FACTA UNIVERSITATIS (NIŠ)
SER.: ELEC. ENERG. vol. 20, no. 3, December 2007, 381-394

Building Free Binary Decision Diagrams Using SAT Solvers

Robert Wille, Görschwin Fey, and Rolf Drechsler

Abstract: Free Binary Decision Diagrams (FBDDs) are a data structure for the representation of Boolean functions. In contrast to Ordered Binary Decision Diagrams (OBDDs) FBDDs allow different variable orderings along each path. Thus, FBDDs are the more compact representation while most of the properties of OBDDs are kept. However, how to efficiently build small FBDDs for a given function is still an open question. In this work we propose FBDD construction with the help of SAT solvers. 'Recording' the single steps of a SAT solver during the search process leads to an FBDD.

Furthermore, by exploiting approaches for identifying isomorphic sub-graphs, i.e. cutlines or cutsets, reduced FBDDs are constructed.

Keywords: Decision diagrams, Boolean satisfiability, logic synthesis, data structures.

1 Introduction

Ordered Binary Decision Diagrams (OBDDs) [1] are a data structure for efficient representation and manipulation of Boolean functions. OBDDs have been successfully applied in many applications in VLSI CAD, but due to the ordering restrictions, there are many Boolean functions for which no efficient representation exists. This is the reason why in the last 15 years several extensions to OBDDs have been proposed. While some of them modify the decomposition type assigned to each node, others loose the ordering restriction. For an overview on the various types see [2].

As one extension *Free BDDs* (FBDDs) have been proposed. In FBDDs the Shannon decomposition is carried out in each node, analogously to OBDDs, but

Manuscript received August 31, 2007.

The authors are with the Institute of Computer Science at the University of Bremen, Germany (e-mail: [rwille,fey,drechsle]@informatik.uni-bremen.de).

along different paths different orderings may occur. However, each variable is still allowed to appear only once. If the variables are chosen in the same order along all paths, OBDDs result. In this sense FBDDs are a superset of OBDDs. With some modification FBDDs are a canonical data structure [3]. There exist several theoretical studies (e.g. [4]) that show that Boolean functions exist that can be efficiently represented by FBDDs, i.e. in polynomial space, while each OBDD requires an exponential number of nodes, independent of the chosen variable ordering. The main problem is that there do not exist efficient heuristics for FBDD construction. First approaches have been presented in [5, 6, 7], but no significant improvements over OBDDs have been reported.

In the recent past, there is a significant improvement in algorithms for solving *Boolean Satisfiability* (SAT). The resulting tools, so called SAT solvers, typically work on *Conjunctive Normal Forms* (CNFs) and can cope with instances of several thousand clauses and literals. The search tree traversed by a SAT solver results in an OBDD, if the variable ordering is fixed [8, 9]. If the variable decisions vary along different paths, the underlying structure becomes an FBDD.

In this paper we investigate the use of SAT solvers for minimization of FBDDs[1]. Since BDDs in general profit from merging of nodes, algorithms are studied that try to maximize the node sharing. The algorithms are based on cutsets and cutlines. Similar approaches based on cutsets were already used before for OBDD construction [9]. Cutlines have been applied successfully to reachability analysis [12, 13].

This paper is structured as follows. Section 2 briefly reviews the background needed for the rest of this paper and formulates the problem considered here. The general idea of using SAT solvers for FBDD construction is described in Section 3. For identifying isomorphisms two approaches – cutlines and cutsets – are presented in Section 4. Finally, experimental results and a conclusion are given in Section 5 and Section 6, respectively.

2 Background and Problem Formulation

2.1 Boolean functions

In this work the representation of Boolean functions as FBDDs is considered. A Boolean function is a mapping $f : \mathbb{B}^n \rightarrow \mathbb{B}$ over Boolean variables $X = \{x_1, x_2, \ldots, x_n\}$. Each Boolean function can be represented in *Conjunctive Normal Form* (CNF). A CNF is the conjunction of clauses. A clause is a disjunction of literals and each literal is a propositional variable (x_i) or its negation ($\overline{x}_i$).

