Available online at www.sciencedirect.com
SCIENCE DIRECT®
Electronic Notes in Theoretical Computer Science
ELSEVIER Electronic Notes in Theoretical Computer Science 119 (2005) 33–49
www.elsevier.com/locate/entcs

Exploiting Target Enlargement and Dynamic Abstraction within Mixed BDD and SAT Invariant Checking

Gabriel P. Bischoff, Karl S. Brace

Massachusetts Microprocessor Design Center, Intel Architecture's Group, Shrewsbury, MA

G. Cabodi, S. Nocco, S. Quer[1]

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

Abstract

In this paper, we propose a methodology to make Binary Decision Diagrams (BDDs) and Boolean Satisfiability (SAT) Solvers cooperate. The underlying idea is simple: We start a verification task with BDDs, we go on with them as long as the problem remains of manageable size, then we switch to SAT, without losing the work done on the BDD domain.
We propose *target enlargement* as an attempt to bring some of the advantages of state set manipulation from BDDs to SAT-based verification. We first, "enlarge" initial and target state sets of a given verification problem by affordable BDD manipulations. This step is carried on with a few breadth-first steps of traversal, or with what we call *high-density dynamic abstraction*, i.e., a new technique to collect under-approximate reachable state sets. Then, we perform SAT-based verification with the newly computed "enlarged" sets.
We experimentally test our methodology within an industrial environment, the Intel *BOolean VErifier* BOVE. Preliminary results on standard benchmarks (the ISCAS'89, ISCAS'89–addendum, and VIS suites), and industrial ones (the IBM Formal Verification Benchmark Library) are provided. Results show interesting improvements over state-of-the-art techniques: We could decrease CPU time up to a 5x factor, when performing verification with the same depth, or we could increase the verification depth up to 30%, when performing verification within the same time limit.

Keywords: Invariant Checking, Satisfiability Solver (SAT), Binary Decision Diagrams (BDDs), Target Enlargement.

[1] Email: stefano.quer@polito.it

doi:10.1016/j.entcs.2004.06.061

1 Introduction

Current design methods are so complex that simulation is no longer adequate. In current design frameworks, hundreds to thousands of bugs must be found and removed during the initial phases of the design. While finding bugs remains an important goal, it is also essential to be able to prove their absence, i.e., formally verifying the correctness, before starting the production process.

Binary Decision Diagrams [7] (BDDs) have been widely used to formally verify correctness because of their ability to exhaustively analyze a problem. Nevertheless, BDDs have never been able to deal with the largest models and problem instances. Boolean Satisfiability [25] (SAT) Solvers, on the other hand, have been gaining ground especially for their debugging capability adopting Bounded Model Checking [3,11] (BMC). Unbounded inductive verification [23] guarantees full verification, but it is more complex than BMC and typically converges at higher sequential depths than BDDs.

In this work, we explore a new way to make BDD-based and SAT-based tools cooperate. Our target is to trade-off space and time, i.e., to improve "time efficiency" of SAT-based verification with the help of affordable, i.e., "space manageable", BDD-based operations. First of all, we "enlarge" initial and target state sets of the given verification problem by means of BDD manipulations. Then we perform SAT-based verification exploiting the newly computed "enlarged" set. In this way, we partition a verification task between two different tools. Initial BDD operations are useful for their breadth-first state space visit, and for their ability to represent state sets. Whenever we switch to SAT, BDD-based state sets provide a tighter space pruning and an enforced convergence within the SAT engine. Our main contribution is to study the impact of enlarged sets over SAT-based (both bounded and unbounded) model checking. We show that our approach is a task partitioning strategy, a way to restrict the SAT search, and to tighten the convergence of the unbounded algorithm. Another contribution of our work is to propose a BDD sub-setting procedure mixing the high density paradigm with an effort to reduce the number of variables (abstraction) in the support of a BDD. We call this technique "dynamic abstraction".

Preliminary results are reported on standard benchmarks (the ISCAS'89, ISCAS'89–addendum, and VIS suites), and industrial ones (the IBM Formal Verification Benchmark Library [18]). They show interesting improvements in terms of both efficiency and power, and demonstrate how target enlargement is able to boost BMC and induction toward larger verification tasks.

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 work. Section 4 introduces the outline of our approach justifying and describing algorithms for target enlargements within SAT. Section 5 describes our dynamic under-approximate reachability analysis to compute target enlargements. Section 6 presents our top-level algorithm with some more implementation-level details. Section 7 presents our experimental evidence. Finally, Section 8 concludes the paper with a brief summary, and some possible future works.

2 Background

2.1 Model and Notation

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

An invariant property [2] P is checked by attempting to prove (or disprove) the reachability of its complement T (target state set, $\mathsf{T} = \neg\mathsf{P}$) from S. For sake of simplicity, we will always refer to the above kind of properties, even though our techniques can be applied to a broader set, the LTL properties supported by SAT-based verification algorithms.

