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

Conference Paper · July 2013

DOI: 10.1109/TASE.2013.11

3 authors, including:

Fei He
Tsinghua University
74 PUBLICATIONS **438** CITATIONS

SEE PROFILE

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

***Abstract*—This paper considers bounded model checking for extended labeled transition systems. Bounded model checking relies on SAT solver to prove (or disprove) the existence of a counterexample with a bounded length. During the translation of BMC problem to 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 that the search process of SAT solving should follow the structure of the transition system. Experiments with this heuristic on real industrial designs show 5-12 times speedup over the standard bounded model checking.**

I. Introduction

Bounded model checking (BMC) [1][2] has been widely accepted as a complementary of 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, 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 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.

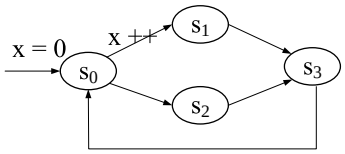

Fig. 1. A simple example for BMC

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 decision variable. Among the many transition variables, we follow the structure of the transition system to assign their priorities. In such a way, the structure information is utilized to guide the search process of SAT solver.

The strategies are realized in VCS[2], a model checker for component-based systems in BIP language [3]. Experiments with this heuristic on real industrial designs show 5-12 times speedup over the standard bounded model checking.

A. Related work

In last decades, bounded model checking has mainly been applied in hardware verification. There are some works on utilizing circuits' structure to speedup the SAT solving. [4] focuses on circuit-based SAT solvers, and uses circuit specific knowledge to guide the search. However, CNF-based solvers are better than these solvers in handling conflict analysis and learned clauses. [5] proposes to combine the strengths of circuit-based and CNF-based SAT solvers; it proposed a hybrid SAT solver where the original logic formula is processed in circuit form, and the learned clauses are processed separately in CNF. [6] explores implications learned from the circuit structure to help the SAT solving. For a CNF with no structure information, [7] 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/

There are also some works carried out to exploit the characteristics of BMC to refine the SAT decision ordering. In [8], observed that the sequence of problems produced in BMC are highly correlated, it 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 [9]. 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, such as the variables appeared in the property P_i for each cycle i. 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 of our approach, including transition systems, bounded model checking and SAT solving. Section III-A defines the transition variables, and Section III-B presents our decision heuristics. Section IV reports the experimental results. Finally Section V concludes this paper.

II. Preliminaries

A. Labeled transition system

Definition 1: A labeled transition system (LTS) is a tuple $\langle L, L_0, P, T\rangle$, where

- L is a set of locations,
- $L_0 \subseteq L$ is a set of initial locations,
- P is a set of labels, and
- $T \subseteq L \times P \times L$ is the transition relation.

Definition 2: An extended labeled transition system (ELTS) S is a tuple $\langle L, L_0, P, T, V, \{g_t\}_{t\in T}, \{\alpha_t\}_{t\in T}\rangle$ where:

- $\langle L, L_0, P, T\rangle$ is a labeled transition system,
- V is a set of variables,
- $\{g_t(V)\}_{t\in T}$ is a set of predicates on V, and for each transition $t \in T$, g_t is a guard,
- $\{\alpha_t(V, V')\}_{t\in T}$ is a set of predicates on V and V', and for each transition $t \in T$, α_t is an update function.

The unprimed and primed variables are used for the current and the next step respectively. For instance V and V', X and X', v and v'.

Note that each transition is adhered with a guard predicate and an action predicate. Given a transition $t = (l, p, l')$, we write $l \xrightarrow{p,g_t,\alpha_t} l'$. Given a location l and a transition t, $Post(l, t) = \{l' \in L | l \xrightarrow{p,g_t,\alpha_t} l'\}$ is the set of successors of l with the transition t, $Post(l) = \bigcup_{t\in T} Post(l, t)$ is the set of all successors of l. And for any two different transitions t_i, t_j, we have $l_{t_i} \neq l_{t_j}$ or $l'_{t_i} \neq l'_{t_j}$ or $\alpha_{t_i} \neq \alpha_{t_j}$, otherwise, t_i and t_j can be merged into one transition.

