Can BDDs compete with SAT solvers on Bounded Model Checking?

Gianpiero Cabodi Paolo Camurati Stefano Quer

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

ABSTRACT

The usefulness of Bounded Model Checking (BMC) based on propositional satisfiability (SAT) methods has recently proven its efficacy for bug hunting. BDD based tools are able to verify broader sets of properties (e.g. CTL formulas) but recent experimental comparisons between SAT and BDDs in formal verification lead to the conclusion that SAT approaches are more robust and scalable than BDD techniques.

In this work we extend BDD-based verification to larger circuit and problem sizes, so that it can indeed compete with SAT-based tools. The approach we propose solves Bounded Model Checking problems using BDDs. In order to cope with larger models it exploits approximate traversals, yet it is exact, i.e. it does not produce false negatives or positives. It reaps relevant performance enhancements from mixed forward and backward, approximate and exact traversals, guided search, conjunctive decompositions and generalized cofactor based BDD simplifications.

We experimentally compare our tool with BMC in NuSMV (using mchaff as SAT engine), and we show that BDDs are able to accomplish large verification tasks, and they can better cope with increasing sequential depths.

1. INTRODUCTION

BDD-based symbolic manipulation has been one of the most widely used core technologies in the verification domain, over the last decade. Recently, however, this methodology has been challenged by approaches using propositional satisfiability (SAT) solvers for Bounded Model Checking (BMC) problems. The basic goal of BMC is to disprove properties by searching counter-examples of bounded length.

This has shown great efficacy in bug hunting whenever productivity is a key issue (e.g. in industrial settings). Several recent papers have compared BDD and SAT based model checking and worked at mixed approaches. Even though no definite conclusion can be drawn, researchers and engineers agree that SAT tools are complementary to BDD based ones, and they are able to easily solve many model checking hard problems for BDDs. Quite often SAT-based BMC performs well compared to BDD-based model checking, even though it shows its limits when facing sequentially deep bugs.

Our main target in this work is not to confute or debate all good qualities of SAT based BMC. We rather explore possible ways to complement it with BDD based symbolic verification, and we use SAT based BMC as a term of comparison for experimental valuation.

In order to achieve the proposed goal, we follow some of the trends proposed in the past years to face the BDD explosion problem. We exploit *approximate traversals* to simplify exact verification, we follow the *guided search* paradigm when combining *forward and backward* traversals to mutually focus and narrow their search area, we finally exploit conjunctive partitioning and cofactor based simplifications to keep BDD sizes under control.

Some major contributions of our approach are:

- A new BDD based verification strategy for Bounded Model Checking. Bounding the sequential depth of the verification problem enables us to specifically target optimizations for individual traversal iterations
- A technique for exact verification (without false negatives or positives), exploiting approximate traversals as a way to drive and drastically simplify bug hunting efforts in exact symbolic traversals.
- A set of optimizations within image and pre-image procedures, sharing the common goal of keeping BDDs as much partitioned and simplified as possible, in order to avoid memory blow-ups.

Our experimental results show that we are able to check properties on circuits outside the scope of state-of-the-art BDD based verification tools. A further minor contribution of our work is to introduce an automatic property generation strategy that allows us to compare invariant verification tools on sequential models or circuits of unknown functionality and/or with no available property.

2. PRELIMINARIES

Our verification framework handles Finite State Machines described by their transition relation TR with initial state set S. We check an invariant property P by attempting to prove (or disprove) the reachability of its complement T (target state set) from S.

DAC 2002, June 10-14, 2002, New Orleans, Louisiana, USA.

Standard BDD-based *exact forward verification* (Figure 1) is a breadth-first least fix-point visit of the state space that starts from S and tries to find a path to T.