2.2 SAT-Based Bounded Model Checking and Inductive Verification

In order to decide if a Boolean formula f is satisfied, most solvers adopt variants of the basic Davis-Putnam recursive algorithm. SAT solvers generally operate on problems for which the propositional formula 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*.

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. Complete verification is usually achieved by BMC with longer and longer bounds or by inductive techniques [23,19]. In short, in inductive verification, a sequence of BMC steps with increasing bound is complemented with SAT-based induction checks. In [23] the authors call a *path* a sequence of states through TR:

$$path(s_{[0..n]}) = \bigwedge_{0 \le i < n} \mathsf{TR}(s_i, s_{i+1}) \tag{1}$$

[2] Or AG CTL property.

and they define *loopFree* a path that visits a state at most once:

$$loopFree(s_{[0..n]}) = path(s_{[0..n]}) \wedge \bigwedge_{0 \leq i < j \leq n} (s_i \neq s_j) \tag{2}$$

The property is proved correct, i.e., S and T are mutually unreachable, if the following conditions hold:

- *Base case*: No bounded path of length less than k connects S to T.
- *Inductive proof*: No simple path of length k exists such that its first state is initial and no other state is initial, or such that its final state is in T and no other state is in T.

The authors of [23] demonstrate that the resulting method is sound and complete. Intuitively, the two conditions introduced as inductive proof correspond to the fact that, starting from the initial state, the full reachable state space has been visited, or that, after the property has been proved correct for $k-1$ time steps, it cannot become false (i.e., starting from the target state, we cannot reach the initial state). For this reason, in the sequel we call the two inductive checks as "forward" or "backward" induction.

2.3 BDD-Based Model Checking

Standard BDD-based *forward model checking* is presented in Figure 1. The procedure is based on the iterated application of the post-image (Img) function, to compute symbolic post-images of the set of state R_{i-1}. R_i are the state sets generated at each traversal iteration. Notice that in the pseudo-code 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 *whole* reached states set could be used. On-the-fly tests for intersection with the 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, by the function Trace, starting from S, T and the array R of frontier sets R_i identifying all admissible paths.

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 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. In the sequel we will call FR and BR the forward and backward reachable state set respectively.

Approximate Traversals [10,13] are a popular way to extend the applicability of reachability analysis to larger circuits. The approach is based on

```
FwdMC (TR, S, T)
    R_0 = S
    i = 0
    do
        if ((R_i ∧ T) ≠ ∅)
            return (Trace (TR, S, R, T))
        i = i + 1
        R_i = S ∨ Img (TR, R_{i-1})
    while (R_i ≠ R_{i-1})
    return (Pass)
```

Fig. 1. Forward Model Checking.

the *approximate image* (Img^+) operator, returning over-estimations of exact images:

$$\text{Img}^+(\mathsf{TR}, \mathsf{FR_i}) \supseteq \text{Img}(\mathsf{TR}, \mathsf{FR_i}) \tag{3}$$

Notice that, in the sequel, we will indicate with FR^+ and BR^+ (FR^- and BR^-) the over (under) estimations of the forward and backward reachable state set respectively.

3 Related Works and Comparison Remarks

With the advent of SAT-based BMC tools a lot of researchers compared SAT-based methods with more traditional BDD-based ones. As different researchers agree that the two approaches are essentially complementary, a lot of recent works concentrate on dovetailing the two approaches in a loose or strict fashion. In this section, we review, among these works, the ones more strictly related to our approach.

Gupta et al. [16,17] perform BDD-based reachability analysis by using a SAT procedure within symbolic image computation. They call their approach *BDDs at SAT Leaves*. More specifically, they use BDDs to represent state sets and a CNF formula to represent the transition relation. Symbolic image of a state set is computed by exhaustive SAT search of all solutions within the space of primary input, present and next state variables. However, rather than using SAT to enumerate each solution all the way down to a leaf, image switches to BDD-based computations at certain intermediate points within the SAT decision tree. This is done as a trade off between space complexity of BDDs and time complexity of full SAT enumeration. In a sense, this approach can be regarded as SAT providing a disjunctive decomposition for image computation into many sub-problems, each of which is handled symbolically using BDDs.

As far as SAT performance is improved by BDD learning, Cabodi et al. [9] propose BDD pre-processing by means of over-approximate reachability. The authors show how to translate over-approximate state sets from BDDs to CNF

clauses, to be used by a SAT solver as an extra learning, which is redundant, yet able to improve overall performance in BMC.

Gupta et al. [14] propose an approach sharing similarities with the previous one. They start from the CNF representation of the problem, and they build BDDs of selected "structural" points to learn useful information, in order to improve the learning ability of the solver.

Another work by Gupta et al. [15] may be considered as a variation of [9] for the BMC case, as they also use over-approximate reachability analysis to constrain the BMC search. A novel idea in their work is the extension to induction-based unbounded verification, where the authors exploit over-approximate information as an additional (non redundant) constraint.

