# An Integrated Approach for Combining BDD and SAT Provers

Rolf Drechsler          Görschwin Fey          Sebastian Kinder

*Institute of Computer Science, University of Bremen, 28359 Bremen, Germany*
{drechsle,fey,kinder}@informatik.uni-bremen.de

## Abstract

*Many formal verification tools today are based on Boolean proof techniques. The two most powerful approaches in this context are Binary Decision Diagrams (BDDs) and methods based on Boolean Satisfiability (SAT). Recent studies have shown that BDDs and SAT are orthogonal, i.e. there exist problems where BDDs work well, while SAT solvers fail and vice versa. Beside this, the techniques are very different in general. E.g. SAT solvers try to find a single solution and BDDs represent all solutions in parallel.*

*In this paper the first integrated approach is presented that combines BDDs and SAT within a single data structure. This hybrid approach combines the advantages of the two techniques, i.e. multiple solutions can be computed while the memory requirement remains small. First experimental results demonstrate the quality of the approach in comparison to BDDs and SAT solvers.*

## 1. Introduction

Many problems in circuit design can easily be formulated in terms of Boolean variables. E.g. in verification or automatic test pattern generation a satisfying assignment for a Boolean formula has to be determined (see e.g. [28, 2, 25]). Several Boolean techniques to solve this problem have been proposed in the past. Between them are simulation based approaches, like random pattern simulation. But with increasing design complexity pure simulation is not sufficient to find solutions in huge search spaces. For this, complete methods based on formal proof techniques have been proposed.

The two most frequently used methods are *Binary Decision Diagrams* (BDDs) and provers for *Boolean Satisfiability* (SAT). Experimental studies have shown that these techniques are orthogonal, i.e. there exist problems where BDDs work well, while SAT solvers fail and vice versa. This trade-off can even be formally proven [9].

BDDs and SAT provers are very different in nature. While BDDs compute all solutions in parallel, they require a large amount of memory. In contrast SAT is very efficient regarding memory consumption, but only gives a single solution. There are many applications where multiple solutions are needed (see e.g. [7, 12]). Motivated by this, many authors tried to combine the best of the two approaches, by applying SAT solvers and BDDs alternatively or iteratively.

Even though remarkable results have been obtained, so far none of the approaches considered an integration of the two methods within a single data structure. (A more detailed discussion of related work is given in the next section.)

In this paper we present the first approach that allows to tightly combine BDDs and SAT. Even though the overall principle of the two techniques is very different, there are also some similarities. In both concepts, starting from a Boolean description the problem is decomposed by assigning a Boolean value to a variable. This has already been observed in [22]. For this, we introduce the concept of *expansion nodes*. The given Boolean problem is initially represented by a single expansion node that is recursively expanded. If this is done in a strict *Depth First Search* (DFS) manner, the resulting algorithm is close to a SAT procedure. But if all operations are carried out symbolically, the algorithm computes a BDD. The relation between the two approaches is discussed in more detail later. Experimental results demonstrate the efficiency of the approach.

The paper is structured as follows: Related work is discussed in Section 2. SAT and BDDs are briefly reviewed in Section 3 to make the paper self-contained. Then, the relation between the two is considered. The new approach is presented in Section 4. In Section 5 experiments are presented. Finally the results are summarized and directions for future work are given.

## 2. Related Work

In this section we discuss earlier work that is related to our approach.

In the context of extensions of the classical BDD concept introduced by Bryant [5], some approaches have been presented that make use of different types of functional nodes. These nodes have been used to speed up the construction process [13, 1] or to reduce the size of the graph [14, 17]. To reduce the memory requirement also streaming BDDs have been proposed [18]. But all these approaches only considered BDDs.

The approach in [23] keeps control of the memory needed for the BDD construction by projecting some parts of the graph to a new terminal node $U$ (=unknown), but by this gives up exactness.