On the fly tests for intersection with target are done at each iteration, thus avoiding full computation of reachable states whenever the property is violated (T is reached) before the fix-point. Conterexamples are computed working on the array of state sets R.

```
FwdVer (TR, S, T)
    k = 0
    R_k = New = S
    while (New ≠ ∅)
        if ((T · New) ≠ ∅)
            return (CounterEx (R))
        k = k + 1
        Next = Img (TR, New)
        New = Next · R̄_{k−1}
        R_k = R_{k−1} + New
    return (PASS)
```

Figure 1: Breadth-First Exact Forward Verification.

The basic idea in SAT based BMC is to consider only paths of bounded length k and to construct a propositional formula that is satisfiable iff there is a counter-example (a path from S to T) of the same length. For that reason it targets falsification and partial verification rather than full verification. In order to fully verify a property one needs to look for longer and longer counter-examples.

The propositional formula describing a path from s_0 to s_k requires that there is a sequence of transitions from s_i to s_{i+1} for all $0 \leq i < k$:

$$path(s_0, \ldots, s_k) = \mathsf{TR}(s_0, s_1) \cdot \ldots \cdot \mathsf{TR}(s_{k-1}, s_k)$$

BMC tools implement different checks, characterized by decreasing complexity and verification capability.

The first one, referred to as bound k, looks for counter-examples of any length from 0 to k:

$$(s_0 = \mathsf{S}) \cdot path(s_0, \ldots, s_k) \cdot (\mathsf{T} \cdot s_0 + \ldots + \mathsf{T} \cdot s_k)$$

The second one, referred to as exact k, looks for paths of length exactly equal to k:

$$(s_0 = \mathsf{S}) \cdot path(s_0, \ldots, s_k) \cdot \mathsf{T} \cdot s_k$$

The third one, referred to as exact-assume k, looks for paths of length K strictly reaching T only at the last cycle:

$$(s_0 = \mathsf{S}) \cdot path(s_0, \ldots, s_k) \cdot (\overline{\mathsf{T} \cdot s_0}) \cdot \ldots \cdot (\overline{\mathsf{T} \cdot s_{k-1}}) \cdot (\mathsf{T} \cdot s_k)$$

Both BDD and SAT based verification tasks are usually preceded by model decompositions, reductions (e.g., Cone of Influence - COI), and abstractions, often able to bring down even very large problems to manageable sizes.

Let us finally note that in this work we use $\downarrow$ to denote a generalized cofactor function, defined as:

$$(f \downarrow g)(x) = \begin{cases} f(x) & \text{if} \quad g(x) = 1 \\ - & \text{if} \quad g(x) = 0 \end{cases}$$

In other words, g can be viewed as care set for f, that can be arbitrarily simplified in the domain subspace where $g = 0$. Due to this degree of freedom, several cofactor operators have been proposed in the literature. "Constrain" cofactor was proposed as image restrictor for symbolic manipulations and canonical decompositions, whereas "restrict", and other function simplification oriented operators, were mainly conceived for logic synthesis optimizations. Since our main purpose is function simplification and BDD reduction given a care set, we use the restrict cofactor in our implementation.

```
BoundedTraversal (TR, S, T, k)
    Frontier_0 = S
    for (i = 0; i < k; i++)
        if (Frontier_i · T ≠ ∅)
            return (FAILURE)
        Frontier_{i+1} = Img (TR, Frontier_i)
    return (PASS)
```

Figure 2: Bounded Traversal.

3. RELATED WORKS

In [1] we combined approximate forward and exact backward reachability analysis. The proposed methodology was applied to sequential equivalence verification, and it was used for Automatic Test Pattern Generation of stuck-at faults. The original implementation used a static variable ordering framework, with image and pre-image computations based on transition functions. Our work follows [1] in its idea of combining approximate forward and exact backward traversals, using frontier sets computed in the forward direction to guide and simplify backward traversals. We extend the methodology to bounded invariant checking. Moreover, we exploit optimized image and pre-image procedures based on transition relations, conjunctive partitioning and cofactor based simplifications inspired by [2].