[1] This work summarizes the results obtained by a diploma thesis at the University of Bremen [10]. Preliminary results have been proposed in [11].

2.2 Binary decision diagrams

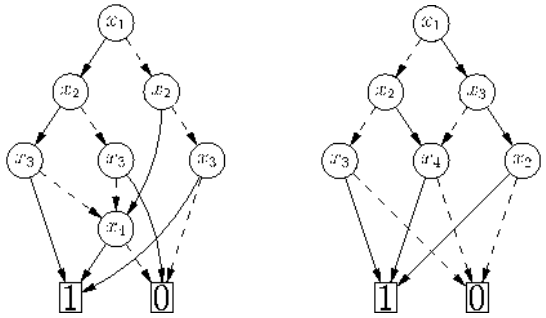
As is well-known a Boolean function f can be represented by a *Binary Decision Diagram* (BDD) [1]. A *Binary Decision Diagram* is a directed acyclic graph where a Shannon decomposition

$$f = \overline{x}_i f_{x_i=0} + x_i f_{x_i=1} \quad (1 \leq i \leq n)$$

is carried out in each node.

A BDD is called *Free Binary Decision Diagram* (FBDD) if each variable is encountered at most once on each path from the root to a terminal node. A BDD is called *Ordered Binary Decision Diagram* (OBDD) if in addition all variables are encountered in the same order on all such paths. The *size* of a decision diagram is defined by the number of nodes.

Since OBDDs are restricted to a given order for all paths, the size for OBDDs significantly depends on the choosen ordering [14]. In contrast this restriction is loosen when FBDDs are used. This may lead to smaller representations.



(a) OBDD (b) FBDD

Fig. 1. OBDD and FBDD for $f = \overline{x}_1 \cdot \overline{x}_2 \cdot x_3 + \overline{x}_1 \cdot x_2 \cdot x_4 + x_1 \cdot \overline{x}_3 \cdot x_4 + x_1 \cdot x_2 \cdot x_3$

Example 1 *Figure 1 shows the minimal OBDD (a) and an FBDD (b) representing the function* $f(x_1,x_2,x_3,x_4) = \overline{x}_1 \cdot \overline{x}_2 \cdot x_3 + \overline{x}_1 \cdot x_2 \cdot x_4 + x_1 \cdot \overline{x}_3 \cdot x_4 + x_1 \cdot x_2 \cdot x_3$ *(taken from [3]). Since the OBDD is restricted to a fixed order for all paths at least seven nodes are needed to represent the function. In contrast the FBDD can be built with six nodes.*

The main problem to be solved is now, how to find a small FBDD for a given function.

2.3 Boolean satisfiability

For FBDD construction Boolean satisfiability is utilized in this work. The *Boolean Satisfiability* (SAT) problem is defined as follows:

Definition 1 *Let f be a Boolean function in* Conjunctive Normal Form *(CNF). Then the SAT problem is to determine whether there exists an assignment to the variables of f such that f evaluates to true or to prove that no such assignment exists.*

Example 2 *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 for f. The values of x_1 and x_2 ensure that the first clause becomes* 1 *while x_3 ensures this for the remaining two clauses.*

The corresponding decision problem is one of the central $\mathcal{NP}$-complete problems. In fact, it was the first known $\mathcal{NP}$-complete problem as proven by Cook in 1971 [15]. Today SAT is not only used in theorem proving, but in many application domains, like automatic test pattern generation [16] and verification [17].

In the past several (backtracking) algorithms (so called *SAT solvers*) have been proposed [18, 19, 20, 21]. Most of them are based on three essential procedures:

1. The decision heuristic assigns values to free variables,
2. the propagation procedure determines implications due to the last assignment(s) and,
3. the conflict analysis resolves conflicts by backtracking.

Advanced techniques like e.g. *efficient Boolean constraint propagation* [20] or *conflict analysis* [19] are common in state-of-the-art SAT solvers today.

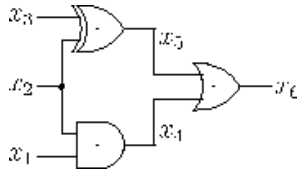
2.4 Circuits

Circuits represent Boolean functions. A circuit is a graph, where the signals are represented by edges and the gates are represented by nodes. In this work only circuits with one output are considered.