To denote the current location, a special variable loc is imported which ranges over L, and we say it a location variable. Given a transition $t \in T$ and its associated g_t, α_t, its symbolical representation is

$$F_t = (loc \Leftrightarrow l_t) \wedge g_t \wedge \alpha_t \wedge (loc' \Leftrightarrow l'_t).$$

Theorem 1: An extended labeled transition system $S = \langle L, L_0, P, T, V, \{g_t\}_{t\in T}, \{a_t\}_{t\in T}\rangle$ can be symbolically represented as a tuple $\langle X, F_I, F_T\rangle$, where

- $X = V \cup \{loc\}$,
- F_I is the characteristic function of L_0 in L, and
- $F_T = \bigvee_{t\in T} F_t$.

Definition 3: Given a set of ELTSs $S_1, S_2, \cdots, S_n$, where $S_i = \langle L_i, L_{0i}, P_i, T_i, V_i, \{g_t\}_{t\in T_i}, \{\alpha_t\}_{t\in T_i}\rangle$, an interaction γ is a tuple $\langle ST, g_\gamma, \alpha_\gamma\rangle$, where

- $ST \subseteq \bigcup_{i=1}^{n} P_i$ is a set of labels, such that for all $i \in \{1, \cdots, n\}, |ST \cap P_i| \leq 1$. If $ST \cap P_i = 1$, S_i participates γ, namely $S_i \in dom(\gamma)$.
- g_γ is a predicate on $\bigcup_{S_i\in dom(\gamma)} V_i$, which defines the guard of the interaction.
- α_γ is a predicate on $\bigcup_{S_i\in dom(\gamma)} V_i$ and $\bigcup_{S_i\in dom(\gamma)} V'_i$, which defines the action of the interaction.

Definition 4: A system $\mathbb{S}$ is a pair $\{\mathbb{H}, \mathbb{I}\}$, where $\mathbb{H} = \{S_1, S_2, \cdots, S_n\}$ is a set of ELTSs, and $\mathbb{I} = \{\gamma_1, \gamma_2, \cdots, \gamma_m\}$ is a set of interactions on $\mathbb{H}$.

Given a system $\mathbb{S} = \{\mathbb{H}, \mathbb{I}\}$, a transition of $\mathbb{S}$ is either an inner transition of $S \in \mathbb{H}$, which does not join any interaction, or a concurrent transition which is synchronized by an interaction γ. Denote T^I the set of inner transitions in $\mathbb{S}$.

Given an inner transition t of S, its symbolic representation in the system view of $\mathbb{S}$ is

$$\mathbb{F}_t = F_t \wedge \bigwedge_{S_k \neq S} (X'_k = X_k).$$

Given an interaction $\gamma = \{ST, g_\gamma, \alpha_\gamma\}$, the corresponding concurrent interaction in the system view of $\mathbb{S}$ can be represented as

$$\mathbb{F}_\gamma = (\bigwedge_{t\in ST} F_t) \wedge g_\gamma \wedge \alpha_\gamma \wedge \bigwedge_{S_k \notin dom(\gamma)} (X'_k = X_k).$$

Theorem 2: Given a system $\mathbb{S} = \{\mathbb{H}, \mathbb{I}\}$ where each $S_i \in \mathbb{H}$ is symbolically represented as $\langle X_i, I_i, F_{Ti}\rangle$, then $\mathbb{S}$ can be symbolically represented as $\langle \mathbb{X}, \mathbb{F}_I, \mathbb{F}_T\rangle$ where

- $\mathbb{X} = \bigcup_{S_k\in\mathbb{H}} X_k$,
- $\mathbb{F}_I = \bigwedge_{S_k\in\mathbb{H}} F_{I_k}$, and
- $\mathbb{F}_T = \bigvee_{t\in T^I} \mathbb{F}_t \vee \bigvee_{\gamma\in\mathbb{I}} \mathbb{F}_\gamma$.