In [3] 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. False negatives are possible because counter-examples are generated with approximate state sets. A major contribution of their work is the use of overlapping projections to obtain tighter approximations than with disjoint partitions. Our work shares with them the idea of focusing and guiding more accurate traversals with previously done approximate ones. But our method ends with an exact traversal (we have no false negatives), and we use individual frontier sets instead of fix-point reached states as tighter constraints for symbolic search.

4. BDD BASED BOUNDED MODEL CHECK

A straightforward implementation of BDD based BMC is presented in [4]. The proposed algorithm is an adaptation of standard breadth-first search. TR is the transition relation, S is the start (init) state set, T is the target set (the set of bad states).

The adoption the "bounded" search paradigm removes the fix-point check and the necessity to store the reachable state set (see Figure 2). Memory efficiency is thus increased. Moreover, the authors fully integrated the implementation with a threshold based partitioning strategy called "prioritized-traversal",

Our approach is similar to [4] in its main loop, but we address more aggressive strategies to reduce the BDD size of frontier sets. Some of the optimizations work at the level of traversal iterations, and they are shown in this section. Other ones are discussed in the next section, as they are related to inner steps of image/preimage computations.

Our starting idea is to combine exact and approximate forward and backward search. One (or more) initial approximate traversal(s) is (are) used to guide and simplify an exact verification task. In order to represent (and differentiate) the various state sets involved in different traversal strategies, we adopt the following notations:

- F_i: Generic frontier set in exact forward traversal starting from initial state set S ($\mathrm{F}_0 = \mathsf{S}$). Index i increases from S to T. We do not compute F in our approach, but we use it as a term of comparison while describing the combined forward-backward traversals.
- B_i: Frontier sets in exact $k-$bounded backward traversal starting from target state set T ($\mathrm{B}_k = T$). Index i decreases from T to S. We do not explicitly compute B, but we use it the same way as F.
- $Step_i$: Set of states that can be found at the $i-$th step of paths connecting S to T. This set can be formally represented as the intersection between forward and backward frontiers ($Step_i = \mathrm{F}_i \bigcap B_i$).
- F_i^+: Frontier sets in over-approximate forward traversal starting from initial state set S ($\mathrm{F}_0 = S$).
- FB_i: Frontier sets in exact backward traversal over approximate forward frontiers (F_i^+). Computation of FB sets is guided and simplified using previously computed F^+ sets.

In the sequel we first introduce a BMC procedure (ExactFwdBwdBMC, Figure 3) which corresponds to the bound k check of [4] (see section 2). We then introduce other bounded verification procedures as variations of the first one. The ExactFwdBwdBMC procedure is able to make checks on paths of length k from S to T, so it might miss shorter mismatch sequences. As in SAT based BMC this is may be a good trade-off accuracy for performance. We describe it in detail, with proofs of correctness and a discussion on performance issues.

```
ExactFwdBwdBMC (TR, S, T, k)
    F0+ = S
    for (i = 1; i ≤ k; i++)
        Fi+ = Img+ (TR, F(i-1)+)
    FBk = T ↓ Fk+
    for (i = k; i > 0; i−−)
        FB(i-1) = PreImg (TR, FBi) ↓ F(i-1)+
    if (F0+ · FB0 = 0)
        return (PASS)
    else
        return (CounterEx (TR, F+, FB ))
```

Figure 3: Forward Backward Exact BMC.

The procedure first computes over-approximate forward frontier sets. Each entry in the F^+ array overestimates the frontier set of an exact traversal:

$$\mathrm{F}_i^+ \supseteq \mathrm{F}_i$$

Entries in the FB array represent the frontier sets of an exact bounded backward traversal (B_i), each one simplified using the corresponding forward entry as care set.

$$\mathrm{FB}_i = \mathrm{B}_i \!\downarrow \mathrm{F}_i^+$$

Indexes in backward traversal are used from k down to 0 so that corresponding frontiers in forward and backward directions have the same index. The generic $\mathrm{FB}_i \cdot \mathrm{F}_i^+$ set includes all states in $Step_i$:

$$\mathrm{FB}_i \cdot \mathrm{F}_i^+ \supseteq Step_i$$

So the combined FB and F^+ arrays implicitly include all paths connecting S to T. Furthermore, all states in $\mathrm{FB}_i \cdot \mathrm{F}_i^+$ are guaranteed to be backward reachable from T:

$$\mathrm{FB}_i \cdot \mathrm{F}_i^+ \subseteq B_i$$

The forward-backward algorithm guarantees exact bounded verification, as expressed by the following theorem:

