Improving SAT-based Bounded Model Checking by Means of BDD-based Approximate Traversals

Gianpiero Cabodi Sergio Nocco Stefano Quer

Politecnico di Torino
Dip. di Automatica e Informatica
Turin, ITALY

Abstract

Binary Decision Diagrams (BDDs) have been widely used for hardware verification since the beginning of the '90s, whereas Boolean Satisfiability (SAT) has been gaining ground more recently, with the introduction of Bounded Model Cheking (BMC).

In this paper we dovetail BDD and SAT based methods to improve the efficiency of BMC. More specifically, we first exploit inexpensive symbolic approximate reachability analysis to gather information on the state space. We then use the above information to restrict and focus the overall search space of SAT based BMC.

In the experimental results section we show how the information coming from a BDD tool can improve the efficiency of a SAT engine by drastically reducing the number of "variable assignments" and "variable conflicts". This results in a significant overall performance gain associated with a general, robust, and easy-to-apply methodology.

1 Introduction

Given a propositional formula, the Boolean Satisfiability Problem (commonly abbreviated as SAT) consists of determining a variable assignment such that the formula evaluates to true, or establishing that no such assignment exists. Although SAT is an NP-complete problem, or at least no polynomial algorithm to solve it is known, SAT solvers have received considerable research attention and large practical instances have been worked out thanks to efficient implementation procedures [1, 2]. Their application ranges from EDA to ATPG, from logic synthesis to verification [3]. In the verification domain, SAT techniques are mainly used for Bounded Model Cheking (BMC), looking for bugs (and counter-examples) of limited length k.

BDDs have often been used in the same fields since they started gaining interest at the end of the '80s. Nevertheless, they have never been able to deal with the largest models and problem instances, because of the so called "BDD blow-up" (or memory explosion) problem.

Several recent papers have compared BDD and SAT methodologies on sequential verification within the unbounded and bounded model checking frameworks. Even though no definite conclusion can be drawn, researchers and engineers agree that SAT tools are complementary to BDD-based ones and that the quest for efficient and comprehensive combinational and sequential verification methods is still not completed.

In this work we explore a new way to make BDD-based and SAT-based tools cooperate. Our target is to improve the efficiency of SAT-based BMC with the help of "cheap" and affordable BDD-based operations. To this respect, "approximate" traversals may deal with larger circuits than "exact" ones, at the expense of exactness. Moreover, as the degree of approximation can be trimmed, it is always possible to trade-off memory and time with the accuracy of the result. Unfortunately nothing comes for free and the limit of approximate techniques in verification is that they are not complete, i.e. over-approximate reachability can prove correctness, but it cannot disprove it.

Our driving idea is to complement the initial over-approximate BDD information with a final SAT-solver search, using BDDs to prune and focus the search. In a first phase, we compute (in the forward and/or backward directions) an over-approximate estimate of the traces connecting the initial state set to the target one. Then the estimate is combined, as an additional constraint, with the Bounded Model Check problem, to be solved by a SAT tool.

Our target is to obtain an efficient pruning of the SAT solver search space, which somehow mimics the contribution of "conflict clauses", generated by means of "conflict analysis" in state-of-the-art SAT tools. Each new conflict clause specifies a sub-set of the state space in which there exist no solution. Similarly, our over-approximate knowledge of reachable states restricts the SAT solver state space. We presently implement this extra information as an initial pre-processing or "learning" phase. On the one hand, we might loose some optimizations achievable through a tighter and more dynamic inter-leaving with the SAT solving tool. On the other hand our method is quite simple and it is compatible with any SAT solver, since we do not require any interaction with inner steps of SAT algorithms. As far as we know, this is the first time a symbolic BDD-based over-approximate information is used to prune a SAT-solver search space.

A further minor contribution of our work is to introduce a set of strategies to store a BDD (in a monolithic or conjoined form) as a CNF formula. These methods will be compared in terms of their compactness to generate the resulting CNF problem (in terms of variables, literals and clauses), and their influence over the SAT-engine (in terms of pruning efficiency).