Remark 1 *Circuits with more than one output can be transformed into a representation with one output by modifying the underlying function f to its characteristic function χ_f.*

Each circuit can be transformed into a CNF. Therefore, for each internal signal of the circuit a new free Boolean variable is introduced while each node is substituted by a set of clauses according to the functionality of the respective gate. This transformation is linear in time and space in the size of the circuit [22]. By additionally assigning the output to one (by a *unit clause*, i.e. by a clause consisting of one literal only), a CNF results which is equivalent to the function represented by the circuit.

Example 3 *Figure 2(a) shows a circuit with three inputs, one output and three gates. The corresponding CNF representing this circuit is given in Figure 2(b). By adding the unit clause x_6, a formula results which is equivalent to the function represented by the circuit.*



(a) Circuit

AND:	XOR:	OR:
$x_4 + \overline{x}_1 + \overline{x}_2$	$\overline{x}_2 + x_3 + x_5$	$\overline{x}_6 + x_4 + x_5$
$\overline{x}_4 + x_1$	$x_2 + \overline{x}_3 + x_5$	$x_6 + \overline{x}_4$
$\overline{x}_4 + x_2$	$x_2 + x_3 + \overline{x}_5$	$x_6 + \overline{x}_5$
	$\overline{x}_2 + \overline{x}_3 + \overline{x}_5$	

(b) CNF

Fig. 2. Transforming a circuit into CNF

Since the values of all internal signals and primary output signals can be determined from the values of the primary inputs, only inputs are considered as the support for the CNF derived from the circuit. In the following only variables in the support are considered for FBDD construction.

In the rest of this work the following problem is considered:

> *How can we obtain an FBDD representing a Boolean function with the help of a SAT solver?*

Thereby the function is always provided in CNF or derived from a circuit representation.

3 Using SAT Solvers for FBDD Construction

In this section the general idea for FBDD construction with SAT solvers is proposed. SAT solvers search for an assignment α for a given function f such that $f(\alpha) = 1$ or prove that no such assignment exists. During the search process conflicting assignments may be found or implications are performed. The SAT solver terminates if a satisfying assignment is found. Observing this process leads to properties of the search process which can be exploited for FBDD construction.

Observation 1 *Let $f : \mathbb{B}^n \to \mathbb{B}$ be a Boolean function in CNF. Then, three properties for FBDD construction hold:*

1. *Each satisfying assignment found by a SAT solver corresponds to a* 1*-path of an FBDD representing f.*

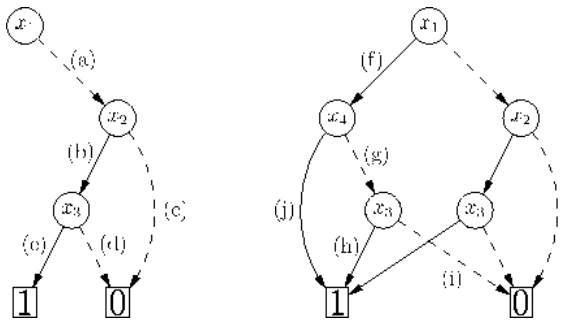


Fig. 3. FBDD construction using SAT solver

2. *Each conflicting assignment determined by the SAT solver corresponds to a* 0*-path of an FBDD representing* f.
3. *Each implication to* $x_i = b$ *(*$x_i \in f, b \in \mathbb{B}$*) performed by the SAT solver leads to a* 0*-path of an FBDD representing* f. *This path can be constructed by using the current assignment of the SAT solver and* $x_i = \overline{b}$.

According to these properties 'recording' the single steps of a SAT solver during the search process leads to a partial FBDD. The following example describes this idea in more detail.