Another recent direction of research are efficient all-solution SAT solvers, that do not stop after reaching the first satisfying assignment, but calculate all possible satisfying solutions, e.g. [15]. A drawback of these approaches
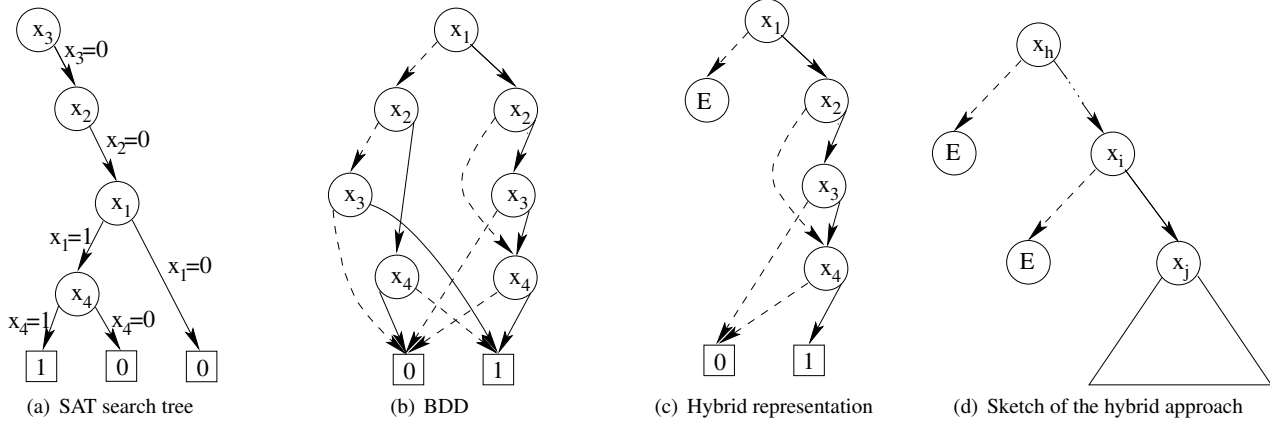
| (a) SAT search tree | (b) BDD | (c) Hybrid representation | (d) Sketch of the hybrid approach |

**Figure 1. Different approaches**

is the potentially large representation of all solutions usually as cubes or as BDDs. In contrast our approach targets applications where not *all* but a set of *good* solutions is needed.

Recently, several techniques have been proposed to combine BDDs and SAT solvers (see e.g. [10, 6, 24]), but no real integration is done. Instead, the proof engines are started one after the other, or alternating. By this, often good experimental results have been obtained, demonstrating the potential of an integrated approach.

## 3. Proof Techniques

In this section we briefly review BDDs and SAT. Then the relation between the two is discussed to provide a better understanding and a motivation for the hybrid approach presented below.

### 3.1. BDD

As well-known each Boolean function $f : \mathbf{B}^n \rightarrow \mathbf{B}$ can be represented by a *Binary Decision Diagram* (BDD) [5], i.e. a directed acyclic graph where the Shannon decomposition with respect to a variable $x_i$ is carried out in each node:

$$f = \overline{x}_i f_{x_i=0} + x_i f_{x_i=1}$$

A BDD is called *ordered* if each variable is encountered at most once on each path from the root to a terminal and if the variables are encountered in the same order on all such paths. A BDD is called *reduced* if it does not contain vertices either with isomorphic subgraphs or with both edges pointing to the same node.

Reduced, ordered BDDs are a canonical data structure for Boolean functions and allow efficient manipulations [5]. In the following only reduced, ordered BDDs are considered and for briefness these graphs are called BDDs.

### 3.2. SAT

Let $f$ be a Boolean function in *Conjunctive Normal Form* (CNF), i.e. in a product-of-sum representation. Then, the problem of *Boolean Satisfiability* (SAT) is to determine an assignment of the variables of $f$ such that $f$ evaluates to 1 or to prove that such an assignment does not exist.

**Example 1.** *Let $f = (x_1 + x_2 + \overline{x}_3)(\overline{x}_1 + x_3)(\overline{x}_2 + x_3)$. Then $x_1 = 1$, $x_2 = 1$ and $x_3 = 1$ is a satisfying assignment. The values of $x_1$ and $x_2$ ensure that the first sum becomes 1, while $x_3 = 1$ ensures this for the remaining sums.*

In many applications, like formal verification and automatic test pattern generation, the problem is initially given in the form of a circuit. This circuit can be transformed to a CNF by a simple transformation.