Theorem 1. *Let* FB *be the array of frontier sets produced by the* ExactFwdBwdBMC *procedure, then*

$$\mathrm{FB}_0 \cdot S = 0 \Leftrightarrow \mathsf{InvariantTrue}$$

A key issue for performance is keeping BDD sizes under control by means of generalized cofactor simplifications and threshold based control over conjunctions.

- Frontier sets in approximate forward traversals (F_i^+) are "clustered" on a threshold basis. Each set is a conjunction of terms with disjoint [1] support: $\mathrm{F}_i^+ = \prod_{j \in Groups} f_i^j$ (see Section 5). As in transition relation clustering, we partially compute products as far as the size of their result is under a chosen threshold.
- Accuracy of the over-approximate forward traversal is not as important as in approximate model check [3], where the goodness of a verification task heavily relies on the ability of an approximate model to represent the exact behavior. The main purpose of approximate traversal in our solution is to generate good care sets for effective BDD simplifications throughout an exact traversal. Since accuracy often implies larger BDDs, even in approximate traversals, we achieve better results with intermediate solutions (trading off accuracy for BDD size).
- Frontier sets in exact backward traversal (FB_i) are always computed and manipulated in their cofactored form, i.e. the PreImg procedure directly works with the F_i^+ set in order to simplify $\mathrm{FB}_i = \mathrm{B}_i\!\downarrow \mathrm{F}_i^+$ while computing it (see next paragraph). Furthermore, FB_i sets are generated in conjunctively decomposed forms, using a technique derived from [2].

A second procedure we adopt is an extension of ExactFwdBwdBMC, able to find existing minimum length counterexamples within the chosen k bound (see exact k, section 2). The procedure (FwdBwdBMC) is shown in Figure 4.

Forward frontier sets are here over-approximations of reachable state sets: $\mathrm{F}_i^+ \supseteq R_i$. This is due to the fact that at each iteration we take the union of the approximate image with the initial state set S, so that the i-th frontier set includes all states reachable from S in at most i steps.

Moreover, the check for failure is done (on the fly) at each backward iteration, and it guarantees minimal length

[1] Overlapping projections are also possible, as adopted in [1, 3], but our present implementation is limited to an approximate image procedure with non-overlapping components.

$\textsc{FwdBwdBMC}$ (TR, S, T, k)
  $F_0^+ = S$
  for (i = 1; i ≤ k; i++)
    $F_i^+ = \textsc{Img}^+ (\mathsf{TR}, F_{i-1}^+) + \mathbf{S}$
  $FB_k = T \downarrow F_k^+$
  for (i = k; i > 0; i−−)
    $FB_{i-1} = \textsc{PreImg} (\mathsf{TR}, FB_i) \downarrow F_{i-1}^+$
    if $(S \cdot FB_i \neq 0)$
      return ($\textsc{Trace}$ (TR, $F_{i..h}^+$, $FB_{i..k}$)
  return (PASS)

Figure 4: Forward Backward BMC.

counter-examples. It is worth noticing that FwdBwdBMC has the same cost of ExactFwdBwdBMC whenever the initial state set is included in the image of the initial state set itself, so the F^+ arrays in the two procedures are exactly the same. The only extra effort paid by FwdBwdBMC is testing intersection with the initial set at each backward iteration.

A third verification procedure (FwdBwdFwdBMC, see Figure 5) is based on a couple of forward and backward approximate traversals, followed by a final exact forward one. Here we can exploit the combined contribution of approximate forward (F^+) and backward (FB^+) frontiers to guide exact search through a narrower subspace. Moreover, the initial part of this procedure has an effect similar to the approximate forward-backward verification of [3] in the case of successful checks (property is proved correct right after the approximate traversals).

