTP	p4	30	-	105.82	18.11	1043.50	58.29	136.52	74.45	131	1282
ATM	p1	15	-	627.81	98.26	3140.70	388.68	232.50	136.92	74	352
DPP	p1	10	33.01	6.62	44.69	150.13	377.13	9.26	99.50	62	264

(with the een-sorensson algorithm), *bmc* (the standard BMC algorithm in VCS) and *bmc** (our enhanced BMC algorithm in VCS), "Decisions" and "Propagations" columns list the total number of decisions ($\times 10^3$) and the total number of propagations ($\times 10^6$) made by the SAT solver of *bmc* and *bmc** respectively, and "Prop/Dec" gives the ratio of "Propagations" to "Decisions". If a checker runs out of memory or fails to give the result in 900 seconds, "-" is marked in the table.

Comparing *bmc* to *NuSMV*, we observed that *bmc* runs a bit faster than *NuSMV* for all examples. We believe this performance improvement is due to the new CNF conversion methods implemented in VCS. Comparing *bmc* to *bmc**, we observed that *bmc** runs much faster for the first 3 examples, where each model contains plenty of local variables. The least speedup is 5 times (with model GCS and property p1), the greatest speedup is 12 times (with model DPU and property p1), and the average speedup is 8 times. For the remaining two examples from BIP website, *bmc** run faster for ATM but slower for DPP. This is reasonable since the models for these two examples contain no local variable, which may bring our technique into an explicit metod.

From Table I, we also observed that for the first three examples, the numbers of decisions and propagations made by *bmc** are much less than those made by *bmc*. In total, the number of decisions has been reduced by 97%, and the number of propagations has been reduced by 73%. This finding supports our fist observation on the performance improvement of *bmc** over *bmc*.

By looking at the last two columns, we observed that for the examples with plenty of local variables, the number of propagations per decision in *bmc** is $10+$ times larger than that in *bmc*. Note *bmc** uses our heuristic, while *bmc* uses VSIDS. As we discussed in section III-B, a better heuristic for SAT decision result in more unit clause propagations. This finding gives an evidence for the superiority of our heuristic.

V. CONCLUSION

This paper considers bounded model checking for extended labeled transition systems. It proposed an enhanced BMC algorithm with optimized SAT decision ordering. This ordering is based on the so called transition variables, which can be used to prune lots of unreachable branches in the search tree. Experiments on real industrial designs show significant performance improvement of our algorithm over the standard bounded model checking.

ACKNOWLEDGMENT

This work was supported by the Chinese National 973 Plan under grant No. 2010CB328003, the NSF of China under grants No. 61272001, 60903030, 91218302, the Chinese National Key Technology R&D Program under grant No. SQ2012BAJY4052, and the Tsinghua University Initiative Scientific Research Program.

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 annual ACM/IEEE Design Automation Conference*. ACM, 1999, pp. 317–320.

[2] A. Basu, M. Bozga, and J. Sifakis, "Modeling heterogeneous real-time components in BIP," in *Software Engineering and Formal Methods, 2006. SEFM 2006. Fourth IEEE International Conference on*. IEEE, 2006, pp. 3–12.

[3] A. Kuehlmann, M. Ganai, and V. Paruthi, "Circuit-based boolean reasoning," in *Design Automation Conference, 2001. Proceedings*. IEEE, 2001, pp. 232–237.

[4] A. Gupta, M. Ganai, C. Wang, Z. Yang, and P. Ashar, "Learning from BDDs in SAT-based bounded model checking," in *Design Automation Conference, 2003. Proceedings*. IEEE, 2003, pp. 824–829.

[5] R. Ostrowski, É. Grégoire, B. Mazure, and L. Sais, "Recovering and exploiting structural knowledge from CNF formulas," in *Principles and Practice of Constraint Programming-CP 2002*. Springer, 2006, pp. 199–206.

[6] C. Wang, H. Jin, G. Hachtel, and F. Somenzi, "Refining the SAT decision ordering for bounded model checking," in *Proceedings of the 41st annual Design Automation Conference*. ACM, 2004, pp. 535–538.

[7] O. Shtrichman, "Tuning SAT checkers for bounded model checking," in *Computer Aided Verification*. Springer, 2000, pp. 480–494.

[8] G. S. Tseitin, "On the complexity of derivation in propositional calculus," *Studies in Constructive Mathematics and Mathematical Logic*, p. 115C125, 1962.

[9] D. Plaisted and S. Greenbaum, "A structure-preserving clause form translation," *Journal of Symbolic Computation*, vol. 2, no. 3, pp. 293–304, 1986.

[10] M. Davis and H. Putnam, "A computing procedure for quantification theory," *J. ACM*, vol. 7, pp. 201–215, 1960.

[11] J. Marques-Silva, "The impact of branching heuristics in propositional satisfiability algorithms," *Progress in Artificial Intelligence*, pp. 850–850, 1999.

[12] H. Wan, C. Huang, Y. Wang, F. He, M. Gu, R. Chen, and B. Marius, "Modeling and validation of a data process unit control for space applications," in *Embedded Real Time Software and Systems*, 2012.

[13] R. Wang, M. Zhou, L. Yin, L. Zhang, M. Gu, J. Sun, and M. Bozga, "Modeling and validation of PLC-controlled system: A case study," Tsinghua University, Tech. Rep., 2011.