Recently, several very powerful tools have been developed that make use of e.g. Boolean constraint propagation and clause recording to speed up the proof process [16, 20, 8].

### 3.3. Discussion

Both techniques have advantages and disadvantages. While BDDs represent all solutions in parallel at the cost of large memory requirements, A SAT solver only provides a single solution, while the memory needed is very low. In [22] the relation between BDDs and SAT has been studied from a theoretical point of view. It has been proven that the BDD corresponds to a complete representation of the SAT backtrack tree, if a fixed variable order is assumed.

As a motivation for the next section, where our approach is described in more detail, an example is given to show the main difference between SAT and BDDs. We will later come back to this example.

**Example 2.** *Consider a Boolean function $f$ over four variables given by*

$$\begin{aligned} f \quad = \quad & (x_1 + x_2 + x_3)(x_1 + \overline{x}_2 + \overline{x}_4)(\overline{x}_1 + x_2 + x_4) \\ & (\overline{x}_1 + \overline{x}_2 + x_3)(\overline{x}_1 + \overline{x}_2 + \overline{x}_3 + x_4) \end{aligned}$$

*A sketch of the search tree, if the function is processed by a SAT solver is shown in Figure 1(a). The corresponding BDD is given in Figure 1(b). As can be seen, the SAT solver by construction only gives a single solution, while the BDD represents all satisfying assignments in parallel at the cost of a larger number of nodes.*

## 4. Hybrid Approach

In this section we describe our approach for BDD and SAT integration. First, the overall idea is given. Then the concept of *expansion nodes* is introduced followed by a discussion of expansion heuristics. Finally, we comment on some issues related to an efficient implementation.

### 4.1. Basic Idea

In our approach we start the processing by symbolic operations analogously to BDDs. For the operations the ITE operator [3] has been modified. During the starting phase, the constructed graphs are simply BDDs. But when composing BDDs a heuristic is used to decide, which parts of the solution space are explored.

To guarantee that the algorithm is exact, i.e. no solution is missed, a node is introduced where the computation can be resumed. These nodes are called *expansion nodes* in the following. By this, our approach stores all necessary information resulting in a complete proof method.

A sketch of a configuration during the run is shown in Figure 1(d). In this case the upper part is "SAT-like" while the lower part is a complete symbolic representation as it occurs in BDDs. The expansion nodes are denoted by $E$. The decomposition nodes are labeled by variables, these variables occur in the same order on all paths. In the following we refer to such graphs that allow a smooth transition between SAT and BDDs as *hybrid structure*.

**Remark 1.** *Several expansion nodes in a hybrid structure may represent the same function. This cannot be detected before completely expanding the node. Thus, a hybrid structure is not a canonical representation of Boolean functions.*
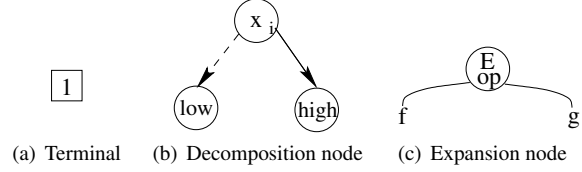
### 4.2. Expansion Nodes

The hybrid approach makes use of three types of nodes (see Figure 2):

(a) Terminal nodes
(b) Decomposition nodes
(c) Expansion nodes

The first two can also be found in BDDs. Terminal nodes represent the constant functions $0$ and $1$. In decomposition nodes the Shannon decomposition is carried out.

Expansion nodes are labeled by a Boolean operation $op$ and have two successors $f$ and $g$, that represent Boolean functions (which are also denoted by $f$ and $g$ for simplicity). The expansion node represents the function $f \; op \; g$.

**Example 3.** *Consider again the function from Example 2 and Figures 1(a) and 1(b). A possible hybrid structure is shown in Figure 1(c). This one results if the top variable is only decomposed in one direction, while on the other branch an expansion node is placed. As can be seen the structure is more memory efficient. Compared to the BDD five instead of seven nodes are needed. At the same time three solutions are represented in contrast to the SAT approach that only returns a single solution.*



**Figure 2. Overview over different node types**

This simple example demonstrated that the approach combines the two proof techniques SAT and BDD. A crucial point to address is where to place the expansion nodes. For this, we propose a heuristic in the next section.