$\textsc{FwdBwdFwdBMC}$ (TR, S, T, k)
  $F_0^+ = S$
  for (i = 1; i ≤ k; i++)
    $F_i^+ = \textsc{Img}^+ (\mathsf{TR}, F_{i-1}^+ + S)$
  $FB_k^+ = T \downarrow F_k^+$
  for (i = k; i > 0; i−−)
    $FB_{i-1}^+ = \textsc{PreImg}^+ (\mathsf{TR}, FB_i^+) \downarrow F_{i-1}^+$
  $FBF_0 = F_0^+ \cdot FB_0^+$
  for (i = 1; i ≤ k; i++)
    $FBF_i = \textsc{Img} (\mathsf{TR}, FBF_{i-1} \cdot F_i^+) \downarrow FB_i^+$
    if $(S \cdot FBF_i \cdot FB_i^+ \neq 0)$
      return ($\textsc{Trace}$ (TR, $FBF_{0..i}$, $FB_{0..i}^+$))
  return (PASS)

Figure 5: Forward Backward Forward BMC.

5. PERFORMANCE ISSUES IN IMAGE AND PRE-IMAGE COMPUTATIONS

Let us concentrate now on the inner steps of image and pre-image computations, involving some of the most effective optimizations for the overall performance.

Approximate image is computed as

$$F_i^+ = \textsc{Img}^+(\mathsf{TR}, F_{i-1}^+) = \prod_{j \in Groups} \textsc{Img}(\mathsf{TR}_j, F_{i-1}^+)$$

where

$$\mathsf{TR} = \prod_{j \in Groups} \mathsf{TR}_j$$

is the clustered transition relation (each group is in turn a statically generated product of clusters). Partial images (of individual group) are conjoined under threshold control, so the result F_i^+ is a conjunctively partitioned BDD.

Exact pre-image exploits two levels of partitioning and cofactor simplifications. First of all, the cofactoring term F_{i-1}^+ (see Figure 3) is given as a parameter to the PreImgWithCare function, in order to be used as a care set for inner operations:

$$\begin{aligned} FB_{i-1} &= \textsc{PreImg}(\mathsf{TR}, FB_i)\downarrow F_{i-1}^+ \\ &= \textsc{PreImgWithCare}(\mathsf{TR}, FB_i, F_{i-1}^+)\downarrow F_{i-1}^+ \end{aligned}$$

The latter function is implemented as the classical linear "and-exist" (or "relational product") with early quantification, where the generic step (computing a new intermediate product $P_j = \exists_{x_j}(P_{j-1} \cdot \mathsf{TR}_j)$) is optimized as follows

$$\begin{aligned} P_j &= \exists_{x_j}(P_{j-1} \cdot \mathsf{TR}_j)\downarrow F_{i-1}^+ \\ P_j &= \exists_x P_j \cdot (P_j \downarrow \exists_x P_j) \end{aligned}$$

Each partial product is first cofactored with the care set F_i^+, then the basic decomposition step of [2] is applied, exploiting the sets of early quantification variables as variable layers driving the decomposition (instead of the variable ordering). The latter step produces a conjunctively decomposed pre-image, that is finally clustered under threshold control.

Approximate pre-images in FwdBwdFwdBMC (see Figure 5) are computed like approximate images, but we adopt for them a fully dynamic group selection strategy, driven by BDD size (as for clustering), since static pre-computation of groups gave too rough over-approximations in the backward mode: Preimage is exact as far as the size of its intermediate result is below a threshold, otherwise it switches to approximate by dynamically restarting a new group.

6. EXPERIMENTAL RESULTS

The presented technique is implemented in a program called FBV (Forward-Backward Verifier), running on top of the Colorado University Decision Diagram (CUDD) package. In this section we present an experimental comparison between this tool and the SAT-based BMC tool NuSMV [5] (version 2.0.2), using both the internal and a state-of-the-art SAT solver, mchaff [6], as slave engines. We also experimented with the BDD-based verification procedure of NuSMV and VIS, but we do not report data on these runs since they could only verify the smaller presented models.