Example 4 *With the help of a SAT solver an FBDD is constructed, which represents the function* $f(x_1,x_2,x_3,x_4) = (x_1 + x_2) \cdot (x_3 + x_4) \cdot (x_1 + x_3 + \overline{x}_4)$. *For this purpose each step of the SAT solver is recorded. These steps are illustrated in the left side of Figure 3.*

At the beginning the solver assigns $x_1 = 0$ *using the decision heuristic (a). This leads to* $x_2 = 1$, *due to the first clause (b). Because the last step was an implication, according to Property 3 we conclude that* $x_1 = 0$ *and* $x_2 = 0$ *is a conflicting assignment, i.e.* $f(0,0,x_3,x_4) = 0$, *which is represented by a* 0*-path (c). In a next step* $x_3 = 0$ *is assigned. This causes a conflict and a* 0*-path is recorded (d) according to Property 2. After backtracking to* $x_3 = 1$, *all clauses become satisfied and thus a satisfying assignment is found. This is recorded by a* 1*-path (e) according to Property 1. Because a satisfying assignment is found, the SAT solver terminates.*

As shown in the left part of Figure 3, a partial FBDD representing the function was created by starting the SAT solver and recording all assignments. To retrieve a complete FBDD, the SAT solver is used to search for all solutions instead of only one (also known as *All Solution SAT*). To this end a *blocking clause* [23] is added after calculating a satisfying assignment. This clause excludes the solution from the

remaining search space. When the solver continues the search, only new solutions will be found.

Example 5 *The FBDD construction of Example 4 continues. To block the satisfying assignment* $\alpha = (x_1 = 0, x_2 = 1, x_3 = 1)$ *and to continue the search process, the blocking clause* $(x_1 + \overline{x}_2 + \overline{x}_3)$ *is added to the CNF. This causes a conflict and the solver backtracks to* $x_1 = 1$ *(f), which satisfies the first and the third clause. The next step is the assignment of* $x_4 = 0$ *(g) which leads to one satisfying (h) and one conflicting (i) assignment due to the second clause. The satisfying assignment will be blocked by* $(\overline{x}_1 + x_4 + \overline{x}_3)$ *and thus the solver backtracks to* x_4 *where the last solution is found (j). Because the whole search space has been explored, no further solutions can be found and the solver terminates. The FBDD has been completed.*

Thus, the construction of FBDDs can be achieved by modifying a SAT solver in two steps:

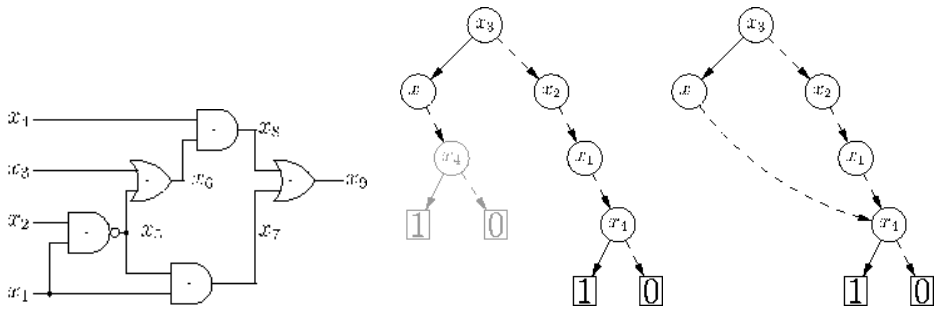
1. *Instruct the SAT solver to find all solutions instead of only a single one*
   That is, add a blocking clause whenever a solution is found such that the SAT solver will backtrack and continue the search.
2. *Apply the properties of Observation 1 such that an FBDD results*
   That is, construct a 1-path (0-path) for each satisfying (conflicting) assignment which is found by the SAT solver. Furthermore, construct a 0-path if the SAT solver performs an implication.

However, when the SAT solver terminates the resulting FBDD may still include isomorphisms, i.e. parts of the decision diagram are identical and can be merged. Approaches to identify such isomorphisms are described in the next section.

4 Identify Isomorphisms

FBDDs created by the proposed approach may include isomorphisms. Merging isomorphic sub-graphs often reduces the size of a decision diagram significantly.