### 4.3. Expansion Heuristics

Inserting expansion nodes at suitable locations is crucial for the approach to work. If too many expansion nodes are inserted, no solutions can be found. Only structures without a path to a terminal will be constructed and the expansion of partial trees will take most of the run-time until computing a solution. On the other hand not inserting enough expansion nodes will lead to a memory blow-up as known from BDDs.

In a BDD-based approach the final solutions are computed by composing intermediate BDDs. This is similar for the new approach. The following steps are necessary to retrieve solutions:

(1) Build BDDs for basic functions without any expansion nodes.
(2) Compose the basic functions and insert expansion nodes according to a predetermined heuristic.
(3) Select expansion nodes to calculate the final solutions.

Which functions are considered as basic functions in step (1) depends on the problem and the input format, e.g. projection functions and cubes were chosen in our experiments. Building BDDs for these basic functions is not necessary for the approach to work, but having the basic functions completely represented improves the performance drastically by reducing the number of necessary expansions.

So far the following two heuristics to limit the size of the resulting hybrid structure in step (2) have been evaluated:

(S1) A fast procedure is to directly limit the memory consumption. This limit can be detected efficiently. Once the limit is reached no further decomposition nodes are created, but only expansion nodes. Therefore, prior to performing an expansion the memory limit is increased by a user defined value.

(S2) The second procedure is to limit the number of nodes in a subgraph to a certain threshold. Tracking this limit is computationally more expensive. But allowing more than $n$ nodes in a subgraph guarantees that there is at least one path to a terminal node. I.e. for at least one assignment the function can directly be evaluated.

The selection of nodes to expand in step (3) has also been done using two different heuristics:

(E1) Randomly
(E2) Heuristically (using the algorithm in Figure 3): The hybrid structure is traversed in a depth first manner until an expansion node is reached. This node is selected and then expanded by carrying out the stored operation. The same scheme is applied recursively if further selections are necessary.

```
1   Node∗ DFS(N){
2       if(isTerminal(N)) return NULL;
3       tmp = DFS(Nhigh);
4       if(tmp) return tmp;
5       if(isFuncNode(N)) return N;
6       tmp = DFS(Nlow);
7       return tmp;
8   }
```

**Figure 3. Depth first traversal**

```
1   struct Node {
2       HalfWord index;
3       HalfWord ref;
4       Node ∗next;
5       union {
6           Terminal value;
7           Children kids;
8           ExpNode func;
9       }
10  }
11
12  struct ExpNode{
13      Node ∗F;
14      Node ∗G;
15  }
```

**Figure 4. Modified node structure**

Here, (E2) also heuristically ensures a moderate growth of the memory needs. Experimental studies showed that the combination of a hard limit on memory consumption (S1) with deterministic DFS (E2) gives the best results, i.e. small run-times and a large number of solutions. From a more general point of view this combination of heuristics leads to a SAT-like search tree in the upper part of the hybrid structure which is enriched by a BDD-like lower part.

**Remark 2.** *When using heuristics (S1) and (E2) in combination the search space is traversed similar as with "BDDs at SAT leaves" in [10, 11]. But the proposed hybrid structure is more general in the sense that switching between SAT-like and BDD-like behavior is subject to heuristics.*

**Remark 3.** *During expansion canonicity is also an issue. When expanding a node, a function that is already represented by another node may be the result. The hybrid structure can be reduced at a computational cost linear in the number of nodes using an algorithm similar to [26]. In our implementation no reduction was carried out to save run-time.*

### 4.4. Implementation

The technique described above has been integrated into the CUDD package [27], where the core data structures are taken from. To store the expansion nodes, the structure for storing nodes has been extended (see line 8 in Figure 4). The structure for the new type is given in lines 12-15.