Our work shares with the previous ones a few guiding ideas. We use both BDD and SAT-tools to cope with their contrasting limits but we partition the work-load in a new way. Moreover, in our approach, BDDs are not used to perform a learning activity virtually useful only to prune the subsequent search, but are adopted to partially truly perform the verification task. As in other approaches, we rely on a loose coupling between the BDD and the SAT tool, but we strongly interact with the model checker.

4 Methodology's Outline

In this section we overview our methodology. The approach we propose can be viewed:

- As a way to partition a verification task between a BDD and a SAT engine. We perform a preliminary effort with BDDs, we conclude the task through a SAT solver, working on the solution space left uncovered by BDD preprocessing. In other words, counterexamples are computed (or refuted) partially within the BDD domain and partially within the SAT one.
- As an optimization of SAT-based model checking, where the results of BDD preprocessing are used not merely to reduce the search area, but to further optimize SAT search in its target sub-space. More in detail, state sets generated during the BDD phase are used as a stronger constraint for the relative SAT problem, and to enforce the convergence of SAT-based unbounded model checking.

In the rest of this section we first show how BDD-based target enlargement can be considered in terms of task partitioning between the BDD and SAT tools. Then we introduce some specific optimizations for SAT searches. BDD preprocessing for target enlargement and the overall verification algorithm will be described in the following sections.

4.1 Target Enlargement as a Task Partitioning Strategy

Target Enlargement [24,2] is a known paradigm in hardware verification. It is an effort to better coordinate and balance a verification workload between two tools or two different search procedures within the same tool.

Whenever a verification task looks for a path to a given target state, one might "enlarge" the target by computing a set of states reachable in the opposite direction from the target. Roughly speaking, the target is now replaced by a wider area, and the chance to reach it (or to prove it is unreachable) is now higher. Analogous considerations hold for the initial set of states.

We apply it by means of under-approximate BDD-based reachability starting either/both from S or/and T. Our idea is to first compute BDD-based reachable state sets, so that they can be used as new targets for easier SAT processing.

Let us work on a SAT-based model checking problem where the goal is to prove the mutual reachability between S and T. We can generate a related SAT problem with new enlarged initial and/or target state sets S^e and T^e, such that

$$\begin{aligned} \mathsf{S}^e &= \mathsf{FR}^-(\mathsf{S}) \subseteq \mathsf{FR}(\mathsf{S}) \\ \mathsf{T}^e &= \mathsf{BR}^-(\mathsf{T}) \subseteq \mathsf{BR}(\mathsf{T}) \end{aligned} \tag{4}$$

The new *enlarged* start (target) state set is a set of states for which a path from S (to T) exists. BDD-based computation of S^e and T^e will be discussed in the next section. Let us just consider here the straightforward case of under-approximation by bounded exact reachability, i.e., a few traversal iterations, with $\mathsf{S}^e = \mathsf{FR}_{d_F}$ and $\mathsf{T}^e = \mathsf{BR}_{d_B}$. In this case d_F and d_B are the depths of the bounded traversals in the forward and backward directions respectively. The enlarged set S^e replaces S and the first d_F time frames in the combinational unrolling. Dually for T^e. More formally:

$$\mathsf{S}^e = \exists_{s_0..s_{d_F-1}}(\mathsf{S} \bigwedge_{i=0..d_F-1} \mathsf{TR}(s_i, s_{i+1})) \tag{5}$$

- Disproving mutual reachability for S^e and T^e is equivalent to disproving it for S and T.
- Proving mutual reachability for S^e and T^e, i.e., finding a counterexample connecting them, is equivalent to proving mutual reachability for S and T. The generated counterexample is partial, and we need to complete it with a prefix and a suffix, in order to reach the original S and T states.

Intuitively, we may expect a performance gain from the above task partitioning if the preprocessing work done with BDDs is manageable and, more

importantly, it is able to decrease the overall memory/time complexity. As BDDs are able to start a mutual reachability task from S and T, the sequential depth of SAT exploration (e.g., the induction depth) can be lower with target enlargement than with the original problem. Moreover, we may expect to handle sequentially deeper problems, given the expected capacity of the SAT tool. Obviously, the above expectations strongly depend on how efficiently the enlarged state sets replace the equivalent set of time frames in the combinational unrolling. In general we may expect an advantage related to the reduction of variables and clauses in the final problem.

Let us examine the substitution on the particular case of BMC. We choose it for sake of simplicity, and we show that a given BMC problem of bound k can be solved, with target enlargement, as a BMC problem with shorter bound. Similar formulations can be done with unbounded model checking. The following proposition holds.