Example 6 *An FBDD representing the circuit given in Figure 4(a) is constructed. As shown in Figure 4(b) parts of the search space are already traversed and the corresponding paths are created, respectively. By assigning* $x = 1$ *the SAT solver enters into an already traversed part of the search space, i.e. an isomorphic sub-graph is built. Identifying this isomorphism leads to an FBDD as shown in Figure 4(c). Thus, the size of the FBDD is reduced by one node.*



(a) (b) (c)

Fig. 4. Identifying isomorphism during FBDD construction

During OBDD construction the reduction is applied efficiently by using a *unique table*, i.e. a hash table, which stores all nodes in a unique way. Due to the same order of variables along each path an isomorphic sub-graph can be detected immediately. Because FBDDs may have different orders of variables, other techniques are needed to detect isomorphic structures while creating the decision diagram.

In this work we compared two techniques to identify isomorphisms: Cutlines (previously applied to reachability analysis [12, 13]) and cutsets (previously applied to OBDD construction [9]).

4.1 Cutlines

Cutlines work on CNFs derived from a circuit (see Section 2.4) and with the restriction that only primary inputs are assigned by the SAT solver's decision heuristic. The assignment of the internal signals is implied. Already traversed parts of search space can be detected by comparing internal signals of the circuit.

Definition 2 *Let C be a circuit and the primary inputs of C are partially assigned. Then, the assignments of some of the internal signals of C may be implied. The set of internal signals, which form a border to the unassigned signals of the circuit, defines the* cutline*.*

An example illustrates the idea in more detail:

Example 7 *Again the circuit given in Figure 4(a) is considered. Two different assignments to the primary inputs of this circuit are given:* $\alpha_1 = (x_1 = 0, x_2 = 0)$ *and* $\alpha_2 = (x_1 = 0, x_3 = 1)$*. Both assignments lead to* $x_5 = 1$*,* $x_6 = 1$ *and* $x_7 = 0$*. This forms a border between the assigned and unassigned signals of the circuit as depicted in Figure 5 – the cutline. Since both assignments* α_1 *and* α_2 *result in the same state, the remaining search space is equivalent.*

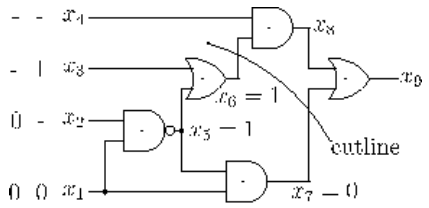


Fig. 5. Cutline

Thus, cutlines can be used for identifying isomorphic sub-graphs as follows: After a part of the search space is completely traversed the current cutline is determined. This is done by traversing the circuit (i.e. the CNF representation) from the primary outputs to the primary inputs. The traversing stops when assigned internal signals are reached. These signals (x_6 and x_7 in Example 7) are stored and form the cutline. The resulting cutline is – supplemented by a pointer to the respective sub-graph – inserted into a hash table. When the SAT solver enters a new part of the search space, a cutline is determined and synchronized with the hash table. If the respective cutline already exists, an isomorphic sub-graph is identified and the stored pointer is used to update the FBDD.

Since cutlines consist of assigned variables only, they can easily be stored by clauses (similar to blocking clauses). If such a clause causes a conflict, the solver does not have to add a 0-leaf to the FBDD but the respective sub-graph, whose pointer is associated with the clause. Thus, the hash table is replaced by such 'cutline'-clauses. This leads to an improved access to the cutlines since e.g. *Boolean constraint propagation* [20] can be exploited using clauses instead of a hash table.

However, to apply cutlines in general, the search process of the SAT solver has to be restricted in such a way that implications are only allowed from the primary inputs to the output. In comparison: When no cutlines are used, the output can be assigned to one (as described in Section 2.4). This may lead to implications from the outputs to the inputs, which decrease the size of the search space and might result in better FBDDs. The experiments in Section 5 show the effect of this restriction.

4.2 Cutsets

Cutsets directly work on the CNF formula. An already traversed part of the search space is stored by a subset of clauses in combination with a set of assigned variables. Cutsets are defined as follows:

Definition 3 *Let* $f : \mathbb{B}^n \to \mathbb{B}$ *be a Boolean function in CNF over variables* $X =$

$\{x_1, x_2, \ldots, x_n\}$ and $S_\alpha \subset X$ is a set containing all variables $x_i \in X$, which are assigned by the SAT solver. Then, for a given S_α the CNF can be separated into three subsets:

1. *Clauses, containing assigned literals only,*
2. *clauses, containing unassigned literals only and*
3. *clauses containing at least one assigned and one unassigned literal, respectively.*

The third subset is defined as the cutset.