The remainder of this paper is organized as follows. In Section 2 we introduce some preliminary concepts on notation, SAT problems and reachability analysis. Section 3 is dedicated to the related works. Section 4 introduces our approach and Section 5 describes our technique to store BDDs as CNF problems. Section 6 presents the experimental results. Finally, Section 7 concludes the paper with a brief summary and some hints on possible future work.

2 Background

2.1 Model and Property Definition

The sequential systems we address are usually modeled as Finite State Machines (FSMs). Each FSM is described by a Transition Relation TR, which indicates its present-next state behavior, and an initial state set S.

An invariant property[1] P is checked by attempting to prove (or disprove) the reachability of its complement T (target state set, $\mathsf{T} = \overline{\mathsf{P}}$) from S.

2.2 SAT-Based Model Checking

For an overview on SAT solvers and a complete list of references the reader can refer to the tutorial [3].

SAT based BMC considers only paths of bounded length k and builds a propositional formula f that is satisfiable *iff* there is a counter-example (a path from S to T) of the same length. For the above reason the technique works well in falsification and partial verification, whereas Full verification is usually achieved by BMC with longer and longer bounds, possibly augmented with inductive proofs, when proving correctness rather than seeking for bugs.

SAT solvers generally operate on problems for which f is specified in *Conjunctive Normal Form* (CNF). This form is a two-level decomposition: The logical AND of one or more *clauses*, each of which consists of the logical OR of one or more *literals*. A *literal* is merely an instance of a variable or its complement.

In order to decide if f is satisfied, most solvers adopt variants of the basic Davis-Putnam recursive algorithm. At each step of recursion, the algorithm basically proceeds through the following three steps:

- *Variable Decision*: Assign a value to an unassigned variable so exploring new regions of the search space.
- *Boolean Constraint Propagation*: Carry out all possible implications due to the previous assignment.
- *Conflict Analysis*: Check for "conflicting clauses", i.e., clauses whose literals are all assigned to a value zero, and in case one of this is discovered to undo the current assignment (so that another assignment can be tried). In this phase, "conflict clauses", i.e., clauses which identify previous conflicts, are also added to the clause database for early detection (and pruning) of bad decisions and/or variable assignments.

2.3 BDD-Based Model Checking

A standard BDD-based forward reachability analysis procedure is a breadth-first visit of the state space that starts from $\mathsf{R} = \mathsf{S}$ and proceeds through a least fix-point (*lfp*) iteration:

$$\mathsf{FR} = \mathit{lfp}\ \mathsf{R}.(\mathsf{S} \lor (\textsc{Img}(\mathsf{TR}, \mathsf{R}))$$

which returns FR, i.e., the set of forward reachable states. The method is based on the iterated application of the $\textsc{Img}$ function, to compute symbolic images of the R state set. We indicate with R_i the state sets generated at each traversal iteration (the so called frontier sets).

[1] Or AG CTL property.

As T may be reached before the fix-point it is possible to avoid a full computation of FR with on the fly tests for intersection with T. A counter-example is eventually computed starting from the array FR of frontier sets R_i.

CTL model checking procedures are often implemented as backward traversal procedures, computing BR sets in the backward direction. This is easily expressed by swapping the S and T sets, and changing the $\textsc{Img}$ function with the $\textsc{PreImg}$ computation.

Approximate Traversals [4, 5] are a popular way to extend the applicability of reachability analysis to larger circuits.

The approach is based on the *approximate image* ($\textsc{Img}^+$) operator, returning over-estimations of exact images:

$$\textsc{Img}^+(\mathsf{TR}, R) \supseteq \textsc{Img}(\mathsf{TR}, R)$$

Notice that although R^+ represents more states, its BDD representation is usually much smaller and simpler than R, as many mutual interactions and dependences among state variables disappear because of the approximation. As a final remark, let us remember that the limit of approximate techniques is that they allow a sufficient but not necessary check, i.e., they can prove equivalence but they cannot disprove it.