B. Bounded model checking

Given a transition system $S = \{X, F_I, F_T\}$ [3] and a safety property AGp, the existence of a counter-example of length k can be reduced to a BMC problem, i.e.

$$\Omega(k) = F_I(X_0) \wedge (\bigwedge_{i=0}^{k-1} F_T(X_i, X_{i+1})) \wedge (\bigvee_{i=0}^{k} \neg p(X_i)), \quad (1)$$

[3] In the following, we do not distinguish a system and a module in the system, by representing them uniformly as S.

where $X_i (0 \leq i \leq k)$ is a copy of X characterizing the i-th state in a path, $F_I(X_0)$ means the first state must be an initial state, and $F_T(X_i, X_{i+1})$ represents the transition from the i-th state to the $(i+1)$-th state.

Note that $\Omega(k)$ is a SAT formula, which is firstly converted to the conjunctive normal form (CNF), and then solved by a SAT solver. Property p is false iff there exists k such that $\Omega(k)$ is satisfiable.

C. SAT solving

Given a Boolean formula f over a set of Boolean variables, the CNF of f is simply a conjunction of clauses, where each clause is a disjunction of literals, and each literal is either a Boolean variable or the negation of a Boolean variable. Given any propositional formula f, it can be converted to CNF in linear time by introducing intermediate variables [10][11][12][13]. 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 use DPLL framework [14][15], which is based on Depth-First Search (DFS) on the variable space. In the beginning, a 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 constrain propagation (BCP). A DPLL algorithm iteratively makes decisions and applies BCP, until either a satisfiable assignment is found or a conflict is encountered. In the latter case, the algorithm learns a conflict clause and then backtracks to a former decision point.

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

For simplicity, we use F_t to represent both the original formula and the CNF of transition t. To convert the F_T to CNF, we need to add one intermediate variable for each F_t, i.e.

$$F_T = \bigvee_{t \in T} F_t = \bigvee_{t \in T} v_t \wedge \bigwedge_{t \in T} (v_t \Leftrightarrow F_t).$$

Definition 5: When converting F_T to CNF, the intermediate variable v_t added for each transition F_t is called a transition variable.

Consider an ELTS example shown in Fig. 2, its transition relation is $F_T = F_{t1} \vee F_{t2}$, where

$$F_{t1} = ((loc \Leftrightarrow L1) \wedge (x' \Leftrightarrow 0) \wedge (loc' \Leftrightarrow L2)),$$

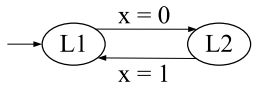

Fig. 2. Transition variables of ELTS

$$F_{t2} = ((loc \Leftrightarrow L2) \wedge (x' \Leftrightarrow 1) \wedge (loc' \Leftrightarrow L1)).$$

When converting F_T to CNF, two transition variables v_1, v_2 will be added to represent F_{t1} and F_{t2} respectively, then F_T can be expressed as

$$(v_1 \vee v_2) \wedge (v_1 \Leftrightarrow F_{t1}) \wedge (v_2 \Leftrightarrow F_{t2}).$$

Proposition 1: A transition variable v_t can be assigned true only if the current location is at l_t, and the guard condition g_t is true.

Proof: If one of the two conditions is not satisfiable, then the formula $v_t \Leftrightarrow ((loc \Leftrightarrow l_t) \wedge g_t \wedge \alpha_t \wedge (loc' \Leftrightarrow l'_t))$ will be false, which will make all the transition relations false. ■

Proposition 2: Taking a transition t in S is equivalent to assigning the corresponding transition variable v_t true in the SAT search process.

Proof: Here selecting transition t is in the intuitive perspective. It means current loc is at l_t, g_t is true, and then loc changed to l'_t, and all the local variables updated as described in g_t. According to the definition of F_t and transition variable, we have $v_t \Leftrightarrow ((loc \Leftrightarrow l_t) \wedge g_t \wedge \alpha_t \wedge (loc' \Leftrightarrow l'_t))$, so if v_t is assigned value true, all the conditions will be satisfied. ■

From one location l, there may be more than one transitions, we say that these transitions (also the corresponding transition variables) originate from l.

Proposition 3: Given a set of transition variables that originated from location l, only one transition variable can be set true, and all other transition variables must be set false.

