Mixing Forward and Backward Traversals in Guided-Prioritized BDD-Based Verification

Gianpiero Cabodi, Sergio Nocco, and Stefano Quer

Dip. di Automatica e Informatica, Politecnico di Torino, Turin, Italy
{cabodi,nocco,quer}@polito.it
http://www.polito.it/~{cabodi,quer}

Abstract. Over the last decade BDD-based symbolic manipulations have been among the most widely used core technologies in the verification domain. To improve their efficiency within the framework of *Unbounded Model Checking*, we follow some of the most successful trends proposed in this field.

We present a very promising approach based on: Mixing forward and backward traversals, dovetailing approximate and exact methods, adopting guided and partitioned searches, efficiently using conjunctive decompositions and generalized cofactor based BDD simplifications. One of the main contributions of this paper is a backward verification procedure based on a prioritized traversal. We call the method "inbound-path-search". Initially, an approximate forward traversal produces over-approximate onion-ring frontier sets. After that, these rings are used as distance estimators and guides to partition state sets in terms of the *estimated* distance from the "target" set of states. Finally, while the subsequent search is performed, the higher priority is given to the subset with the smallest estimated distance.

We experimentally compare our methodology with a state-of-the-art technique (*approximate-reachability don't cares model checking*) implemented in the freely available VIS tool. Results show interesting improvements in terms of both efficiency and power.

1 Introduction

Binary Decision Diagrams (BDDs[1]) are one of the most widely used core techniques in the field of Formal Verification. They provide a compact implicit representation and manipulation formats for functions with a support varying from a few tens to a few hundreds of Boolean variables. Albeit many optimized procedures and variants of the original BDD representation format have been introduced over the years, most of the existing techniques are limited by the well-known memory explosion problem.

Recently, BDD-based tools have been challenged by approaches based on propositional satisfiability (SAT) such as Bounded Model Checking (BMC).

[1] Reduced Ordered BDDs (ROBDDs), or simply BDDs whenever no ambiguity arises.

While traditional BDD-based model checking searches for counter-examples of *unbounded* length, SAT-based model checking drops the requirement for a fix-point computation, and only targets bounded execution runs. As a consequence SAT methods search for counter-examples of *bounded* length, by working on a combinational unrolling of given length of the sequential system under check. For that reason BMC targets falsification and partial verification rather than full verification.

Some recent papers, e.g., [1, 2], have compared BDD and SAT based model checking. SAT methods have often been considered more efficient even though part of the gain is a direct consequence of comparing SAT based *bounded* against traditional BDD based *unbounded* model checking. As a consequence, in [2] the authors adapt a BDD-based model checker to bounded model checking in order to make a fair comparison between the BDD and SAT engines. Some works also address unbounded model checking using SAT solvers [3], but so far their practical impact has been much less relevant than BMC tools. In other cases mixed strategies have been proposed to cope with advantages and disadvantages of both the techniques [4].

Albeit representing a good compromise for bug hunting in large models, BMC is unfortunately a partial verification method in practice, since a complete verification would require computing the "diameter"[2] of the system, which is often impossible. Another major problem of semi-exhaustive verification is that it shows a severe degradation in usability for corner-case bugs, where the tuning effort becomes higher and recovery more difficult.

Our main goal is to find bugs, since we address the verification of large designs under development, but we are also interested in complete algorithms, able to efficiently handle both verification and falsification cases. In order to achieve the above goals, we explore possible enhancements of BDD based symbolic verification, following some of the trends proposed in the past years to face the BDD explosion problem.

We exploit *approximate traversals* [5] to simplify exact verification; we follow the *guided search* [6, 7] paradigm when combining *forward and backward* [8, 9] traversals to mutually focus and narrow their search area; we finally exploit conjunctive partitioning and cofactor based simplifications to keep BDD sizes under control. More specifically, we present what we call "inbound-path-search", i.e., a backward search approach based on partitioning and prioritized traversal. We first derive from an over-approximated breadth-first traversal an onion-ring partitioning of the state space. Each ring (or frontier) is a set of states at a given distance from the initial state. Distances are based on approximate computations, so we just use them as estimates of correct measures, in order to drive a backward search to its "target", i.e., the initial state. To do so, state sets in backward traversal are partitioned by the onion-ring frontiers, and higher priorities are given to partitions with smallest distance from the target set of states.