3 Related Works

With the advent of SAT-based BMC tools a lot of researchers have tried to compare SAT-based method with more traditional BDD-based methods. To this respect different researchers agree that the two approaches are essentially complementary. For example in [6] the author compare BDD-based and SAT-based using a new algorithm solving unbounded model checking problems. As a conclusion he shows that performance strictly depends on the problem instances and no clear winner can be drawn at least at the moment. Driven by the same conclusions, other researchers tried to combine the two approaches. In [7] the authors perform reachability analysis by using a SAT engine to create and manipulate a disjunctive partition of the transition relation and BDDs to represent state sets and deal with them.

To extend approximate traversals to complete checks, a lot of researchers have somehow mixed approximate, exact, forward, and backward traversals [8, 9]. In [8] authors use a combination of approximate forward and backward reachability analysis. The proposed algorithm attempts to prove the mutual reachability between initial and failure states by iteratively performing over-approximate forward and backward traversals. Each new traversal increases the accuracy of the approximation, and the property is proved whenever a forward (backward) traversal reaches a fix-point outside its target.

Our work shares with these works the idea of focusing and guiding a final search with previously cheaper and approximate ones. In any case, our method ends with a SAT solver call as we use reachability analysis frontier sets to help the SAT solver search in its quest.

4 Proposed Methodology

The main flow or our methodology is represented in Figure 1.

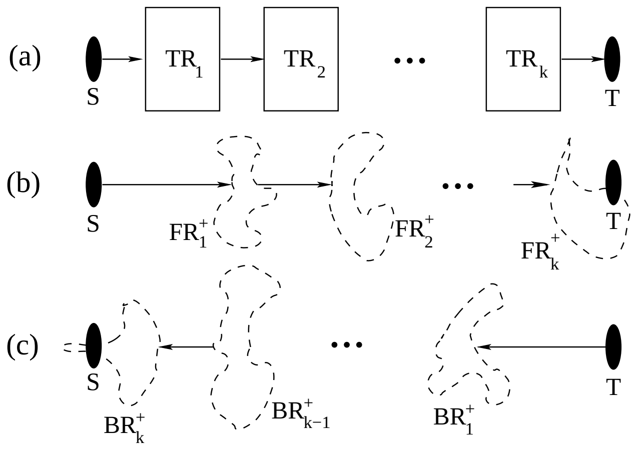


Figure 1: (a) Standard combinational unrolling for SAT-based BMC; (b) Approximate forward traversal from S to T; (c) Approximate backward traversal from T to S.

Figure 1(a) shows a graphical representation of the standard SAT-based BMC. As introduced in Section 2, to find a path of length k between S and T a combinational unrolling of the circuit representation, TR, of length k is generated. By adding the expressions for S and T, and performing a proper variable relabeling for TR a propositional formula is generated in CNF format. The SAT-engine is finally run on the resulting problem to solve it. It is useful here to remember that the value k of the bound is generally unknown. This represents one of the drawbacks of the method, as a complete verification requires checks with increasing values of k, usually reaching computational limits.

Our basic idea is to help the SAT solver with information coming from a BDD-based reachability analysis tool. In the simpler version we perform a standard forward breadth-first traversal as the one indicated in Figure 1(b) or a backward breadth-first traversal as the one indicated in Figure 1(c). In reality the applied approach is a little more sophisticated and it is detailed in Section 4.1. This phase gives an over-estimate of the paths (a "cylinder") leading from S to T so that all possible real paths are included in this over-approximation. Notice that, at this stage, we work with BDD tools, then each set of state is represented by means of BDDs[2]. From a SAT solver point of view these sets of states constitute a guide for the search. Whereas standard "conflict clauses" defines impossible assignments, our state sets define possible assignments so potentially they drastically reduce the search space for the SAT solver.