In case of an expansion node, also the operation has to be stored. For reasons of efficiency we restrict ourselves to store only operations of type AND and XOR. Negation is

**Table 1. Index of node types (32-bit)**

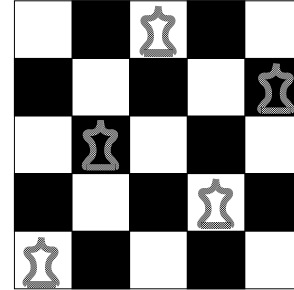| Node type | Index |
|---|---|
| decomposition nodes | 0 - 65532 |
| XOR-node | 65533 |
| AND-node | 65534 |
| terminal node | 65535 |



**Figure 5. Solution for the 5-Queens problem**

realized by complemented edges [3]. All other Boolean operators are mapped accordingly. The information is stored in the index of each node. The complete encoding is given in Table 1, i.e. three indices have a special meaning, while all the remaining ones are used for decomposition variables.

## 5. Experimental Results

In the following we present two types of experiments. The well-known $n$-Queens problem is considered as an example of a combinational problem where BDDs are known to perform poorly on large instances while a large number of solutions is available. The synthesis problem of minimizing *EXOR-Sum-Of-Product* (ESOP) representations is studied as an optimization problem that is known to be hard. All experiments have been carried out on an Intel Pentium 4 processor with 3 GHz and 1 GByte of main memory running Linux.

### 5.1. $n$-Queens

The $n$-Queens problem is a well-known combinational problem. The objective is to place $n$ queens on an $n \times n$ board such that no queen can be captured by another one. An example for a solution of the 5-Queens problem is shown in Figure 5. This game problem is encoded using $n^2$ binary input variables, each one deciding, if a queen is placed on the corresponding field of the chess board or not. Obviously the constraints are to place one queen per row and column and at most one queen per diagonal.

In a first experiment the heuristics to limit the size were considered. For all experiments the limits were loose enough to retrieve all solutions. Therefore the overhead of the heuristics to limit the size can directly be measured in comparison to BDDs. Results are reported in Table 2. Given are the number of solutions for increasing values of $n$ and run-times in CPU seconds for BDDs and the two heuristics introduced in Section 4.3, respectively. The resource requirements for BDDs increase rapidly and no further solutions beyond $n = 13$ could be retrieved. Also the

**Table 2. Heuristics to limit the size of the hybrid structure**

| | | BDD | Memory (S1) | | Subgraph (S2) | |
|---|---|---|---|---|---|---|
| n | #sol. | sec. | sec. | overhead | sec. | overhead |
| 6 | 4 | 0.00 | 0.00 | - | 0.01 | - |
| 7 | 40 | 0.01 | 0.01 | 0.00 % | 0.03 | 200.00 % |
| 8 | 92 | 0.05 | 0.06 | 20.00 % | 0.18 | 260.00 % |
| 9 | 352 | 0.37 | 0.37 | 0.00 % | 1.30 | 251.35 % |
| 10 | 724 | 1.56 | 1.59 | 1.92 % | 8.20 | 425.64 % |
| 11 | 2680 | 7.81 | 7.82 | 0.13 % | 62.39 | 698.84 % |
| 12 | 14200 | 48.12 | 48.54 | 0.87 % | 490.33 | 918.97 % |
| 13 | 73712 | 352.11 | 353.21 | 0.31 % | 4566.75 | 1196.97 % |

computational overhead of limiting the size of subgraphs using heuristic (S2) is too large. But directly limiting the memory consumption according to heuristic (S1) does introduce almost no overhead. This heuristic has been used in all remaining experiments to restrict the size.

The performance of heuristics to select nodes for expansion has been investigated in the next experiment. Expansion was carried out until a total memory limit of 750 MB was reached. Due to the expansion of subfunctions more than one solution can be contained in the final representation. The results are shown in Table 3. Up to $n = 13$ all solutions were obtained with both heuristics.

Then, the random selection performs very poorly. When expanding the last node in a cascade of expansion nodes new decomposition nodes are created. But the next expansion will often occur at an expansion node in a different subgraph. Thus, the previously created decomposition nodes cannot be utilized for the next step.