The general idea behind cutsets is the following: All clauses in the first subset can be ignored when identifying isomorphic sub-graphs, since they are either satisfied or conflicting. In the former case, the SAT solver backtracks and continues the search in another part of the search space. In the latter case the respective clauses do not affect the remaining search space. Furthermore, clauses in the second subset can be ignored, since they have not been altered because none of their variables have been set. Thus, the third set – the cutset – is sufficient to identify an isomorphic subset.

To apply this for identifying isomorphisms, both the set S_α of all assigned variables and the cutset have to be determined and synchronized if the SAT solver enters a new part of the search space. Both can easily be obtained, i.e. in contrast to cutlines no complex modifications or any restrictions to the CNF are necessary.

However, storing S_α and cutsets by clauses is not as easy as it is for cutlines. Thus, using a hash table is the better choice here. Furthermore – if FBDDs are built – cutsets do not ensure that *all* isomorphisms will be found as the following example shows.

Example 8 *Once again the circuit given in Figure 4(a) is considered. As described in Example 7 the assignments $\alpha_1 = (x_1 = 0, x_2 = 0)$ and $\alpha_2 = (x_1 = 0, x_3 = 1)$ lead to an isomorphism. Since the set S_α differs for both assignments ($S_{\alpha_1} = \{x_1, x_2, x_5, x_6, x_7\} \neq S_{\alpha_2} = \{x_1, x_3, x_5, x_6, x_7\}$), the isomorphism cannot be determined by using cutsets.*

However, the experiments in the next section show that in spite of these negative effects, better results can be achieved when using cutsets instead of cutlines.

5 Experimental Results

This section provides experimental results for the construction of FBDDs with a SAT solver. The proposed approaches were implemented in C++ on top of the SAT

solver MiniSat [21]. Instances of the LG-Synth93 package are used as benchmarks. Here, all *m*-output functions were transformed into their characteristic functions, which has a single output. All experiments have been carried out on an AMD Athlon 3500+ with 1 GB of main memory. The timeout was set to 400 CPU seconds.

The results are presented by bar charts. The x-axis refers to the benchmark, while the y-axis refers to the size of the resulting decision diagram (in number of nodes). Note the logarithmic scale of the y-axis. Aborted benchmarks are indicated by bars exceeding the y-axis.

5.1 Construction of FBDDs

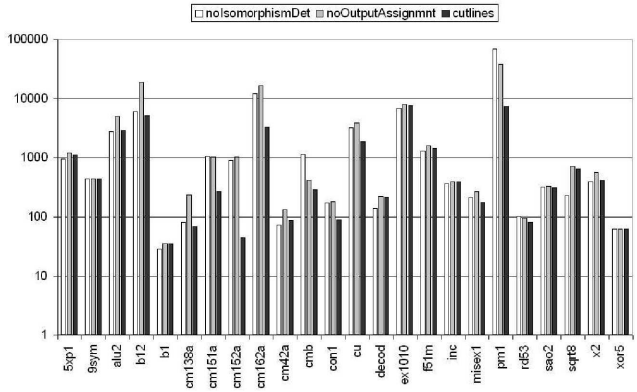


Fig. 6. Effect of using cutlines

First, the effect of using cutlines is discussed. The results are given in Figure 6. Bars denoted by *noIsomorphismDet* report the results of the approach without detection of isomorphisms, *noOutputAssignmnt* denotes the results of the same approach, but without the use of unit clauses which assign the primary output (i.e. with the restrictions necessary for cutlines), and *cutlines* denotes the approach which identifies isomorphisms with the help of cutlines.