Notice that a minor contribution of applying approximate reachability analysis before the satisfiability analysis is to identify early terminations[3], saving computation time. Moreover, the approximate search may also be useful to identify an inferior limit for the value of the bound k[4]. This can be useful to avoid useless searches for impossible values of k. Finally, the over-approximation reveals to be particularly useful for high value of the bound.

[2]Depending from the kind of BDD representation/decomposition used, which we do not talk about for the lack of space, the state sets may be represented by monolith, disjunctive or conjunctive forms.

[3]Whenever the over-estimation of the reachable state set R^+ does not intersect the target set of states T the property will "pass" and in these are also the hardest cases to be proved by the SAT engine. In these cases the approximate reachability phase would consists in a preliminary and free-of-charge check.

4.1 The Approximate Reachability Analysis Phase

The approximate traversal phase is very important as far as the resulting space simplification is concerned. From a theoretical point of view, the more the approximate traversal is accurate the more the sets of states are close to the exact ones and the more the SAT solver search space is reduced. From a practical point of view, the more the approximate traversal is accurate, the larger are the BDDs representing the R_i sets and the more likely is the translation process to CNF formulas (see Section 5) to introduce a larger amount of temporary variables and clauses. As a consequence there is a trade-off between accuracy and usability of the result and this is also balanced by the cost of computing the over-estimation.

Our basic approach is to perform a forward or a backward over-approximate analysis from S to T or from T to S as described in Section 4. More in detail, we follow previous approaches on the use of forward and backward approximations [8, 10], and we adopt an iterative refinement process based on a sequence of alternate forward and backward traversals to produce better estimates. The pseudo-code of the procedure is shown in Figure 2. It proceeds through a cycle computing least fix points in the forward and backward direction computing forward FR and backward BR reachable state sets. Each forward fix point is performed by restricting the search with the BR state set and viceversa. The iteration stops when no further simplification occur or when the traversal costs, in terms of CPU time and memory usage, exceed a user-defined threshold. The COSTEVALUATE function checks hardware constraints, memory and time costs after each iteration and the process is stopped whenever the costTh threshold is exceeded.

```
FWDBACKVER (TR, S, T)
  R = FR = S
  BR = 1
  do
    FR = lfp R.(S ∨ (IMG+ (TR, R) ∧ BR))
    BR = lfp R.(T ∨ (PREIMG+ (TR, R) ∧ FR))
  while (FR ≠ BR AND COSTEVALUATE < costTh)
  return (BR)
```

Figure 2: Over-approximate Forward/Backward Traversal.

4.2 Using Symbolic State Sets Information

Once we have our set of states over-estimating a path from S to T we can apply two possible strategies:

- Adding the new information to the original CNF problem.
- Using each single "ring" of the estimate, R_i^+, to simplify the circuit representation at the corresponding time frame and then create the CNF problem using the simplified instances.

In the first case the approach is straightforward as it is enough to append the set of states, represented as monolithic or decomposed

[4]Notice that, to this respect, the paths found by the over-approximation are always shorter than the exact ones.

BDDs, as a set of CNF clauses. In the second case, we try to perform some simplifications before the CNF problem formulation using cofactor based techniques on the BDD representations of the circuit and the state sets.

Notice that in all the cases the counter-example eventually obtained possibly includes some temporary variables generated by the BDD to CNF translation process. So we need to bring it back to the original representation space, by quantifying out the temporary variables (see Section 5).

5 Dumping BDDs as CNF formulas

Given a BDD representing a function f in monolithic or conjunctive form, we develop three possible ways to store it as a CNF formula.