[2] We call "diameter" of a system its sequential depth, or the maximum shortest path connecting an initial state to any reachable state.

1.1 Related Works and Contributions

Our approach shares goals and underlying ideas with recent works in two main research paths.

On the first path, partitioning and guided searches, as *"divide-and-conquer"* methodologies, are used to attack large problems [10–13]. In the present work we do partition set of states, in order to operate smaller traversals from individual partitions, and we do adopt a priority based selection strategy. However, our partitioning strategy is driven by a traversal related heuristic, instead of mere BDD based measures (e.g., BDD size [11–13] or density [10]). We partitions state sets and we assign them priorities based on their estimated distance from the target set of states. We call our prioritized traversal "inbound-path-search", as we first explore space regions with smaller estimated distance from the target set.

On the second path, approximate, exact, forward, and backward traversals [8, 9, 14] are mixed and adopted together. Our work shares with this set of papers the general idea of focusing and guiding more accurate traversals with previous approximate ones. We follow [8] 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. Our sequence of traversals always ends up with an exact traversal, i.e., we do not have false negatives. We share with [14] the idea of using an over-approximation of the reachable state set to simplify exact verification. This is the term of comparison we choose in the experimental section, as a complete and exact model checking approach using approximate traversals. Our main contribution is to propose more sophisticated and efficient simplification strategy for BDD representation of state sets. We use individual frontier sets, i.e., the onion-rings, instead of fix-point reached states as tighter constraints for symbolic search.

The overall verification method we present is able to check invariant properties[3] on large problems made up of some hundreds of memory elements. We compare our methodology with the approximate reachability don't care model checking [14] as implemented in the VIS tool [15]. 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.

To sum up, some of the major contributions of our approach are the following:

- A new BDD based property verification strategy for Unbounded Model Checking.
- A technique for exact verification (without false negatives), 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.

[3] The extension to more general temporal logic (e.g., CTL) formulas is possible, and it has been partially implemented, but it is just mentioned in the sequel.

As a final remark it is worth pointing out that the present work extends the one presented in [16]. In that work we show that BDD-based verification can deal with large circuit and problem sizes in the *bounded* verification domain and we compare our approach with SAT-based BMC. Here we work in the *unbounded* verification domain, i.e., we propose a complete verification procedure, and we compare our results with a state-of-the-art BDD-based unbounded model checker.

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 the target state set T, i.e., the complement of P ($\mathsf{T} = \neg\mathsf{P}$), from S. Extending our approach to temporal logic formulas (e.g., CTL model check) is possible, provided that the simplification and partitioning are properly embedded within backward CTL evaluation procedures. We limit our description to invariant properties both for their practical relevance and for sake of simplicity in the explanation: support for the outermost AG or EX operators in CTL formulas is trivial, whereas nested operators would require *ad-hoc* simplifications.

```
FwdVer (TR, S, T)
  i = 0
  R0 = S
  repeat
    if ((T ∧ Ri) ≠ ∅)
      return (counterEx(TR, R))
    i = i + 1
    Ri = S∨ Img (TR,Ri−1)
  until (Ri ≠ Ri−1)
  return (PASS)
```

The procedure is based on the iterated application of the Img *function, to compute symbolic images of the set of state* R_{i-1}*. The state sets generated at each traversal iteration, i.e.,* R_i*, are often called frontier sets. Notice that the pseudo-code shows a particular case, the one we adopt for approximate forward traversals: The whole reached state set is given as input to the image procedure, whereas any state set in the interval between the newly reached states and the reached states could be used. On the fly tests for intersection with target are done at each iteration, thus avoiding full computation of reachable states whenever* T *is reached before the fix-point. A counter-example is possibly computed starting from the array* R *of frontier sets* R_i*.*

Fig. 1. Breadth-First Exact Forward Verification.

A 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. CTL model checking procedures are often implemented (as well as our exact search) as backward traversal procedures, so let us also mention here that an invariant (or an AG CTL property) can be verified by proving/disproving the mutual reachability of S and T in the backward direction. This is easily expressed by swapping the S and T sets, and changing the IMG function with the PREIMG computation in Figure 1.