We show data for a few ISCAS'89 benchmarks and some other circuits and models taken from [7] and from the NuSMV distribution [5]. They have different sizes, most of them outside the range of problems manageable by state-of-the-art BDD based verifiers. The models are originally available in different formats (blif, Verilog, SMV), so to cope with them we implemented or modified a set of translators working in different directions.

Properties (invariants) are either available with the original description or automatically generated (e.g., for the ISCAS'89 circuits) as described in the following paragraph. Albeit artificially generated properties may not be as meaningful as designer given ones, that was the only way we had to cope with the lack of publicly available and suitable benchmarks. For any given property we performed a COI reduction before starting the verification phase.

6.1 Automatic invariant property generation

Automatic invariant property generation was adopted for two reasons:

1. We had no property and no specific knowledge about the functionality of the ISCAS benchmarks,
2. We aimed at measuring the ability of a verifier to explore arbitrary properties in the state space.

We thus generated target state sets T as sets with increasing Hamming distance from the initial state set S. The invariant property requires the T states to be unreachable.

As an example, let us suppose to have a model with 5 state variables $(d_0, d_1, d_2, d_3, d_4)$. and initial state $\mathsf{S} = (0,0,0,0,0)$. We have just one state with Hamming distance 5 from S, i.e, $(1,1,1,1,1)$, we have 6 states at Hamming distance ≥ 4, i.e, $(1,1,1,1,1)$, $(0,1,1,1,1)$, $(1,0,1,1,1)$, $\ldots$, $(1,1,1,1,0)$, and so on so forth. In the first case our invariant property is $\overline{(d1 \cdot d2 \cdot d3 \cdot d4 \cdot d5)}$. Our experience is that the shorter the Hamming distance from the initial state, the easier is to falsify the property, whereas most properties generated using high Hamming distances are proved correct (i.e. state space regions distant from initial state are unreachable).

6.2 Experimental data

Our experiments ran on a 600 MHz Alpha IV5 Workstation with 512 MByte of main memory, running RedHat Linux. We compare the results attained by our BDD based tool (FBV) with NuSMV – mchaff.

Figure 6 shows a time comparison on a couple of selected test cases. We plot execution time for the same property checked with increasing values of the bound.

SAT-based BMC are extremely efficient with small values of the bound, but FBV is more performing with larger bounds. We show data for both mchaff and the default/internal SAT solver in NuSMV (NuSmv-SAT). In both the cases NuSMV is used as a front-end to generate the CNF formulation of the problem. The overall behavior seems to confirm the expected degradation in terms of performance of SAT solvers with increasing values of the bound. This is intuitively explained by the linear increment of the number of variables and clauses, directly affecting the depth-first decision procedure. In the case of BDDs no additional variable is required for new traversal iterations, and performance is strictly related to the total amount of BDD nodes. As a consequence our technique is more scalable as far as our optimizations are able to avoid BDD blow-up.

The above observations are confirmed by all data in Table 1, where we report a performance comparison (over a broader set of circuits) in terms of both memory and CPU time. Models are sorted by the original (before the COI extraction) number of state variables (column # SV). Properties proved correct within the chosen bound are denoted by *pass*, properties falsified by *fail*. We avoid reporting data for the NuSmv-SAT SAT solver, since mchaff always outperformed it. Columns # Var. and # Clauses indicate the size of the generated SAT problems, in terms of CNF variables and clauses. The CPU time is reported in different columns. The Setup column indicates the time spend to generate the CNF clauses for the SAT solver or the BDDs and the transition relation for the FBV tool. SAT reports the running time for mchaff. Trav shows the time spent by FBV on symbolic traversals.

NuSMV – mchaff were unable to deal with some of the larger instances FBV solved, because of time or memory constraints. The above failures are usually related to large bound values.