1. The first method, which we call **Single-Node-Cut**, models each BDD nodes, but the ones with both children equal to the constant node 1, as a multiplexer. Each multiplexer has two data inputs (i.e., the children nodes), a selection input (i.e., the node variable) and one output (i.e., the function value) whose value is assigned to an additional CNF variable. The final number of variables is equal to the number of original BDD variables plus the number of "internal" nodes of the BDD.
2. The **No-Cut** method creates clauses starting from f corresponds to the "off-set" (i.e., the set of cubes from the root to the terminal node zero) of the function f. Within the BDD for f, such clauses are found by following all the paths from the root node of the BDD to the constant node 0. The final number of variables is equal to the number of original BDD variables.
3. The **Auxiliary-Variable-Cut** method is a trade-off between the first two strategies. Internal variables, i.e., cut points, are added in order to decompose the BDD into multiple sub-trees each of which is stored following the second strategy. The trade-off is guided by a cut point selection strategy, and we experimented with two methodologies. In the first one, a new CNF variable is inserted in correspondence to the shared nodes of the BDD, i.e., the nodes which have more than one incoming edge. This technique, albeit reducing the total number of literals stored, can produce clauses with a high number of literals[5]. To avoid this drawback, the second method, introduces all the previously indicated cutting points more the ones necessary to break the length of the path to a maximum (user) selected value.

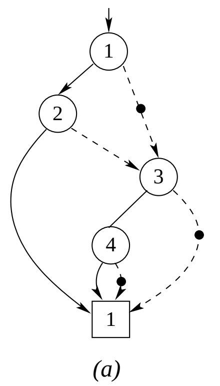
Actually, all the methods described above can be brought back to the basic idea of possibly breaking the BDD through the use of additional cutting variables and dumping the paths between the root of the BDD, the cutting variables and the terminal nodes. Such internal cutting variables are added always (for each node), never or sometimes respectively.

While the *Single-Node-Cut* method minimizes the length of the clauses produced, but it also requires the higher number of CNF variables, the *No-Cut* technique minimizes the number of CNF variables required. This advantage is counter-balanced by the fact that in the worst case the number of clauses, as well as the total number of literals, produced is exponential in the BDD size (in terms of number of nodes). The application of this method is then limited to the cases in which the off-set of the represented function f has a small cardinality. The *Auxiliary-Variable-Cut* strategy is a trade-off between the first two methods and the ones which gives more compact results. As a final remark notice that for us the compactness of the formula takes second place after the efficiency on the formula itself of the SAT engine.

[5]This value is superiorly limited by the number of variables of the BDD, i.e., the longest path from the root to the terminal node.

Example 1 *Figure 3 shows an example of how our procedure works to store a small monolithic BDD. Figure 3(a) represents a BDD with* 4 *nodes. BDD variables are named after integer numbers ranging from* 1 *to* 4*, to have an easy-to-follow correspondence with the CNF variables. Figure 3(b), (c) and (d) show the corresponding CNF representations generated by our three methods. As in the standard format* p *indicates the total number of variables used (*4 *is the minimum value as the BDD itself has* 4 *variables), and* cnf *the total number of clauses.*

As a final remark notice that for this specific example the "No-Cut" approach is the one which gives the most compact CNF representation but also the clause with the largest number of literals (4*).*



(a)

```
p cnf 7 11
-5 3 0
-5 4 0
5 -3 -4 0
6 -2 0
6 -5 0
-6 2 5 0
7 1 5 0
-7 1 -5 0
7 -1 -6 0
-7 -1 6 0
7 0
```

(b)

```
p cnf 4 3
1 -3 -4 0
-1 2 3 0
-1 2 -3 4 0
```

(c)

```
p cnf 5 5
-5 1 0
5 -1 2 0
-3 -4 5 0
3 -5 0
-3 4 -5 0
```

(d)

Figure 3: (a) BDD; (b) "Single-Node-Cut" format; (c) "No-Cut" format; (d) "Auxiliary-Variable-Cut" format.

6 Experimental Results

Our experimental set-up is made up of three distinct phases.

During the first phase, we start from the ISCAS'89 and ISCAS'89–addendum benchmark circuits in a Verilog or blif format, and the properties taken from [10]. From these source files we generate the BMC-CNF formulation of the problem using four different tools: the publicly available VIS, NuSMV and BMC and an home-made generator. As far as our package is concerned it is able to generate CNF formulas both from the original network of the circuit and from its transition relation representation. The generated CNF problem is stored as a standard DIMACS CNF file. While NuSMV, VIS and our tool produce similar results, BMC usually produces CNF files 20-30% more compact both in terms of clauses and (intermediate) variables. Nevertheless the BMC tool does not store the variable correspondence, which we need to be congruent with our reachability analysis information, as a consequence we do not report further experiments with this tool at the moment. Among the other, we always report the best results in term of performance of the SAT solver.