Let us finally put a remark on the generalized cofactor operator. The generalized cofactor of f with respect to g is expressed as $f{\downarrow}g$. Whereas the "constrain" cofactor may introduce new variables and BDD nodes in the result, the "restrict" cofactor does not. After some experimental comparison between constrain and restrict, we selected the latter one for our implementation because of its BDD simplification properties.

3 Prioritized Forward-Backward Verification

Our approach follows [14] in its main purpose of exploiting over-approximations of reachable states as care set for backward model checking procedures. However, we address more aggressive simplification strategies: We exploit cofactoring based simplifications and decomposed representations for state sets, whereas [14] simply conjoins (Boolean AND operator) state sets with the pre-computed over-approximation. Some of our optimizations work at the level of traversal iterations, and they are shown in this section. The other ones are related to inner steps of image/pre-image computations and are discussed in the next section.

Our starting idea is to combine approximate forward and exact backward search, as done in [8] for equivalence check and in [16] for bounded model check. An initial approximate traversal is used to guide and simplify an exact backward verification task. In order to represent (and differentiate) the various state sets involved in different traversal strategies, we adopt the following notations:

- F_i (B_i): Frontier set in exact forward (backward) traversal starting from initial state set $\mathrm{F}_0 = \mathsf{S}$ ($\mathrm{B}_0 = \mathsf{T}$). We do not compute F (B) in our approach, but we use it as a term of comparison while proving the correctness of the combined forward-backward traversals.
- F_i^+: Frontier set in the over-approximate forward traversal, starting from the initial state set S ($\mathrm{F}_0 = \mathsf{S}$).
- R (R^+): Set of exact (over-approximate) forward reachable states, representing the least fix point value of F (F^+).
- FB_i: Frontier set in exact backward traversal over-approximate forward states (R^+) and/or frontiers (F_i^+). Computation of FB sets is guided and simplified using previously computed $\mathrm{R}^+$and F^+ sets.
- RB: Reached state set in exact backward traversal over approximate forward states (computed as union of FB frontiers).

Let us first introduce a model check procedure (FWDBWDMC, Figure 2) sharing similarities with [8, 14].

The FWDBWDMC procedure is able to check invariants (or $\mathrm{AG}(\neg\mathsf{T})$ CTL specifications) in the backward direction, using the fix-point over-approximation R^+ to simplify backward state sets FB. It is known that all CTL formulas can be rewritten (and evaluated) in terms of just three operators: EX, EG, and EU. Variants of FWDBWDMC can be adapted to cover all of them, by properly simplifying state sets with the R^+ care set.

```
FwdBwdMC (TR, S, T)
  F_0^+ = S
  i = 0
  /* approximate forward */
  repeat
    i++
    F_i^+ = S∨ Img^+ (TR, F_{i-1}^+)
  until (F_i^+ = F_{i-1}^+)
  R^+ = F_i^+
  RB= FB_0 = T↓R^+
  i = 0
  /* exact backward */
  while ((FB_i ∧ R^+ ∧¬ RB) ≠ 0)
    if (FB_i ∧ S ≠ 0)
      return (counterEx (TR, FB))
    i++
    Care= R^+∧ ¬ RB
    FB_i = PreImg (TR, FB_{i-1}) ↓Care
    RB= (RB∨ FB_i) ↓R^+
  return (PASS)
```

The procedure proceeds in two steps. Firstly, it computes the set of over-approximate forward frontier sets, i.e., F_i^+*. Each entry in the* F^+ *array over-estimates the frontier set of an exact traversal:*

$$F_i^+ \supseteq F_i$$

Secondly, it performs a backward traversal. The fix point value ($R^+ = F_i^+$*) is taken as care set when computing backward frontiers* FB*. More specifically, each new frontier* FB_i *is evaluated taking into account only newly reached states in the care space (*$R^+ \wedge \neg RB$*). The restrict cofactor is adopted for generalized cofactor simplifications. A counter-example is possibly evaluated.*

Fig. 2. Forward Backward for Invariant Model Checking.