7. CONCLUSIONS

We explore possible ways to face larger problems with BDD based symbolic verification and to make BDD-based tools efficient enough to compete with SAT-based tools. We propose to mix forward and backward approximate and exact traversals, guided search, conjunctive decompositions and generalized cofactor based BDD simplifications, to obtain relevant performance enhancements.

We experimentally compare our tool with state of the art BDD and SAT based model checkers. Our experience leads to the following two conclusions. First of all, we are able to deal with larger problems than other BDD-based tools. Secondly, our methodology seems to be more scalable with increasing bounds compared to SAT-based BMC tools.

To sum up, on the one hand we deem our BDD based methodology is complementary to SAT based BMC, especially when looking for sequentially deep bugs. On the other hand, SAT tools might be able to attach bugs on problems even larger than the one analysed in this paper.

8. REFERENCES

[1] G. Cabodi, P. Camurati, and S. Quer. Efficient State Space Pruning in Symbolic Backward Traversal. In *Proc. Int'l Conf. on Computer Design*, pages 230–235, Cambridge, Massachussetts, October 1994.

[2] G. Cabodi. Meta-BDDs: A Decomposed Representation for Layered Symbolic Manipulation of Boolean Functions. In Gérard Berry, Hubert Comon, and Alan Finkel, editors, *Proc. Computer Aided Verification*, volume 2102 of *LNCS*, pages 118–130, Paris, France, July 2001. Springer-Verlag.

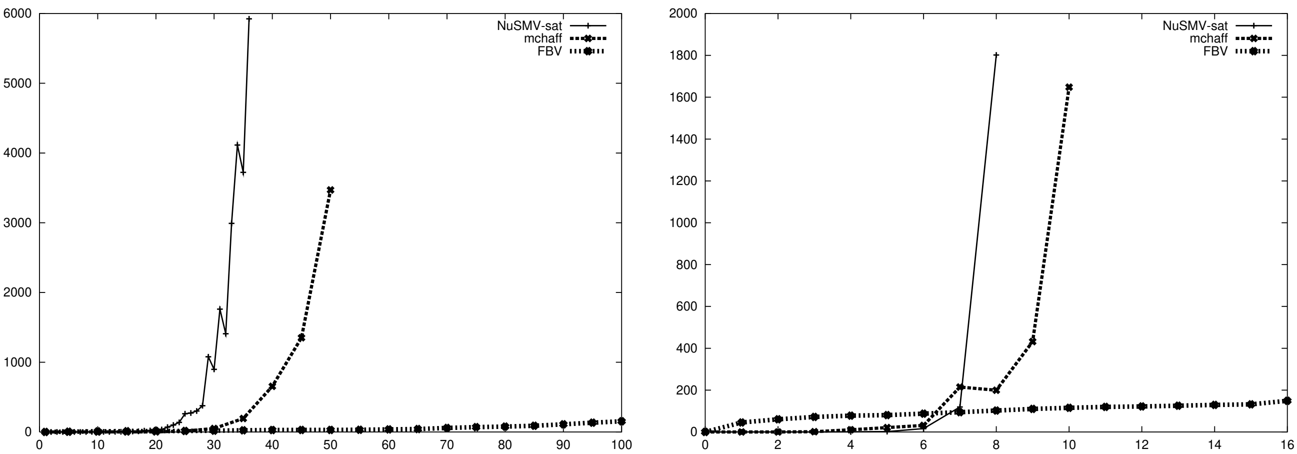
[3] S. G. Govindaraju and D. L. Dill. Verification by Approximate Forward and Backward Reachability. In *Proc. Int'l Conf. on Computer-Aided Design*, pages 366–370, San Jose, California, November 1998.

[4] F. Copty, L. Fix, R. Fraer, E. Giunchiglia, G. Kamhi, A. Tacchella, and M. Y. Vardi. Benefits of Bounded Model Checking at an Industrial Setting. In Gérard Berry, Hubert Comon, and Alan Finkel, editors, *Proc. Computer Aided Verification*, volume 2102 of *LNCS*, pages 435–453, Paris, France, July 2001. Springer-Verlag.