Omitting unit clauses (i.e. applying the restriction for cutlines) leads to significant larger FBDDs in most cases (comparison of *noIsomorphismDet* to *noOutputAssignmnt*). This negative effect can be partially compensated by the detection of isomorphisms (comparison of *noIsomorphismDet* to *cutlines*): This reduces the size of many FBDDs (e.g. at *cm162a* and *pm1*). However, there are still FBDDs which are larger than the ones constructed by the approach without detection of isomorphisms (e.g. *decod* and *sqrt8*).

In contrast this cannot happen when using cutsets. Here, no restrictions are

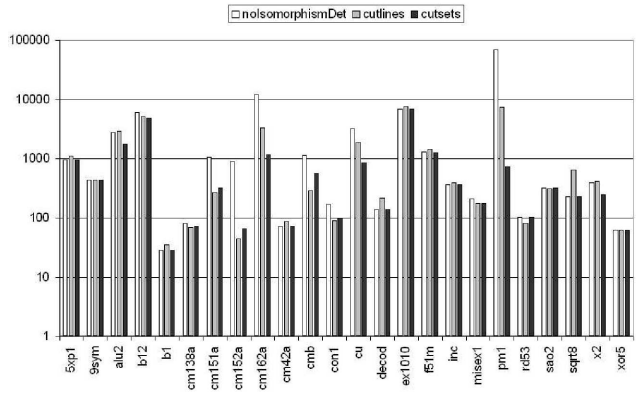


Fig. 7. Identification of isomorphisms

applied to the SAT solver. Thus, in comparison to the the approach without detection of isomorphisms all resulting FBDDs have the same size (if no isomorphism was detected) or are smaller (if at least one isomorphism was detected). This is approved by the results shown in Figure 7 (bars denoted by *cutsets* report the results of the cutset approach). Overall with the help of cutlines smaller FBDDs can be constructed only in six of 24 cases. In contrast, for all remaining benchmarks smaller FBDDs result by exploting cutsets.

5.2 Comparison to OBDDs

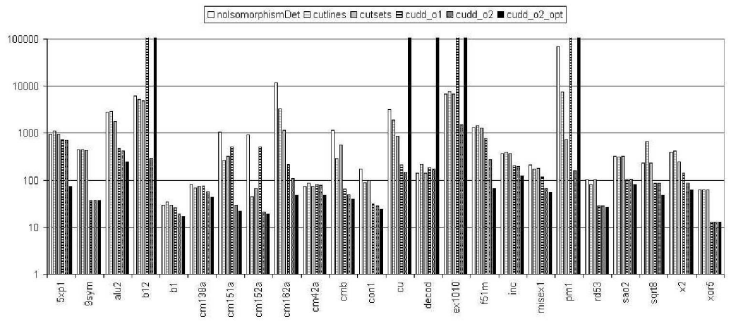


Fig. 8. Comparison to OBDDs

In this section the FBDDs obtained by the proposed approaches are compared to OBDDs obtained with *CUDD* [24]. Two orderings are used: The first (denoted by *cudd_1*) orders all variables according to their index $(x_1, x_2, \ldots, x_n)$. The second ordering (denoted by *cudd_2*) is derived by a depth-first search from the outputs to

the inputs of the respective circuit. Additionally the minimal OBDDs size for each benchmark is given (denoted by *cudd_opt*). The results are shown in Figure 8.

For some benchmarks the SAT-based approach is able to find smaller decision diagrams than CUDD using the first ordering (e.g. *cm151a*). Moreover, for some other benchmarks using CUDD and the first ordering no OBDD is found within the given time limit (i.e. *b12*, *ex1010* and *pm1*). For *decod* the smallest decision diagram is found by the SAT-based approach only. Here, the minimal OBDD cannot be constructed within the given time limit.

However, apart from that it is clear to see, that most of the FBDDs built by the SAT-based approach are larger than the respective OBDDs built by CUDD.

6 Conclusion

In this work we described FBDD construction with the help of SAT solvers. Therefore, two steps have to be performed: (1) instruct the SAT solver to find all solutions instead of only a single one and (2) construct respective paths for each satisfying and conflicting assignment as well as for each implication. For identifying isomorphic sub-graphs, cutlines and cutsets are used and their advantages and disadvantages are discussed. The resulting conclusions were approved by experiments. However, in comparison to OBDDs for most of the benchmarks the resulting FBDDs are still larger than the respective OBDD representations.