Cofactor based simplifications are extensively applied though inner steps of the procedure. The correctness of the result is based on the following observations. Let us express a standard backward traversal from T by the following pre-image computation step

$$\begin{aligned} FB_i &= \textsc{PreImg}(\mathsf{TR}, FB_{i-1}) \\ RB &= RB \vee FB_i \end{aligned}$$

A first way of using care set based simplifications (as done in [14]) is

$$\begin{aligned} FB_i &= \textsc{PreImg}(\mathsf{TR}, FB_{i-1}) \wedge R^+ \wedge \neg RB \\ RB &= RB \vee FB_i \end{aligned}$$

where frontier sets FB_i are conjoined (Boolean AND) with the care set R^+(so that part of the states unreachable from S are not considered) and with $\neg RB$ (only newly reached states are taken into account). A problem with this approach arises from the fact that conjunction often means producing larger BDDs, so our solution is using the *restrict* generalized cofactor:

$$\begin{aligned} FB_i &= \textsc{PreImg}(\mathsf{TR}, FB_{i-1}) \downarrow (R^+ \wedge \neg RB) \\ RB &= (RB \vee FB_i) \downarrow R^+ \end{aligned}$$

We use both the over-approximation of (forward) reachable states (R^+) and the complement of backward reached states ($\neg RB$) as care set for frontier computation. In other words, frontiers are arbitrarily simplified over (forward) unreachable states and already reached (backward) states (included in RB). Cofactor simplification with previously reached states ($\neg$RB) is common to many BDD based traversal tools, whereas the simplification with R^+ is usually performed only for TR, not for reached states. Making it on frontier sets may add unreachable states to FB_i sets, but this does not affect the correctness of the overall process (by introducing false paths to S) since the (exact) pre-image of unreachable states ($U = \neg R$) is outside the set of reachable states:

$$U \subseteq \neg R^+ \subseteq \neg R \;\;\Rightarrow\;\; \textsc{PreImg}(\mathsf{TR}, U) \subseteq \neg R \tag{1}$$

The set of backward reachable states is also simplified by R at each new iteration introducing new values by means of disjunction (Boolean OR with the new FB_i frontier).

A second approach we introduce is more oriented to falsify the property under check (i.e., finding bugs), and it does not merely use the fix point R^+, set, but individual F^+ rings in order to simplify and guide the backward search to the target S set. The outer procedure (FwdBwdPMC) and the recursive step BwdPMCStep are shown in Figure 3.

```
FwdBwdPMC (TR,S,T)
  F_0^+ = S
  i = 0
  /* approximate forward */
  repeat
      i++
      F_i^+ = S∨ Img^+ (TR, F_{i-1}^+)
  until (F_i^+ = F_{i-1}^+)
  R^+ = F_i^+
  i = 0
  /* locate innermost intersection
    frontier - target */
  while (F_i^+ ∧ T= 0)
      i++
  if (i > size(F^+))
      return (PASS)
  /* exact backward */
  FB = BwdPMCStep (
      TR, T, S, F^+, R^+, i)
  if (FB = NULL)
      return (PASS)
  else
      return (counterEx(TR,FB))
```

```
BwdPMCStep (TR,From,T,F^+,Care,i)
  if (From ∧ T∧ Care ≠ 0)
    return (T)
  Care = Care ∧¬ From
  /* inbound-path-search */
  To = PreImg (TR, From ↓F_i^+) ↓F_{i-1}^+
  if (To ≠ 0)
    FB = BwdPMCStep (
        TR,To∧F_{i-1}^+,T,F^+,Care,i-1)
    if (FB ≠ NULL)
      FB = (From, FB)
      return (FB)
  /* global-path-search */
  To = PreImg(TR,From) ↓Care
  if (To ≠ 0)
    while (F_i^+ ∧ To = 0)
      i++
    FB = BwdPMCStep (
        TR,To,T,F^+,Care,i)
    if (FB ≠ NULL)
      FB = (From, FB)
      return (FB)
    return (NULL)
```

Fig. 3. Forward Backward Prioritized Model Check.