In contrast the deterministic DFS starts the next expansion where new decomposition nodes have been constructed previously. As a result the new approach yields solutions up to $n = 21$ in a moderate amount of time.

## 5.2. ESOP Minimization

Compared to a SOP-representation of a function the ESOP-representation can be exponentially smaller. But most algorithms for ESOP minimization only apply local transformations to improve from an initial solution, e.g. [4, 19]. In [21] the problem to compute an ESOP for a given Boolean function $f$ over $n$ variables has been formulated using the Helliwell equation. The Helliwell equation $H_f$ for function $f$ has $3^n$ input variables, each input variable corresponds to a cube and is 1, iff this cube is chosen for the ESOP of $f$. A satisfying assignment to $H_f$ determines an ESOP for $f$ and vice versa. The hybrid structure was built for the Helliwell equation. By additional constraints the number of cubes was limited to be at most $k$. The experimental results for applying this method to $f = \bigoplus_{i=1}^{4} x_i$ are shown in Table 4. Given are results for using BDDs, the hybrid structure, and the SAT solver zchaff [20]. We modified the SAT solver zchaff to calculate more than one solution: For each solution a blocking clause is added and the solve process is continued. For the hybrid structure results are reported when different numbers of solutions are calculated: more than 1, more than $10^3$ and more than $10^6$ solutions,

**Table 3. Selection of expansion nodes**

| | | Randomly (E1) | | DFS (E2) | |
|---|---|---|---|---|---|
| n | #var | #sol. | sec. | #sol. | sec. |
| 3 | 9 | 0 | 0.00 | 0 | 0.00 |
| 4 | 16 | 2 | 0.00 | 2 | 0.00 |
| 5 | 25 | 10 | 0.00 | 10 | 0.00 |
| 6 | 36 | 4 | 0.00 | 4 | 0.00 |
| 7 | 49 | 40 | 0.02 | 40 | 0.01 |
| 8 | 64 | 92 | 0.06 | 92 | 0.06 |
| 9 | 81 | 352 | 0.37 | 352 | 0.37 |
| 10 | 100 | 724 | 2.10 | 724 | 1.83 |
| 11 | 121 | 2680 | 16.54 | 2680 | 10.30 |
| 12 | 144 | 14200 | 158.86 | 14200 | 73.34 |
| 13 | 169 | 73712 | 2062.39 | 73712 | 578.54 |
| 14 | 196 | 0 | 384.45 | 56672 | 1836.93 |
| 15 | 225 | 0 | 289.01 | 33382 | 1669.50 |
| 16 | 256 | 0 | 652.64 | 20338 | 2555.35 |
| 17 | 289 | 0 | 1366.25 | 5061 | 2055.97 |
| 18 | 324 | 0 | 693.13 | 204 | 2238.79 |
| 19 | 361 | 0 | 529.37 | 1428 | 3357.97 |
| 20 | 400 | 0 | 1923.07 | 38 | 1592.94 |
| 21 | 441 | 0 | 1957.39 | 111 | 1972.60 |

respectively. For different values of $k$ the CPU time in seconds, the memory requirements in kB and the number of nodes in the BDD or the hybrid structure, respectively, are reported. For zchaff the CPU time is given. The number of available solutions is not reported, but grows rapidly. While there are only 38 valid solutions for $k = 4$, there are more than 5000 for $k = 6$ and more than $4 \cdot 10^6$ for $k = 9$.

The results show the superiority of the hybrid approach compared to BDDs. For a tightly restricted solution space ($k < 25$) BDDs are feasible. But after that the memory and especially the run-time requirements grow prohibitively fast. In contrast the hybrid approach exhibits a rather stable performance as CPU time and memory requirements remain in the same order for all runs. The increased run-time for $k = 10, 15$ when calculating more than $10^6$ solutions is due to the small number of possible solutions. In this case a large part of the BDD has to be recreated using the expansion technique without retrieving more solutions. In this case BDDs are faster. But usually even calculating a large number of solutions does not degrade the performance of the new approach.