Proposition 4.1 *Let $BMC_k(\mathsf{TR}, \mathsf{S}, \mathsf{T})$ be a BMC problem of bound k, over a given transition relation TR, an initial S and a target T state sets. Let S^e and T^e be "enlarged" initial and target sets as defined above. Let us define $\widehat{d_F}$ ($\widehat{d_B}$) the maximum value of l_F (l_B) such that $\mathsf{FR}_{l_F} \subseteq \mathsf{S}^e$ ($\mathsf{BR}_{l_B} \subseteq \mathsf{T}^e$). Then the original BMC_k problem can be solved as a new BMC problem $BMC_h(\mathsf{TR}, \mathsf{S}^e, \mathsf{T}^e)$ with possibly shorter bound, i.e., such that $h = k - (\widehat{d_F} + \widehat{d_B})$.*

4.2 Restricting SAT Search with Enlarged Target

We now show that target enlargement can restrict the search space of the SAT solver, with additional benefits, in terms of performance, besides simple bound reduction. More specifically, let us concentrate on a particular form of BMC, the so called *exact − assume* variant of BMC, and on the inductive steps of unbounded SAT as proposed in [23].

Exact-assume BMC can be expressed as:

$$BMC_k(\mathsf{TR}, \mathsf{S}, \mathsf{T}) = \mathsf{S}(s_0) \wedge path(s_{[0..k]}) \wedge \bigwedge_{0 \le i < k} (\neg \mathsf{T}(s_i)) \wedge \mathsf{T}(s_k) \tag{6}$$

Inductive steps in unbounded model checking [23] can be expressed as:

$$\textsc{FwdInductiveStep}_k(\mathsf{TR}, \mathsf{S}) = loopFree(s_{[0..k]}) \wedge \mathsf{S}(s_0) \wedge \bigwedge_{0 < i \le k} (\neg \mathsf{S}(s_i)) \tag{7}$$

$$\textsc{BwdInductiveStep}_k(\mathsf{TR}, \mathsf{T}) = loopFree(s_{[0..k]}) \wedge \mathsf{T}(s_k) \wedge \bigwedge_{0 \le i < k} (\neg \mathsf{T}(s_i)) \tag{8}$$

In all the above formulas, the complement of S and/or T are used to constrain the state at the i-th time frame. The effect of our approach is not only to constrain the SAT search space, but also to tighten the termination conditions of

unbounded model checking. Our argument here is that larger start and target state sets are able to provide tighter constraints, enhancing SAT performance and enforcing termination of unbounded checks.

In the BMC case with bound k, all time frames but the last one are constrained by $\neg \mathsf{T}^e$. In the induction case with bound k, all time frames but the first one are constrained by $\neg \mathsf{S}^e$, whereas all time frames but the last one are constrained by $\neg \mathsf{T}^e$. This is clearly a stronger search space pruning than in the original BMC problem. Intuitively all states in *all* BDD-computed enlarged state sets are forbidden in *all* state paths considered by the SAT solver. In another way, a state is not considered within a SAT state path if it belongs to any BDD-based state path prefix or suffix, which is not necessarily a possible prefix or suffix of the current path in the SAT search. This search state pruning is not achievable by SAT reasoning on the original $BMC_k(\mathsf{TR}, \mathsf{S}, \mathsf{T})$ problem, due to the "linear time" reasoning employed. Moreover, the termination of unbounded induction-based verification is enforced by the smaller number of states available for loop-free state paths starting from S (leading to T). The above fact directly stems from the fact that every loop free path satisfying the FwdInductiveStep$_k$ obtained by using the enlarged state sets is also included or equal to a loop free path satisfying the same problem generated by using the original state sets, whereas the reverse is not true.

As a final remark, notice that the complement of our target state set T (i.e., $\neg \mathsf{T}^e$) can be used as constraint for the i-th time frame (with $i < k$) in FwdInductiveStep$_k$ (TR, S^e). Moreover, S^e can by used dually in BwdInductiveStep$_k$ (TR, T^e). This is not far from using an over-approximation of forward (backward) reachable states as constraint for the backward (forward) induction, as described in [15]. Nevertheless, here $\neg \mathsf{T}^e$ ($\neg \mathsf{S}^e$) is not an over-approximation of the forward (backward) reachable state set. On the contrary, it is an over-approximation of the search area where we look for states in loop-free paths. Similar considerations also hold for the BMC case.

5 Under-approximate Reachability and Dynamic Abstraction

Over-approximate reachability has been proposed in several works as an abstraction technique, with the aim of improving capacity and scalability. On the contrary, under-approximate techniques have received less attention in formal verification in recent years. Under-approximation was specifically addressed in the BDD sub-setting work by Somenzi et al. [21]. Many works then followed the partitioning and guided search paradigms [8,22,12] where a difficult reachability task could be faced by case splitting or focusing on a

search sub-space. Within this framework, BDD sub-setting was just one of the possible ways to produce a cut on a complex state set.

The techniques we are proposing here essentially aim at using under-approximation to gather "enlarged" state sets $\mathsf{S}^e$and T^e, such that:

- They are included in the exact forward and backward reachable state sets:

$$\begin{aligned} \mathsf{S} \subseteq \mathsf{S}^e \subseteq \mathsf{FR}(\mathsf{S}) \\ \mathsf{T} \subseteq \mathsf{T}^e \subseteq \mathsf{BR}(\mathsf{T}) \end{aligned} \tag{9}$$

- Their characteristic functions can be computed and represented in terms of BDDs at a manageable cost.