Approximate forward frontier sets F_i^+ are initially computed in FwdBwdPMC as in FwdBwdMC, then backward traversal is achieved recursively by BwdPMCStep receiving, as parameters, the transition relation TR, a local starting state From (a frontier set at the generic call, S at the outer one), the target set T, the forward over-approximated frontiers (F^+), a care subspace (R^+at the outer call), and the index of i the innermost F_i^+ frontier intersecting From. Since the method is particularly oriented to to seek for a backward path connecting From to S, the generic recursion tries to find it by first looking for a path through the inner frontier F_{i-1}^+:

$$\mathsf{To} = \textsc{PreImg}(\mathsf{TR}, \mathsf{From}{\downarrow}\ \mathrm{F}_i^+){\downarrow}\ \mathrm{F}_{i-1}^+$$

This step combines cofactor simplifications both on the domain and image sets. More specifically, the result set To is simplified using the inner F_{i-1}^+ frontier as care set, whereas From is simplified with F_i^+. The result is correct since no states outside F_i^+ can have a pre-image in F_{i-1}^+. In case the computed pre-image To is not void, i.e., a step to the inner frontier has been done, a recursive call is activated to complete the path to S: The procedure receives as initial set $\mathsf{To} \wedge \mathrm{F}_{i-1}^+$, where the conjunction is kept indicated (no product BDD is computed), so that the second term will disappear when recursively computing the pre-image to the F_{i-2}^+ frontier and so on so forth.

As far as pre-images succeed reaching inner frontiers, the approach shows its best guiding and simplification power, since the search is driven by approximate frontiers into sub-spaces where the chance to attain the goal is higher. Of course the chance relies in the goodness of the (forward) approximation process, so the search might fail, either for errors in the approximations, or simply because no paths to S exist (property is not falsifiable). This is the case where backward search is completed by a local wider exploration from the From set to the whole $\textsc{Care}$, which is equal to R^+, with the exclusion of all From sets up in the recursion tree. This attempt is done both to explore the complete search space when no *inbound* path is found, so that starting states for new searches are generated, or the absence of a path to S is finally proved. In order to avoid an excessive amount of partitioning, with consequent exponential degradation of time performance, we have two heuristic controls on the activation of the *inbound* path search: the BDD size threshold on From (no inbound search is done if From is under the threshold) and a maximum amount of partitionings up in the recursion tree. The above controls are common to several techniques adopting partitioning with BDDs, and they are not shown in Figure 3 for sake of simplicity.

Recursions stop whenever the target set T is intersected by From, or a dead end is attained because no success was locally found by both inbound and global attempts.

The correctness of the procedure is guaranteed by the following theorem.

Theorem 1. *Let* F^+ *be an array of* k *over-approximated forward frontiers such that* $\mathrm{F}_0^+ = \mathsf{S}$, $\forall_{0<i<k}\mathrm{F}_i^+ \supseteq \mathrm{F}_i$, *and* $\mathrm{F}_{k-1}^+ = \mathrm{R}^+ \supseteq \mathrm{R}$. *Let* T *be the complement of an invariant under check. Then the* $\textsc{FwdBwdPMC}$ *is correct, i.e., it finds a path from* T *back to* S *if and only such a path exists.*

Sketch of Proof The proof is achieved in two steps:

- We first prove that no false negatives are generated by the procedure, i.e., that any counter-example generated is correct. This is guaranteed by the fact that partitioning and prioritized visit does not introduce states unreachable from S. The only unreachable states considered are those generated by cofactor simplifications, that do not generate backward paths to S (see Equation (1)).
- We then show that the procedure is complete, i.e., it fully explores a subspace able to prove or falsify the property. We do this by showing that the "global-path-search" section of BwdPMCStep is just a recursive version of FwdBwdMC (and it explores the same state sets) in case the "inbound-path-search" fails.

A key issue for performance in the above procedures 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[4] support: $\mathrm{F}_i^+ = \prod_{j \in Groups} f_i^j$ (see Section 4). As in transition relation clustering, we partially compute products as far as the generated products is under a chosen threshold.
- Accuracy of the over-approximate forward traversal is not as important as in approximate model check [9], 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 [18].

4 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. We will describe approximate image, exact and approximate pre-image, under a general attempt to control BDD sizes by means of conjunctive partitioning and clustering[5], combined with generalized cofactor simplifications.

[4] Overlapping projections are also possible, as adopted in [8, 17], but we presently limit our implementation to approximate images with non-overlapping components.