During the second phase, we generate the set of approximate reachable state sets for the circuit. In this phase we use both the

VIS tool and again our home-made tool. Albeit VIS implements almost all the approximate traversal algorithms presented in the literature, we need the over-approximation of the reachable state set at the same bound level for all sub-machines. As a consequence we need variant of the original Machine By Machine (MBM) algorithm with or without overlapping projections. Our tool, implemented on top of the Colorado University Decision Diagram (CUDD) package, implements the approximation verification method presented in Section 4.1. Once we have generated the BDDs for the over-approximation we store them in different format following the methodology reported in Section 5.

During the third and last phase we run the CHAFF sat engine (both the MCHAFF and the ZCHAFF versions) on the two problem instances, i.e., the original problem formulation and the one generated by merging in the information coming from the reachability analysis phase. Notice that in all the cases CHAFF is run with the default settings.

Our experiments ran on a Pentium IV 1700 MHz Workstation with 1 GByte main memory, running RedHat Linux 7.1.

Table 1 and 2 report our results. The meaning of the columns for the two tables is the following. # SV is the number of state variables in the model. # Clauses, # Vars and # Lits represent respectively the number of clauses, variables and literals in the problem (they are reported as an absolute value or as a relative ones). D.M. represents the method we used to dump the BDDs for the states obtained during the approximate traversal ("S" stands for the Single-Node-Cut method, "N" for the No-Cut and "A" for the Auxiliary-Variable-Cut).

# Decs and # Confl. represent the total number of decisions taken and conflicts produced by the sat solver. Finally, Mem. and Time indicate respectively memory occupation (in MBytes) and CPU time (in seconds). As far our technique is concerned the Setup time includes the one to perform the reachability analysis phase and the one to generate the new problem formulation, and the reported memory is mainly the one used by the SAT solver as the one used during the traversal phase is usually much smaller.

For circuit s1512 we report some comparison among different possible settings. First of all, for the property P_3, we store reachable state sets as CNF formulas using the different implemented methods (rows labeled A, S, and N). Secondly, we present some results targeting the influence of the accuracy of R on the CNF problem size. Rows S^*, N^* and A^* report results obtained with a more approximate traversal than the one used for rows *S*, *N* and *A*. For this property the set of generated clauses is larger then the original set from about 8% to more than 1000% in the worst case. More approximate reachable state sets give more compact representation as initially supposed. In Table 2 we report only the best SAT results, which is in correspondence of the *A* storing method of Table 1.

Among the other experiments performed, we try to add BDD traversal information to the original problem formulation in different way (at the bottom of the file, at the top, each reachable state set exactly before or after the relative transition relation, following the original order (from S to T) or following a reverse order, etc.). We obtain small differences, in the range of 10-20% in terms of CPU time, and, for the lack of space, we do not report evidence on that issue.

Moreover we also try to guide the SAT solver variable selection, in the Variable Decision phase, with some indications on the variable order and variable coupling derived from the reachability analysis tool. Also in this case we have slight variations in the SAT engine performances, in the order of about 10%, and we again do not report evidence on that.

7 Conclusions and Future Works

In this paper we propose to exploit inexpensive symbolic approximate forward and/or backward reachability analysis to restrict the overall search space of a SAT-solver engine.

We experimentally compare the resulting problem formulation with the original one and show its power in term of problem simplification and generality.

Among the possible future work we surely need some more experimental work on public domain and industrial benchmarks and difficult-to-find bugs. Moreover, we would like to investigate smarter way to prune the SAT-engine search space using the information coming from the over-approximate estimate.