Among the various available choices, we present here:

- *Bounded traversals.*
- A new form of high-density BDD sub-setting that we call *high-density dynamic abstraction.*

The former strategy is a very straightforward option, particularly suited for symbolic traversals characterized by affordable initial iterations. With the latter one we tackle BDD explosion more aggressively. As BDD blow up is often related to the number of support variables, i.e., the variables BDDs depend on, we aim at reducing the support of state sets, with a possibly minimal impact on the number of represented state sets.

5.1 Under-Approximation by Bounded Traversal

We call bounded traversal a simple variant of a standard forward and/or backward traversal, where BDD-based breadth-first reachability stops before reaching a fix-point. Threshold based BDD control is a natural exit condition, but other options are possible as well, like cardinality of the reached state set, number of support variables, and pre-determined number of traversal iterations.

In all cases, the traversal ends up computing FR_{d_F} and BR_{d_B}, and the traversal depths d_F and d_B are the exact parameter required to decrease the length of SAT checks.

In general, forward reachable state sets depend on all variables since it is the first iteration, whereas backward state sets follow the so called Cone-Of-Influence of the property. In practice the number of support variables (and the BDD size) of T^e grows for growing values d_B. The good choice is a trade-off between BDD size and number of state variable we are able to further constrain in successive SAT processing.

5.2 Under-Approximation by High-Density Dynamic Abstraction

Our target in this section, is to adopt some optimization to gather as more states as possible in the enlarged sets before switching to SAT. As opposed to partitioning strategies, where a complex task is split in subtasks, sub-setting here means that, given a large BDD, we select a "relevant" subset, that we use for further processing steps. The pruned subset is either temporarily or permanently removed from the traversal process, since it is deemed completely or almost completely irrelevant.

We work within the inner loop of a BDD-based traversal. Whenever a state set violates a predefined threshold (BDD size and/or support size), we dynamically operate sub-setting. High density as introduced in [21] aimed at clipping a BDD so that a minimal number of minterms (i.e., states) was removed from it. Density was defined as the ratio number between the minterms in the subset and in the original BDD. Pruning was done recursively, so no particular care was taken at reducing the amount of variables in the support.

The sub-setting technique we propose is a compromise between support reduction and high-density. It can be used either for BDD super-setting or sub-setting, as the basic step is variable quantification. If *sub-setting* is our goal, we *universally quantify* a variable so that a minimal number of minterms are removed from the original BDD. Super-setting would be achieved in a dual way, by adopting existential quantification and minimizing the newly introduced minterms.

Let us suppose a function $f(X)$ is represented as a BDD, and let us use the notation $|f|$ to indicate its minterm count (i.e., the number of domain points where $f = 1$). We compute the subset:

$$f^-_\sigma(X - \sigma) = \forall_\sigma f(X) = f(x)_{\sigma=0} \wedge f(x)_{\sigma=1} \tag{10}$$

The $\sigma \in X$ variable is selected so that $|f^-|/|f|$ is maximal. We loop through such sub-setting steps until we get a sufficient reduction either in terms of BDD size or support variables. The variable selection criteria is clearly greedy, as a sequence of optimal variable selection is not guaranteed to produce the best overall result. We should also notice that, although variable quantification does not guarantee BDD size reduction, it often does so. To take into account possible BDD size increase, our variable selection is a weighted compromise between size and density.

We call our sub-setting *dynamic abstraction* as opposed to over-approximate reachability and abstraction-refinement techniques, where variable abstraction schemes are generally decided *statically* as a pre-processing and/or post-processing step of entire traversals. Here we do variable selection and universal abstraction "on-the-fly", within inner steps of a traversal procedure. The main

motivation for selecting this kind of sub-setting is that we control BDD explosion (as in other abstraction schemes) by reducing the number of support variables. We still resort to a high density heuristic, as we want to maximize the amount of state space "covered" by the enlarged set.

Due to the above mentioned dynamic scheme, time and memory efficiency of variable selection is a key issue, as the introduced overhead should by negligible within the overall traversal process. A naive approach consists of first computing variable abstraction for all variables, then selecting the one with best weighted size-density benefit. This can be very expensive, as it means to compute a new (possibly larger) BDD for each variable.

An alternative and much more effective approach is density estimation without computing the abstraction. Given $f(X)$ and a candidate variable $x_i \in X$, we can compute $|f^-|$ through a variant of the standard minterm count routine, that visits the subset of BDD nodes in $f(x)$ reachable both under the $x_i = 1$ and $x_i = 0$ constrains. A double "linear" visit can achieve this task. In the first visit we mark the $f(x)$ nodes reachable with $x_i = 0$. The second visit achieves the actual minterm count, by working on the previously marked nodes, under the opposite constraint $x_i = 1$.