Proof: 1) Without loss of generality, suppose two different transition variables v_i, v_j are true at the same time, then they will deduce that $(loc \Leftrightarrow l_{t_i}) \wedge \alpha_{t_i} \wedge (loc' \Leftrightarrow l'_{t_i}) \wedge (loc \Leftrightarrow l_{t_j}) \wedge \alpha_{t_j} \wedge (loc' \Leftrightarrow l'_{t_j})$. This formula is true iff $(l_{t_i} = l_{t_j}) \wedge (l'_{t_i} = l'_{t_j}) \wedge (\alpha_{t_i} = \alpha_{t_j})$, but according to the definition of ELTS, for any two different transitions t_i, t_j, $l_{t_i} \neq l_{t_j}$ or $l'_{t_i} \neq l'_{t_j}$ or $\alpha_{t_i} \neq \alpha_{t_j}$. So there is only one transition variable that can be true in one step.

2) Without loss of generality, suppose transition variable v_i is set to be true, then it will imply that $(loc \Leftrightarrow l_{t_i}) \wedge g_{t_i} \wedge \alpha_{t_i} \wedge (loc' \Leftrightarrow l'_{t_i})$. According to the definition of ELTS, for any transition $t_j (j \neq i)$, $l_{t_i} \neq l_{t_j}$ or $l'_{t_i} \neq l'_{t_j}$ or $\alpha_{t_i} \neq \alpha_{t_j}$, then $(loc \Leftrightarrow l_{t_j}) \wedge g_{t_j} \wedge \alpha_{t_j} \wedge (loc' \Leftrightarrow l'_{t_j})$ must be false, which will deduce v_j to be false. So assigning one transition variable true will imply all other transition variables to be false. ■

Note that in formula $\Omega(k)$, $F_T(X_k, X_{k+1})$ represents the transition relation for the system evolving from the i-th state to the $(i+1)$-th state. During a system evolution, when one transition is taken, accordingly in the SAT search process one transition variable should be set true.

Proposition 4: Let v_i^k and v_j^{k+1} be the transition variables assigned true in $F_T(X_k, X_{k+1})$ and $F_T(X_{k+1}, X_{k+2})$ respec-

tively, and v_i^k corresponds to t_i, v_j^{k+1} corresponds to t_j, then t_i must be immediately followed by t_j in the transition system.

Proof: Let loc^k be the location variable in kth cycle, according to the definition of transition variable, assigning v_i^k true can deduce that $(loc^k \Leftrightarrow l_{t_i}) \wedge (loc^{k+1} \Leftrightarrow l'_{t_i})$, and assigning v_j^{k+1} true can deduce that $(loc^{k+1} \Leftrightarrow l_{t_j}) \wedge (loc^{k+2} \Leftrightarrow l'_{t_j})$. As loc^{k+1} must be the same, so l'_{t_i} is equal to l_{t_j}, and transition t_j is a successor of t_i. ■

Proposition 4 states that the transition variables selected during two consecutive system evolutions should meet the structure of the transition system.

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 [16][17][18][19]. 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 more as possible of other variables, such that assigning values to this variable can lead to more propagation than other variables. A transition variable in $F_T(X_k, X_{k+1})$ bridges two big sets of system variables, that is X_k and 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 $F_T(X_k, X_{k+1})$ has higher priorities than those in $F_T(X_{k+1}, X_{k+2})$. According to Proposition 3 and 4, in each system evolution only one transition can be taken, and the transition taken in the next evolution must be 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, W_i the set of transition variables in $F_T(X_k, X_{k+1})$, we assign each transition variable with priority,

$$priority[v] = k + 1 - i, \text{ for } v \in W_i, 0 \leq i \leq k-1 \quad (2)$$

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 [20], 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 [16]. 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 Alg. 1. It accepts 4 parameters, i.e. the initial predicate F_I, the set of transition predicates $\{F_t | t \in T\}$, the safety property P and the maximal search length D.

The search length for BMC increases from 0 to D. With each determined search length, the gen_cnf function reduces the BMC problem to a SAT formula according to Eq. (1), and then converts it to CNF. The priorities of variables are also computed in the gen_cnf function, according to Eq. (2). 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 if the SAT procedure returns *SAT*, otherwise it iterates the process with the search length increased by 1.