References

[1] R. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. on Comp.*, vol. 35, no. 8, pp. 677–691, 1986.

[2] R. Drechsler and B. Becker, "Overview of decision diagrams," *IEE Proceedings*, vol. 144, pp. 187–193, 1997.

[3] J. Gergov and C. Meinel, "Efficient analysis and manipulation of OBDDs can be extended to FBDDs," *IEEE Trans. on Comp.*, vol. 43, pp. 1197–1209, 1994.

[4] R. Bryant, "On the complexity of VLSI implementations and graph representations of Boolean functions with application to integer multiplication," *IEEE Trans. on Comp.*, vol. 40, pp. 205–213, 1991.

[5] W. Günther and R. Drechsler, "Minimization of free BDDs," *ASP Design Automation Conf.*, pp. 323–326, 1999.

[6] K. Takagi, H. Hatakeda, S. Kimura, and K. Watanabe, "Exact mimimization of free BDDs and its application to pass-transistor logic optimization," *IEICE Trans. Fundamentals*, vol. E82-A, no. 11, pp. 2407–2413, 1999.

[7] J. Bern, C. Meinel, and A. Slobodová, "Some heuristics for generating tree-like FBDD types," *IEEE Trans. on CAD*, vol. 15, pp. 127–130, 1996.

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

[9] J. Huang and A. Darwiche, "Using DPLL for efficient OBDD construction," *Proc. 7th Int. Conf. on Theory and Applications of Satisfiability Testing*, pp. 157–172, 2004.

[10] R. Wille, "Erstellung von Free Binary Decision Diagrams mit SAT-Beweisern," Master's thesis, Universität Bremen, Bremen, Nov. 2006.

[11] R. Wille, G. Fey, and R. Drechsler, "Building free binary decision diagrams using sat solvers," *8th Workshop on Applications of the Reed-Muller Expansion in Circuit Design and Representations and Methodology of Future Computing Technology*, 2007.

[12] S. Sheng and M. Hsiao, "Efficient Preimage Computation Using A Novel Success-Driven ATPG," *Design, Automation and Test in Europe*, pp. 822–827, Mar. 2003.

[13] B. Li, M. Hsiao, and S. Sheng, "A novel SAT all-solutions solver for efficient preimage computation," *Design, Automation and Test in Europe*, pp. 10 272–10 278, 2004.

[14] B. Bollig, P. Savicky, and I. Wegener, "On the improvement of variable orderings for OBDDs," *IFIP Workshop on Logic and Architecture Synthesis, Grenoble*, pp. 71–80, 1994.

[15] S. Cook, "The complexity of theorem proving procedures," *3. ACM Symposium on Theory of Computing*, pp. 151–158, 1971.

[16] T. Larrabee, "Test pattern generation using Boolean satisfiability," *IEEE Trans. on CAD*, vol. 11, no. 1, pp. 4–15, Jan. 1992.

[17] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu, "Symbolic model checking without BDDs," *Tools and Algorithms for the Construction and Analysis of Systems*, vol. 1579, pp. 193–207, 1999.

[18] M. Davis, G. Logeman, and D. Loveland, "A machine program for theorem proving," *Comm. of the ACM*, vol. 5, pp. 394–397, 1962.

[19] J. Marques-Silva and K. Sakallah, "GRASP: A search algorithm for propositional satisfiability," *IEEE Trans. on Comp.*, vol. 48, no. 5, pp. 506–521, 1999.

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

[21] N. Eén and N. Sörensson, "An extensible SAT solver," *SAT 2003*, vol. 2919, pp. 502–518, 2004.

[22] G. Tseitin, "On the complexity of derivation in propositional calculus," *Studies in Constructive Mathematics and Mathematical Logic, Part 2*, pp. 115–125, 1968.

[23] K. McMillan, "Applying SAT methods in unbounded symbolic model checking," *Computer Aided Verification*, pp. 250–264, 2002.

[24] F. Somenzi, *CUDD: CU Decision Diagram Package Release 2.3.1*. University of Colorado at Boulder, 2001.