As a result, a best density abstraction variable can be computed in linear time without generating any new BDD. This does not take into account the BDD size of the result. So we add an extra step where we actually compute the abstraction for the topmost variables, after ordering them by estimated minterm density. Possible BDD blow up is avoided by a size threshold controlling universal abstraction as a try-and-abort operator.

6 Overall Verification Algorithms

The techniques described in the previous sections are integrated within a mixed BDD/SAT verification framework using a mix of bounded (Section 5.1) and high-density dynamic abstraction (Section 5.2) traversal. The pseudo-code in Figure 2 shows the unbounded verification procedure based on induction. BMC is easily derived as a simplified version of the shown pseudo-code, where inductive checks are removed, and the main SAT verification loop stops at a predefined bound.

Initial BDD-based traversals compute enlarged sets S^e and T^e. SAT verification iterates over BMC and inductive steps until either a counterexample is found by BMC or an inductive step is unsat. If a counterexample is found, it is extended by a prefix and a suffix, computed within $\mathsf{S}^e$and T^e. If verification passes, the procedure returns the depth of termination.

Figure 2 basically shows that BDD traversals are done as a pre-processing

```
InductiveMcWithTE (TR, S, T)
    Set under-approx thresholds and bounds
    S^e= FwdUnderApproxTrav (TR, S)
    T^e= BwdUnderApproxTrav (TR, T)
    h = 0
    while (true)
        (result, cex) = BMC_h(TR, S^e, T^e)
        if (result = Sat)
            prefix = ComputeCex (TR, S, S^e, cex)
            suffix = ComputeCex (TR, T, T^e, cex)
            return (prefix, cex, suffix)
        result = FwdInductiveStep_{h+1} (TR, S^e)
        if (result = UnSat)
            return (Pass)
        result = BwdInductiveStep_{h+1} (TR, T^e)
        if (result = unsat)
            return (Pass)
        h = h+1
```

Fig. 2. Inductive verification with target enlargement.

step outside the loop over increasing bound h. This means that the overhead due to BDD manipulation decreases as far as verification requires longer and longer depths.

The procedure hides some details, as BDD to CNF conversion, required for generate the BMC and inductive problems with the enlarged initial and target sets. In principle, any BDD can be converted to CNF with an amount of variables and clauses linear in the BDD size. We adopt a more sophisticated approach, as described in [9], with heuristics trading off the number of clauses and of new variables required by the BDD to CNF conversion.

7 Experimental Results

This section describes an experimental comparison in terms of BMC and inductive verification with and without the technique described in this paper. The presented methodology is implemented within the industrial Intel tool *BOolean VErifier* (BOVE) [4]. We do not compare our results with any other tool because we want to discuss only the improvement obtained exploiting the described method over the standard methodology. Furthermore, due to the specific features implemented in BOVE (e.g., ternary encodings, gated clocks, initial state set evaluated during an initial synchronization sequence computation [20], etc.), any comparison would be unfair.

We present experiments on two separate sets of circuits:

- Standard benchmarks: ISCAS'89 [6], ISCAS'89-addendum [1], and circuits belonging to the VIS distribution [5].

- Industrial designs: The IBM Formal Verification Benchmark Library [18]. This suite includes 75 circuits in Blif format. They contain from 95 to 917 memory elements. For each of them, there is a unique output (formula_1) which also indicates the property to check.

Among the previous sets we selected all verification instances hard-enough to be improved with our technique. All our data are collected on a Pentium IV 1700 MHz Workstation equipped with 1 GByte main memory, running RedHat Linux 7.1. We use a time limit of 1 hour for all experiments.

Tables 1 and 2 report our results on BMC and inductive verification respectively.

In both the tables the models are sorted by number of state variables (column # SV). For each design we present a set of properties denoted by P_1, P_2, etc. For each property we indicate the maximum sequential depth explored, i.e., the length of the counterexample B (bound). Within the reported bound, the properties are labeled according to the result they produce: They are either proved correct and denoted by Pass, or they are falsified and labeled by Fail [3].

The subsequent columns report statistics for the original and proposed methods, i.e., number of variables and clauses, memory and time used. More specifically, BOVE Original indicates our implementation (within the BOVE environment) of standard BMC [3] and inductive [23] verification. BOVE with TE is the previous version improved with the target enlargement methodology

[3] For Table 1, where we consider only BMC, when Pass, the property is, of course, proved correct only up to the given bound.