[5] We denote with clustering a partial computation of a product (AND), controlled by a BDD size threshold. The technique is derived from partitioned transition relation

Approximate image is computed as

$$\mathrm{F}_i^+ = \textsc{Img}^+(\mathsf{TR}, \mathrm{F}_{i-1}^+) = \prod_{j \in Groups} \textsc{Img}(\mathsf{TR}_j, \mathrm{F}_{i-1}^+)$$

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). The partial images of each 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 (Carein Figure 2 and Figure 3) is given as a parameter to the PreImgWithCare function, in order to be used as a care set for inner operations:

$$\begin{aligned} \mathrm{FB}_i &= \textsc{PreImg}(\mathsf{TR}, \mathrm{FB}_{i-1}) \downarrow \textsc{Care} \\ &= \textsc{PreImgWithCare}(\mathsf{TR}, \mathrm{FB}_{i-1}, \textsc{Care}) \downarrow \textsc{Care} \end{aligned}$$

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 $\mathrm{P}_j = \exists_{x_j}(\mathrm{P}_{j-1} \wedge \mathsf{TR}_j)$) is optimized as follows

$$\begin{aligned} \mathrm{P}_j &= \exists_{x_j}(\mathrm{P}_{j-1} \wedge tr_j) \downarrow \textsc{Care} \\ \mathrm{P}_j &= \exists_x \mathrm{P}_j \wedge (\mathrm{P}_j \downarrow \exists_x \mathrm{P}_j) \end{aligned}$$

Each partial product is first cofactored with the care set Care, then the basic decomposition step of [18] 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.

5 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.

We describe an experimental comparison between this tool and VIS [15] using the technique described in [14] as implemented with the command "model_check -D 3". Our experience with VIS shows that within the proposed set of experiments this choice is more efficient than other special purpose techniques, e.g., approximate model checking (command "approximate_model_check"), and forward invariant checking (command "check_invar") (both unable to complete most of the experiments). The experiments are performed on a 1.7 GHz Pentium IV Workstation with a 1 GByte main memory, running RedHat Linux.

We present data on two sets of circuits:

---

manipulation: Whenever the size of an intermediate product is too high, the product is aborted, a cluster is put aside and product computation resumes.

– ISCAS'89 and ISCAS'89-addendum benchmarks (upper part of Table 1). In this case the verified properties are automatically generated as described in [16][6].
– Models taken from [6] (lower part of Table 1). In this case the verified properties are available with the original descriptions.

Each line in Table 1 describes one or more checks on a given model. Models are sorted by number of state variables (column # SV). Column # reports the number of checked properties. Whenever # is larger than one, we report overall results for the verified set of properties. Properties are either proved correct, and denoted by *Pass*, or they are falsified and labeled by *Fail*. Depth indicates the maximum sequential depth explored (i.e., the length of the counter-example or the amount of breadth-first backward iterations done to prove correctness) over the set of checked properties.

Overall, the circuits presented have different sizes, some of them outside the range of problems manageable by state-of-the-art BDD based verifiers. Circuit philo is a synchronous version of the Philosopher problem, where the asynchronous behavior is modeled by a scheduler enabling just one philosopher at the time. We check safety properties in all the cases. The Cone-Of-Influence reduction is always applied before starting the verification process.

Model	# SV	Property			VIS		FBV	
		#	Pass/Fail	Depth	Mem. [MByte]	Time [sec]	Mem. [MByte]	Time [sec]
s3330	132	5	Fail	5	45	640	20	17
s3384	183	3	Fail	11	203	*ovf*	65	2993
s9234	211	5	Fail	241	23	198	30	427
s15850.1	534	3	Fail	76	143	1230	118	700
s13207.1	638	5	Fail	430	120	1748	60	1824
s38584.1	1426	1	Fail	11	342	*ovf*	180	456
s35932	1728	1	Fail	32	315	*ovf*	60	831
vsaR	66	2	Fail	20	21	29	17	21
am2901	68	1	Fail	17	36	116	28	81
FIFOs	142	1	Pass	16	121	2618	35	823
philo_{20}	40	1	Pass	35	16	4	20	13
philo_{60}	120	1	Pass	115	251	*ovf*	29	78
philo_{100}	200	1	Pass	195	383	*ovf*	159	6220
Palu_{16}	317	1	Fail	3	37	196	23	110
		1	Pass	3	38	271	23	82

Table 1. Experimental comparison between FBV and the Approximate Reachability Don't Cares strategy of VIS. ovf means overflow on time (time limit = 10800 seconds). In this case Mem. indicates maximum memory usage before aborting verification.