Algorithm 1 Enhanced BMC Algorithm

```
procedure bmc*(F_I, {F_t|t ∈ T}, P, D)
    for k = 0 to D do
        (cnf, priority) = gen_cnf(F_I, {F_t|t ∈ T}, P, k);
        (ret, ce) = sat_solve(cnf, priority);
        if ret == SAT then
            return ce;
        end if
    end for
end procedure
```

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 [3]. A small tool is implemented to extract SMV models from BIP models. Our approach is compared to the existing bounded model checking algorithm implemented in NuSMV-2.5.3[4] (the base step of een-sorensson algorithm). The involved SAT solver in both VCS and NuSMV is minisat2-070221[5]. All experiments are conducted on a computer with a Intel 2.67GHz CPU and 4GB memory.

[4]http://nusmv.fbk.eu/

[5]http://minisat.se/MiniSat.html

TABLE I
MODEL INFORMATION OF EXAMPLES

Example	# ELTSs	# vars	# props
DPU	4	252	3
GCS	2	113	2
RTP	3	190	4
ATM	16	0	1
DPP	20	0	1

To the best of our knowledge, there is no generally accepted benchmarks for extended labeled transition systems. Note there are some examples on BIP's website[6], but most of which concern only the deadlock-freedom property, and cannot be used in our experiments. 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 [21], 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 [22]. The third example is a real time protocol (RTP) used in the train communication network (IEC 61375). Additionally, two examples from BIP's website[6] are also considered: the ATM example and the dining philosophers problem (DPP).

Information of models for these examples are given in Table I, where "# ELTSs" and "# vars" columns list the number of ELTSs and the number of local variables in the model respectively, the "# props" column shows the number of properties to be verified on the model. Since we are comparing the performance of bounded model checking, all properties are false on corresponding models. Especially, note there is no local variable in the models of ATM and DPP.

All experimental results are listed in Table II, 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 (the maximal search length is set to 100), "Time" column gives the run time (in seconds) for *NuSMV* (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 useless.

From Table II, 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.

[6]http://www-verimag.imag.fr/DFinder.html?lang=en

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.

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. Biere, A. Cimatti, E. Clarke, O. Strichman, and Y. Zhu, "Bounded model checking," *Advances in computers*, vol. 58, pp. 117–148, 2003.

[3] 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.

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

[5] M. Ganai, P. Ashar, A. Gupta, L. Zhang, and S. Malik, "Combining strengths of circuit-based and cnf-based algorithms for a high-performance sat solver," in *Proceedings of the 39th annual Design Automation Conference*. ACM, 2002, pp. 747–750.

[6] 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.

[7] 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.

[8] 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.

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

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

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

TABLE II
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

[12] P. Jackson and D. Sheridan, "Clause form conversions for boolean circuits," in *Theory and Applications of Satisfiability Testing*. Springer, 2005, pp. 899–899.

[13] P. Manolios and D. Vroon, "Efficient circuit to cnf conversion," *Theory and Applications of Satisfiability Testing–SAT 2007*, pp. 4–9, 2007.

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

[15] M. Davis, G. Logemann, and D. Loveland, "A machine program for theorem proving," *Comms. ACM*, vol. 5, pp. 394–397, July 1962.

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

[17] M. Buro and H. Büning, *Report on a SAT competition*. Fachbereich Math.-Informatik, Univ. Gesamthochschule, 1992.

[18] J. Freeman, "Improvements to propositional satisfiability search algorithms," Ph.D. dissertation, Citeseer, 1995.

[19] R. Jeroslow and J. Wang, "Solving propositional satisfiability problems," *Annals of mathematics and Artificial Intelligence*, vol. 1, no. 1, pp. 167–187, 1990.

[20] L. Zhang, C. Madigan, M. Moskewicz, and S. Malik, "Efficient conflict driven learning in a boolean satisfiability solver," in *Proc. International Conference on Computer-Aided Design (ICCAD)*, November 2001.

[21] 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.

[22] 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.