Model	# SV	Property	B	BOVE Original				BOVE with Target Enlargement			
				# Var.	# Clauses	Mem. [MByte]	Time SAT	BBwd	Mem. [MByte]	Time [sec] Trav.	Time [sec] SAT
am2901	68	P_1 Pass	10	23252	67413	–	*ovf*	2	94	223	1653
		P_1 Fail	16	37112	107709	–	*ovf*	2	–	223	*ovf*
philo$_{60}$	120	P_1 Pass	20	94139	276838	72	745	2	74	151	591
		P_1 Pass	40	187859	552778	173	2561	2	129	151	1264
FIFOs	142	P_1 Pass	14	31227	89028	80	1395	2	82	231	494
		P_1 Pass	16	35551	101396	–	*ovf*	2	163	231	1659
s15850.1	534	P_1 Pass	75	351424	969151	96	142	5	81	5	28
		P_1 Fail	76	356097	982054	97	130	5	85	5	48
s13207.1	638	P_1 Pass	109	344363	919464	144	580	4	89	3	157
		P_1 Fail	110	347514	927887	121	384	4	108	3	283
		P_2 Pass)	215	678369	1812302	–	*ovf*	10	216	5	1182
		P_2 Fail	216	681520	1820725	–	*ovf*	10	–	5	*ovf*

Table 1
SAT-based Bounded Model Checking: Comparison between standard and target-enlarged BMC. *ovf* means overflow on memory or time (memory limit 512 MBytes, time limit 3600 sec). – means data not available.

Model	# SV	Property	BOVE Original					BOVE with TE				
			B	Time				B	Time			
				BMC	IBwd	IFwd	Total		BMC	IBwd	IFwd	Total
29_batch	95	P_1 Pass	32	1 %	83 %	16 %	ovf	40	5 %	77 %	18 %	ovf
18_batch	143	P_1 Pass	39	11 %	71 %	18 %	3151	35	13 %	68 %	19 %	1826
16_1_batch	150	P_1 Pass	31	16 %	49 %	35 %	ovf	34	15 %	55 %	30 %	ovf
02_1_batch_3	161	P_1 Pass	28	12 %	42 %	46 %	ovf	37	19 %	40 %	41 %	3354
19_batch	181	P_1 Pass	19	23 %	57 %	20 %	2786	16	20 %	56 %	24 %	1022
22_batch	191	P_1 Fail	15	4 %	83 %	13 %	536	15	12 %	72 %	16 %	197
04_batch	256	P_1 Pass	25	2 %	80 %	18 %	ovf	28	8 %	74 %	18 %	ovf

Table 2
Induction-based Verification: Comparison between standard and target-enlarged induction. ovf means overflow on memory or time (memory limit 512 MBytes, time limit 3600 sec).

described in this paper.

Within Table 1 we used the bounded traversal technique in order to get the enlarged sets, so we report the number of backward reachability steps performed (column BBwd), and the Trav. Time needed to conclude this phase. Notice that in this table we compare experiments using BOVE Original and BOVE with TE with the same value of the bound B. This table shows a maximum gain of about 5X for the first experiment, and a good improvement for most of the benchmarks.

For Table 2 we used the dynamic abstraction technique, so that column B represents the depth reached by BOVE when the result was found or an overflow occurred for the two methodologies respectively. In particular, for enlarged case, the meaning of this column must be intended as the sum of the maximum time step analyzed with SAT and those provided by BDDs ($k = h + d_F + d_B$). Columns BMC, IBwd, and IFwd indicate the percentage of time dedicated to the BMC, backward, and forward induction steps respectively over the total amount of Total time. Statistics about the BDD manipulations are not reported.

Analyzing these data, the first observation we do is that the backward inductive step is almost always the hardest check. Indeed, in our experiments, this step was always the one providing the proof of correctness (when the property was true). Nevertheless, with our new technique, the weights of BMC and the inductive (backward) check have been slightly balanced. At first glance, it could seem that BMC now is more costly than with the original method, but, in absolute terms, the time required for each check has been usually reduced. Notice also that we could almost always explore a deeper time step.

Results are still preliminary yet encouraging for both bounded and unbound SAT-based verification. While in some cases we drastically reduced the verification time (up to a 5x factor), on others we could substantially in-

crement the bound (up to almost 30%) analyzed in the same amount of time, i.e., before expiring system resources.

8 Conclusions and Future Works

BDDs and SAT-solvers are the most widely used core technologies within the verification domain. In this paper we propose an idea to exploit the best of this two methodologies. First of all, we perform symbolic forward and backward reachability analysis to partially carry out the verification task. This step also enlarge the initial and the target sets of states. These enlarged sets are subsequently used within a SAT-based bounded model checking or inductive verification phase. Preliminary experimental results show the potentiality of the approach within academic and industrial tools.

Among the possible future work we envisage two main tasks. First of all, we have to check the methodology on a broader set of experiments possibly using real verification instances from the industrial Intel designs. Secondly, we envisage the possibility to integrate the method with an abstraction–and–refinement approach.

References

[1] *MCNC Private Communication*.

[2] Baumgartner, J. and A. Kuehlmann, *Enhanced Diameter Bounding Via Structural Transformation*, in: *Proc. Design Automation & Test in Europe Conf.*, Paris, France, 2004.

[3] Biere, A., 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.*, New Orleans, Louisiana, 1999, pp. 317–320.