References

[1] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an Efficient SAT Solver. In *Proc. 38th Design Automat. Conf.*, Las Vegas, Nevada, June 2001.

[2] E. Goldberg and Y. Novikov. BerkMin: a Fast and Robost SAT-Solver. In *Proc. Design Automation & Test in Europe Conf.*, Paris, France, February 2002.

[3] L. Zhang and S. Malik. The Quest for Efficient Boolean Satisfiability Solvers. In Ed Brinksma and Kim Guldstrand Larsen, editors, *Proc. Computer Aided Verification*, volume 2404 of *LNCS*, pages 17–36, Cophenagen, Denmark, 2002.

[4] H. Cho, G. D. Hatchel, E. Macii, B. Plessier, and F. Somenzi. Algorithms for Approximate FSM Traversal Based on State Space Decomposition. *IEEE Transactions on CAD*, 15(12):1465–1478, December 1996.

[5] S. G. Govindaraju, D. L. Dill, A. Hu, and M. A. Horowitz. Approximate Reachability Analysis with BDDs using Overlapping Projections. In *Proc. 35th Design Automat. Conf.*, pages 451–456, San Francisco, California, June 1998.

[6] K. L. McMillan. Applying SAT Methods in Unbounded Symbolic Model Checking. In Ed Brinksma and Kim Guldstrand Larsen, editors, *Proc. Computer Aided Verification*, volume 2404 of *LNCS*, pages 250–264, Cophenagen, Denmark, 2002.

[7] A. Gupta, Z. Yang, P. Ashar, and A. Gupta. SAT–Based Image Computation with Application in Reachability Analysis. In *Proc. Formal Methods in Computer-Aided Design*, volume 1954 of *LNCS*, Austin, TX, USA, 2000.

[8] S. G. Govindaraju and D. L. Dill. Verification by Approximate Forward and Backward Reachability. In *Proc. Int'l Conf. on Computer-Aided Design*, pages 366–370, San Jose, California, November 1998.

[9] I. Moon, J. Jang, G. D. Hachtel, F. Somenzi, J. Yuan, and C. Pixley. Approximate Reachability Don't Cares for CTL Model Checking. In *Proc. Int'l Conf. on Computer-Aided Design*, pages 351–358, San Jose, California, November 1998.

[10] G. Cabodi, P. Camurati, and S. Quer. Can BDDs compete with SAT solvers on Bounded Model Checking? In *Proc. 39th Design Automat. Conf.*, New Orleans, Louisiana, June 2002.