When calculating a single solution the SAT solver is faster. But even for calculating $10^3$ solutions the computation time increases significantly. Finally, when calculating a large number of solutions the added blocking clauses lead to a memory blow-up even for the SAT solver. Using a more sophisticated approach the blocking clauses could be compacted, but only at the expense of CPU time for logic optimization. By this the new approach provides a good compromise between a SAT-based approach and a BDD-based approach.

## 6. Conclusions and Future Work

We introduced a new approach to handle satisfiability problems. This approach can be seen as an integrated technique using BDDs and SAT solvers and incorporates benefits of both: The memory consumption can be limited while

**Table 4. ESOP minimization**

| | BDD all solutions | | | hybrid structure ≥ 1 solution | | | ≥ $10^3$ solutions | | | ≥ $10^6$ solutions | | | zchaff 1 sol. | $10^3$ sol. | $10^6$ sol. |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| k | sec. | kB | #nodes | sec. | kB | #nodes | sec. | kB | #nodes | sec. | kB | #nodes | sec. | sec. | sec. |
| 4 | 0.55 | 16433 | 628 | 0.50 | 16449 | 568 | 0.53 | 16466 | 1108 | 0.53 | 16466 | 1108 | <0.01 | 0.07 | 0.07 |
| 5 | 0.58 | 16483 | 4075 | 0.53 | 16450 | 638 | 0.60 | 16534 | 4729 | 0.61 | 16534 | 4729 | <0.01 | 0.09 | 0.09 |
| 10 | 1.75 | 23610 | 420655 | 0.47 | 16450 | 145 | 0.70 | 16728 | 11597 | 51.28 | 19140 | 155018 | <0.01 | 0.14 | - |
| 15 | 4.96 | 49270 | 1428139 | 0.48 | 16468 | 352 | 0.61 | 16744 | 11634 | 10.17 | 19420 | 172422 | <0.01 | 0.11 | - |
| 20 | 53.96 | 65539 | 2444782 | 0.47 | 16484 | 112 | 0.54 | 16670 | 7459 | 1.13 | 19516 | 177708 | <0.01 | 0.32 | - |
| 25 | 1945.01 | 84280 | 3449866 | 0.48 | 16500 | 490 | 0.52 | 16582 | 5465 | 0.98 | 18732 | 133396 | <0.01 | 0.37 | - |
| 30 | 9985.37 | 99752 | 4441463 | 0.49 | 16500 | 495 | 0.49 | 16534 | 2618 | 0.66 | 17395 | 48107 | <0.01 | 0.12 | - |
| 35 | 13900.22 | 113883 | 5361182 | 0.52 | 16500 | 544 | 0.51 | 16516 | 878 | 0.75 | 16931 | 21608 | <0.01 | 0.16 | - |
| 39 | 13913.44 | 123635 | 5906441 | 0.44 | 16500 | 217 | 0.45 | 16516 | 1241 | 0.53 | 16662 | 5910 | <0.01 | 0.09 | - |

calculating a large number of solutions in a single run. First heuristics have been proposed and evaluated to increase the performance of the new technique. Experiments show the efficiency of the hybrid technique in contrast to classical approaches.

Future work is the introduction of powerful learning techniques as known from the SAT domain and the application to formal verification. Suitable heuristics for such applications have to be developed.

# References

[1] H. Andersen and H. Hulgaard. Boolean expression diagrams. In *Logic in Computer Science*, pages 88–98, 1997.

[2] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1579 of *LNCS*, pages 193–207. Springer Verlag, 1999.

[3] K. Brace, R. Rudell, and R. Bryant. Efficient implementation of a BDD package. In *Design Automation Conf.*, pages 40–45, 1990.

[4] D. Brand and T. Sasao. Minimization of AND-EXOR expressions using rewrite rules. *IEEE Trans. on Comp.*, 42:568–576, 1993.

[5] R. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. on Comp.*, 35(8):677–691, 1986.

[6] G. Cabodi, S. Nocco, and S. Quer. SAT-based bounded model checking by means of BDD-based approximate traversals. In *Design, Automation and Test in Europe*, pages 898–903, 2003.

[7] G. Fey and R. Drechsler. Finding good counter-examples to aid design verification. In *MEMOCODE*, pages 51–52, 2003.