[4] Bischoff, G. P., K. S. Brace, S. Jain and R. Razdan, *Formal Implementation Verification of the Bus Interface Unit for the Alpha 21264 Microprocessor*, in: *Proc. Int'l Conf. on Computer Design*, Austin, Texas, 1997.

[5] Brayton, R. K., et al., *VIS*, in: M. Srivas and A. Camilleri, editors, *Proc. Formal Methods in Computer-Aided Design*, LNCS **1166** (1996), pp. 248–256.

[6] Brglez, F., D. Bryan and K. Koźmiński, *Combinatorial Profiles of Sequential Benchmark Circuits*, in: *Proc. IEEE ISCAS'89*, 1989, pp. 1929–1934.

[7] Bryant, R. E., *Graph–Based Algorithms for Boolean Function Manipulation*, IEEE Trans. on Computers **C–35** (1986), pp. 677–691.

[8] Cabodi, G., P. Camurati, L. Lavagno and S. Quer, *Disjunctive Partitioning and Partial Iterative Squaring: an effective approach for symbolic traversal of large circuits*, in: *Proc. 34th Design Automat. Conf.*, Anaheim, California, 1997, pp. 728–733.

[9] Cabodi, G., S. Nocco and S. Quer, *Improving SAT-based Bounded Model Checking by Means of BDD-based Approximate Traversals*, in: *Proc. Design Automation & Test in Europe Conf.*, Munich, Germany, 2003, pp. 898–903.

[10] Cho, H., G. D. Hatchel, E. Macii, B. Plessier and F. Somenzi, *Algorithms for Approximate FSM Traversal Based on State Space Decomposition*, IEEE Trans. on Computer-Aided Design **15** (1996), pp. 1465–1478.

[11] Copty, F., L. Fix, R. Fraer, E. Giunchiglia, G. Kamhi, A. Tacchella and M. Y. Vardi, *Benefits of Bounded Model Checking at an Industrial Setting*, in: G. Berry, H. Comon and A. Finkel, editors, *Proc. Computer Aided Verification*, LNCS **2102** (2001), pp. 435–453.

[12] Ganai, M. K., A. Aziz and A. Kuehlmann, *Enhancing Simulation with BDDs and ATPG*, in: *Proc. 36th Design Automat. Conf.*, New Orleans, LA, 1999, pp. 385–390.

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

[14] Gupta, A., M. Ganai, C. Wang, A. Yang and P. Ashar, *Learning from BDDs in SAT-based Bounded Model Checking*, in: *Proc. 40th Design Automat. Conf.*, Anaheim, CA, 2003, pp. 824–829.

[15] Gupta, A., M. Ganai, C. Wang, Z. Yang and P. Ashar, *Abstraction and BDDs Complement SAT–Based BMC in* Diver, in: W. A. H. Jr. and F. Somenzi, editors, *Proc. Computer Aided Verification*, LNCS **2725** (2003), pp. 206–209.

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

[17] Gupta, A., Z. Yang, P. Ashar, L. Zhang and S. Malik, *Partition–Based Decision Heuristic for Image Computation using SAT and BDDs*, in: *Proc. Int'l Conf. on Computer-Aided Design*, San Jose, California, 2001.

[18] IBM Formal Verification Benchmark Library, http://www.haifa.il.ibm.com/-projects/-verification/-rb_homepage/fv-benchmarks.html.

[19] McMillan, K. L., *Interpolation and SAT-Based Model Checking*, in: W. A. H. Jr. and F. Somenzi, editors, *Proc. Computer Aided Verification*, LNCS **2725**, Boulder, CO, USA, 2003, pp. 1–13.

[20] Pixley, C., S. W. Jeong and G. D. Hachtel, *Exact Calculation of Synchronization Sequences Based on Binary Decision Diagrams*, in: *Proc. 29th Design Automat. Conf.*, 1992, pp. 614–619.

[21] Ravi, K. and F. Somenzi, *High–Density Reachability Analysis*, in: *Proc. Int'l Conf. on Computer-Aided Design*, San Jose, California, 1995, pp. 154–158.

[22] Ravi, K. and F. Somenzi, *Hints to Accelerate Symbolic Traversal*, in: *Correct Hardware Design and Verification Methods (CHARME'99)*, LNCS **1703** (1999), pp. 250–264.

[23] Sheeran, M., S. Singh and G. Stålmarck, *Checking Safety Properties Using Induction and SAT Solver*, in: W. A. Hunt and S. D. Johnson, editors, *Proc. Formal Methods in Computer-Aided Design*, LNCS **1954** (2000), pp. 108–125.

[24] Yang, C. H. and D. L. Dill, *Validation with Guided Search of State Space*, in: *Proc. 35th Design Automat. Conf.*, San Francisco, California, 1998.

[25] Zhang, L. and S. Malik, *The Quest for Efficient Boolean Satisfiability Solvers*, in: E. Brinksma and K. G. Larsen, editors, *Proc. Computer Aided Verification*, LNCS **2404**, Cophenagen, Denmark, 2002, pp. 17–36.