Model	# SV	Property	Bound	Original SAT Problem			Modified SAT+BDD Problem			
				# Clauses [k]	# Vars [k]	# Lits [k]	D.M.	# Clauses [%]	# Vars [%]	# Lits [%]
s1512	57	P_1 (pass)	66	92	36	222	A	+ 21.7 %	+ 7.5 %	+ 72.1 %
		P_1 (fail)	67	94	37	225	A	+ 22.0 %	+ 7.6 %	+ 73.2 %
		P_2 (pass)	130	181	70	436	A	+ 53.4 %	+ 20.0 %	+ 168.5 %
		P_2 (fail)	131	183	71	440	A	+ 53.4 %	+ 20.0 %	+ 169.4 %
		P_3 (pass)	259	381	146	917	A	+ 115.0 %	+ 40.0 %	+ 317.7 %
		P_3 (fail)	260	383	147	920	A	+ 115.3 %	+ 40.0 %	+ 318.6 %
							S	*+ 218.3 %*	*+ 153.9 %*	*+ 257.4 %*
							N	*+ 1361.1 %*	*+ 0.0 %*	*+ 13536.2 %*
							A^*	*+ 7.7 %*	*+ 3.3 %*	*+ 13.5 %*
							S^*	*+ 13.9 %*	*+ 10.2 %*	*+ 15.7 %*
							N^*	*+ 56.6 %*	*+ 0.0 %*	*+ 373.9 %*
s9234	211	P_1 (pass)	80	598	214	1443	S	+ 14.6 %	+ 13.4 %	+ 14.5 %
		P_1 (fail)	81	606	217	1461	S	+ 14.6 %	+ 13.4 %	+ 14.5 %
		P_2 (pass)	80	598	214	1443	S	+ 14.6 %	+ 13.4 %	+ 14.5 %
		P_2 (fail)	81	606	217	1461	S	+ 14.6 %	+ 13.4 %	+ 14.5 %
		P_3 (pass)	80	598	214	1443	S	+ 14.6 %	+ 13.4 %	+ 14.5 %
		P_3 (fail)	81	606	217	1461	S	+ 14.6 %	+ 13.4 %	+ 14.5 %
s15850.1	534	P_1 (pass)	75	1057	387	2572	A	+ 2.0 %	+ 0.4 %	+ 13.0 %
		P_1 (fail)	76	1071	391	2607	A	+ 2.0 %	+ 0.4 %	+ 13.0 %
s13207.1	638	P_1 (pass)	55	465	188	1151	A	+ 21.8 %	+ 5.4 %	+ 76.7 %
		P_1 (fail)	56	474	191	1172	A	+ 21.8 %	+ 5.4 %	+ 76.6 %
		P_2 (pass)	109	920	369	2278	A	+ 21.2 %	+ 5.3 %	+ 74.5 %
		P_2 (fail)	110	929	372	2299	A	+ 21.2 %	+ 5.3 %	+ 75.2 %
s35932	1728	P_1 (pass)	31	1495	537	3630	A	+ 4.4 %	+ 1.0 %	+ 33.9 %
		P_1 (fail)	32	1543	554	3747	A	+ 4.3 %	+ 1.0 %	+ 33.2 %

Table 1: Comparison between original and modified CNF problems. For circuit s1512, the rows S^*, N^* and A^* report results obtained with a more approximate traversal than the one used for the previous three lines *S*, *N* and *A*.

Model	Property	Bound	Original SAT Problem				Modified SAT+BDD Problem				
			# Decs	# Confl.	Mem.	Time [s]	# Decs	# Confl.	Mem.	Time [s]	
			[k]	[k]	[MByte]	Search	[k]	[k]	[MByte]	Setup	Search
s1512	P_2 (pass)	66	106	57	28	74	3	1	17	2	2
	P_2 (fail)	67	155	97	28	169	20	1	17	2	5
	P_4 (pass)	130	566	318	85	899	157	73	51	5	104
	P_4 (fail)	131	707	415	84	1326	149	59	50	5	76
	P_5 (pass)	259	3469	2018	172	11330	415	163	146	19	390
	P_5 (fail)	260	4502	2579	299	18677	609	230	97	19	603
s9234	P_1 (pass)	80	167	133	123	517	23	10	95	14	27
	P_1 (fail)	81	275	221	122	950	35	14	71	14	45
	P_2 (pass)	80	135	111	90	426	20	9	95	14	25
	P_2 (fail)	81	127	87	90	256	42	18	71	14	49
	P_3 (pass)	80	259	211	123	813	24	10	95	14	29
	P_3 (fail)	81	245	198	122	755	27	13	95	14	32
s15850.1	P_1 (pass)	75	25	20	175	184	8	6	143	53	35
	P_1 (fail)	76	475	120	140	1227	173	43	141	53	273
s13207.1	P_1 (pass)	55	22	11	55	22	24	16	74	54	53
	P_1 (fail)	56	641	83	46	375	758	35	59	54	116
	P_2 (pass)	109	315	230	193	4287	98	51	133	96	176
	P_2 (fail)	110	640	358	158	3210	2348	109	115	96	432
s35932	P_1 (pass)	31	0	0	205	9	0	0	219	175	10
	P_1 (fail)	32	206	40	201	1156	119	25	181	175	760

Table 2: Comparison between output statistics produced by CHAFF for the original and the modified CNF problems. Memory limit 900 MBytes. Time limit 36000 sec.