[8] E. Goldberg and Y. Novikov. BerkMin: a fast and robust SAT-solver. In *Design, Automation and Test in Europe*, pages 142–149, 2002.

[9] J. F. Groote and H. Zantema. Resolution and binary decision diagrams cannot simulate each other polynomially. *Discrete Applied Mathmatics*, 130(2):157–171, 2003.

[10] A. Gupta, Z. Yang, P. Ashar, and A. Gupta. SAT-based image computation with application in reachability analysis. In *Int'l Conf. on Formal Methods in CAD*, volume 1954 of *LNCS*, pages 354–371, 2000.

[11] A. Gupta, Z. Yang, P. Ashar, L. Zhang, and S. Malik. Partition-based decision heuristics for image computation using SAT and BDDs. In *Int'l Conf. on CAD*, pages 286–292, 2001.

[12] C. Haubelt, J. Teich, R. Feldmann, and B. Monien. SAT-based techniques in system synthesis. In *Design, Automation and Test in Europe*, volume 1, pages 11168–11169, 2003.

[13] A. Hett, R. Drechsler, and B. Becker. MORE: Alternative implementation of BDD packages by multi-operand synthesis. In *European Design Automation Conf.*, pages 164–169, 1996.

[14] S. Jeong, B. Plessier, G. Hachtel, and F. Somenzi. Extended BDD's: Trading of canonicity for structure in verification algorithms. In *Int'l Conf. on CAD*, pages 464–467, 1991.

[15] B. Li, M. Hsiao, and S. Sheng. A novel SAT all-solutions solver for efficient preimage computation. In *Design, Automation and Test in Europe*, pages 272–277, 2004.

[16] J. Marques-Silva and K. Sakallah. GRASP – a new search algorithm for satisfiability. In *Int'l Conf. on CAD*, pages 220–227, 1996.

[17] C. Meinel and H. Sack. ⊕-OBDDs - a BDD Structure for Probabilistic Verification. In *Workshop on Probabilistic methods in Verification*, pages 141–151, 1998.

[18] S. Minato. Streaming BDD manipulation. *IEEE Trans. on Comp.*, 51(5):474–485, 2002.

[19] A. Mishchenko and M. Perkowski. Fast heuristic minimization of exclusive-sums-of-products. In *Int'l Workshop on Applications of the Reed-Muller Expansion in Circuit Design*, pages 242–250, 2001.

[20] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Design Automation Conf.*, pages 530–535, 2001.

[21] M. Perkowski and M. Chrzanowska-Jeske. An exact algorithm to minimize mixed-radix exclusive sums of products for incompletely specified Boolean functions. In *Int'l Symp. Circ. and Systems*, pages 1652–1655, 1990.

[22] S. Reda, R. Drechsler, and A. Orailoglu. On the relation between SAT and BDDs for equivalence checking. In *Int'l Symp. on Quality Electronic Design*, pages 394–399, 2002.

[23] D. Ross, K. Butler, R. Kapur, and M. Mercer. Fast functional evaluation of candidate OBDD variable ordering. In *European Conf. on Design Automation*, pages 4–9, 1991.

[24] S. Safarpour, G. Fey, A. Veneris, and R. Drechsler. Utilizing don't care states in SAT-based bounded sequential problems. In *Great Lakes Symp. VLSI*, pages 264–269, 2005.

[25] J. Shi, G. Fey, R. Drechsler, A. Glowatz, F. Hapke, and J. Schlöffel. PASSAT: Effcient SAT-based test pattern generation for industrial circuits. In *IEEE Annual Symposium on VLSI*, pages 212–217, 2005.

[26] D. Sieling and I. Wegener. Reduction of BDDs in linear time. *Information Processing Letters*, 48(3):139–144, 11 1993.

[27] F. Somenzi. Efficient manipulation of decision diagrams. *Software Tools for Technology Transfer*, 3(2):171–181, 2001.

[28] P. Stephan, R. Brayton, and A. Sangiovanni-Vincentelli. Combinational test generation using satisfiability. *IEEE Trans. on CAD*, 15:1167–1176, 1996.