[5] A. Cimatti, E. M. Clarke, F. Giunchiglia, and M. Roveri. NuSMV: a new Symbolic Model Verifyer. In *Proc. Computer Aided Verification*, volume 1633 of *LNCS*, pages 495–499. Springer-Verlag, July 1999.

[6] 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.

[7] K. Ravi and F. Somenzi. Hints to Accelerate Symbolic Traversal. In *Correct Hardware Design and Verification Methods (CHARME'99)*, pages 250–264, Berlin, September 1999. Springer-Verlag. LNCS 1703.



(Circuit s9234) (Circuit am2901)

Figure 6: BMC execution time (sec) versus bound value.

Model	# SV	Property	Bound	NuSMV – mchaff					FBV		
				# Var.	# Clauses	Mem. [MByte]	Time [sec] Setup	Time [sec] SAT	Mem. [MByte]	Time [sec] Setup	Time [sec] Trav
vsaR	66	P_1 (fail)	20	46294	134479	29	1	46	60	1	24
am2901	68	P_1 (pass)	10	23488	67443	42	2	1648	11	3	111
		P_1 (fail)	16	37360	107757	–	4	ovf	15	3	145
tcas	146	P_1 (fail)	11	258875	770491	27	27	578	35	2	100
philo_{100}	200	P_1 (pass)	100	432672	1227634	ovf	38	–	57	7	1740
s9234	211	P_1 (pass)	80	213608	597565	141	34	1023	19	10	70
		P_1 (fail)	81	216266	605026	141	35	1998	38	10	151
		P_2 (pass)	80	213608	597565	109	35	853	35	10	100
		P_2 (fail)	81	216266	605026	109	36	535	70	10	189
		P_3 (pass)	80	213608	597565	142	37	1699	37	10	143
		P_3 (fail)	81	216266	605026	141	38	1497	64	10	227
		P_4 (pass)	240	639370	1792771	ovf	50	–	37	10	290
		P_4 (fail)	241	641546	1798786	ovf	30	–	66	10	387
		P_5 (pass)	240	639370	1792771	ovf	48	–	39	10	401
		P_5 (fail)	241	641546	1798786	ovf	31	–	70	10	1917
s15850.1	534	P_1 (pass)	19	99703	268876	47	5627	5	33	14	56
		P_1 (fail)	20	104826	282943	43	5653	28	40	14	167
		P_2 (pass)	75	386667	1056856	209	5775	358	35	14	239
		P_2 (fail)	76	391791	1070926	174	5650	1534	80	14	602
s13207.1	638	P_1 (pass)	29	136108	359630	64	43	23	53	15	105
		P_1 (fail)	30	140707	371965	50	44	78	64	15	302
		P_2 (pass)	55	255682	680340	117	74	128	53	15	203
		P_2 (fail)	56	260281	692675	92	74	542	71	15	506
		P_3 (pass)	109	504028	1346430	294	56	5625	59	15	413
		P_3 (fail)	110	508627	1358765	195	55	5330	108	15	1021
		P_4 (pass)	215	991522	2653940	ovf	94	–	63	15	834
		P_4 (fail)	216	996121	2666275	ovf	99	–	137	15	2086
		P_5 (pass)	429	1975708	5293630	ovf	215	–	113	15	1823
		P_5 (fail)	430	1980307	5305965	ovf	217	–	166	15	4507
s38584.1	1426	P_1 (pass)	10	169198	464752	70	52	13	114	161	451
		P_1 (fail)	11	201874	556846	63	65	42	177	161	2842
s35932	1728	P_1 (pass)	31	540248	1494490	243	63	7	125	37	293
		P_1 (fail)	32	557448	1542529	205	65	2054	177	37	3758

Table 1: Comparison between BDD-based and SAT-based Bounded Model Checking. ovf means overflow on memory or time (memory limit 500 MBytes, time limit 10000 sec). – means data not available.