[6] Notice however that [16] refers to bounded verification whereas we present here only data on unbounded verification.

The FwdBwdPMC procedure is used for the s3330, s3384, and s35932, FwdBwdMC in all other cases. Data shows how our technique is more efficient in terms of memory usage (since our optimizations primarily target BDD size reduction). As far as execution time is concerned, in some cases the two tools are comparable (usually on experiments where BDDs could be enough easily deal with the VIS tool), whereas in other cases our optimizations are the key to complete the verification task. On the one hand, minor differences in "easier" experiments are related to different settings and partitions used in approximate reachability. On the other hand, performance in "difficult" experiments is largely dominated by backward traversals, where cofactor based simplifications provides much smaller BDDs in FBV than the corresponding AND based simplifications in VIS.

6 Conclusions and Future Works

BDD-based symbolic manipulation has been one of the most widely used core technologies in the synthesis and verification domain, over the last decade. Nevertheless, existing algorithms are still limited by memory resources in practice.

Following some of the most promising techniques proposed over the last a few years, we present in this paper an approach to face very large *Unbounded Model Checking* problems. 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 a state-of-the-art BDD based model checker. Our experience leads to the conclusion that we are able to deal with problems outside the present scope of other BDD-based tools.

Among the possible future works we need some effort on heuristics, to make our approach more self-tuning, and to obtain a common framework for experiments and comparisons with different tools and sets of models.

7 Acknowledgment

The authors would like to thank Fabio Somenzi for the source descriptions of some circuits used in the experiments.

References

1. A. Biere, A. Cimatti, E. M. Clarke, M. Fujita, and Y. Zhu. Symbolic Model Checking using SAT procedures instead of BDDs. In *Proc. 36th Design Automat. Conf.*, pages 317–320, New Orleans, Louisiana, June 1999.
2. 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.

3. P. F. Williams, A. Biere, E. M. Clarke, and A. Gupta. Combining Decision Diagrams and SAT Procedures for Efficient Symbolic Model Checking. In E. Allen Emerson and A. Prasad Sistla, editors, *Proc. Computer Aided Verification*, volume 2102 of *LNCS*, pages 124–138, Chicago, Illinois, July 2000. Springer-Verlag.
4. 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.
5. 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.
6. 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.
7. M. K. Ganai, A. Aziz, and A. Kuehlmann. Enhancing Simulation with BDDs and ATPG. In *Proc. 36th Design Automat. Conf.*, pages 385–390, New Orleans, LA, November 1999.
8. 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.
9. 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.
10. K. Ravi and F. Somenzi. High–Density Reachability Analysis. In *Proc. Int'l Conf. on Computer-Aided Design*, pages 154–158, San Jose, California, November 1995.
11. G. Cabodi, P. Camurati, and S. Quer. Improving the Efficiency of BDD–based Operators by means of Partitioning. *IEEE Transactions on CAD*, 18(5):545–556, May 1999.
12. G. Cabodi, P. Camurati, and S. Quer. Improving Symbolic Reachability Analisys by means of Activity Profiles. *IEEE Transactions on CAD*, 19(9):1065–1075, September 2000.
13. R. Fraer, G. Kamhi, B. Ziv, M. Y. Vardi, and L. Fix. Prioritized Traversal: Efficient Reachability Analysis for Verification and Falsification. In E. Allen Emerson and A. Prasad Sistla, editors, *Proc. Computer Aided Verification*, volume 1855 of *LNCS*, pages 389–402, Chicago, Illinois, July 2000. Springer-Verlag.
14. 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.
15. R. K. Brayton et al. VIS. In Mandayam Srivas and Albert Camilleri, editors, *Proc. Formal Methods in Computer-Aided Design*, volume 1166 of *LNCS*, pages 248–256, Palo Alto, California, November 1996. Springer-Verlag.
16. 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.
17. 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.
18. 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.