On the Relation between SAT and BDDs for Equivalence Checking

Sherief Reda[1] Rolf Drechsler[2] Alex Orailoglu[1]

[1]Computer Science & Engineering Department
University of California, San Diego
La Jolla, CA, 92093

[2]Institute of Computer Science
University of Bremen
28359 Bremen, Germany

Abstract

State-of-the-art verification tools are based on efficient operations on Boolean formulas. Traditional manipulation techniques are based on Binary Decision Diagrams (BDDs) and SAT solvers. In this paper, we study the relation between the two procedures and show how the number of backtracks obtained in the Davis-Putnam (DP) procedure is linked to the number of paths in the BDD. We utilize this relation to devise a method that uses BDD variable ordering techniques to run the DP procedure. Experimental results confirm that the proposed method results in a dramatic decrease in the number of backtracks and in the time needed to prove the Boolean satisfiability problem as well.

1 Introduction

Boolean Satisfiability (SAT) has received increased attention as a promising technique for Automatic Test Pattern Generation (ATPG) and equivalence checking [1], [2]. Virtually all SAT techniques rely on the use of the Davis-Putnam procedure (DP) [3], [4] to explore the search tree. If there is a pattern that differentiates the circuits under verification, then DP will eventually find it or prove that the SAT formula is unsatisfiable. Numerous techniques have been proposed to reduce the search tree. Some of these techniques such as iterated global implications [2] and recursive learning [5] are applied as a preprocessing step, while others are applied during the course of the application of the DP procedure. For example, clause recording [6] and cache-based backtracking [7] are used to avoid conflicts. An alternative research trend focuses on identifying variable orderings techniques that minimize the number of backtracks executed by the DP procedure to find a satisfying assignment[1]. The effectiveness of these techniques is sharply limited though in case the formula is unsatisfiable.

On the other hand, Binary Decision Diagrams (BDDs), as introduced in [9], have been traditionally used to solve the equivalence checking problem due to their canonical property. However, it is this requirement for canonicity that makes BDDs inefficient in representing certain classes of functions. For example, integer multipliers have displayed exponential memory requirements for any variable ordering [10].

[1] An excellent survey can be found in [8].

There has been increased interest in techniques that integrate SAT and BDDs to reduce the time and space needed to solve the equivalence checking problem [11], [12]. Though it has been noted that SAT and BDDs represent the same entity [7], there is little understanding of the relation between the two procedures and how the techniques of one domain can be utilized in the other.

In this paper we attempt to provide an improved understanding of the relation between these two procedures. We view the equivalence checking problem as a search in the decision trees of the two circuits for a path that leads to the terminal 1(0) in one but leads to the 0(1) terminal in the other. From this perspective, it would be desirable to decrease the number of paths, thus reducing the number of backtracks and time needed to solve the problem. We propose a dynamic variable ordering technique to run the DP procedure. The technique is geared towards minimization of the number of backtracks needed to prove the *unsatisfiability* of the CNF formula that results from proving the equivalence of two circuits. We compare our method with the greedy variable ordering of TEGUS [2]. Experimental results verify that the proposed approach results in a dramatic decrease in the number of backtracks and time needed to solve the problem.

The organization of the paper is as follows. Section 2 introduces the necessary preliminaries for the material in this paper. Section 3 presents the theoretical foundations for the relation between the DP procedure and BDDs. Section 4 proposes a dynamic BDD-based variable ordering technique for the DP procedure. Experimental results are given in Section 5, followed up by conclusions in Section 6.

2 Preliminaries

2.1 Binary Decision Diagrams

A Binary Decision Diagram (BDD) is a rooted directed acyclic graph with two terminal nodes that are referred to as the 0-terminal and the 1-terminal. Every non-terminal node is associated with a primary input variable such that it has two outgoing edges called the 0-edge corresponding to assigning the variable a *false* truth value, and the 1-edge corresponding to assigning the variable a *true* truth value.

An Ordered Binary Decision Diagram (OBDD) is a BDD such that the input variables appear in a fixed order on all the paths of the graph, and no variable appears more than

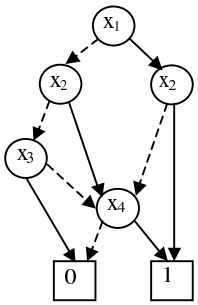

Fig 1: A ROBDD example

once in the path. A Reduced Ordered BDD (ROBDD) is an OBDD that results from the repeated application of the following two rules:

1. Eliminate all redundant nodes whose edges point to the same node.
2. Share all equivalent sub-graphs.

Figure 1 illustrates an example of an ROBDD where the solid edges denote the 1-edges and the dashed edges denote the 0-edges. In the following treatment, only reduced ordered BDDs are considered and for short we denote them as BDDs.

2.2 The Davis-Putnam Procedure

A CNF formula φ is a set of clauses where each clause is the disjunction of a number of literals where a literal is a variable or its negation. Since each logic gate can be represented by a number of clauses [1], a CNF formula of a logic circuit is the conjunction of the CNF formulas of all the gates. A CNF formula is satisfiable if at least one set of assignments to the variables of the formula makes it evaluate to *true*.

Virtually all SAT solvers use the Davis-Putnam (DP) procedure in their core [3], [4] in order to find a satisfying assignment for the formula or conversely to prove the formula unsatisfiable. The DP procedure performs a backtracking depth-first search in the space of all truth assignments to find a satisfying assignment for the CNF formula. The performance of backtracking is greatly improved by employing unit clause propagation: whenever a unit clause arises, the variable occurring in that clause is assigned the truth-value that satisfies the clause. The formula is thereupon simplified which may lead to new unit clauses. Figure 2 outlines the DP procedure. The procedure returns *true* in case the CNF formula is satisfiable, *false* otherwise.

```
assign(sat_formula φ, literal v)
begin
  assign v = true in φ and simplify φ;
  apply unit clause propagation;
  if φ has an empty clause then return false
  else return true;
end

DP(sat_formula φ)
begin
  choose literal v to split on;
  if v = NULL then return true;
  if assign(φ, v) then
     if DP(φ) then return true;
  undo v assignment;
  if assign(φ, ¬v) then
     if DP(φ) then return true;
  return false;
end
```

Fig 2: The DP procedure

3 On the relation between the DP procedure and BDDs

In this section we study the relation between the DP procedure and the BDD representation of the same circuit. We first formalize the various properties of the CNF formulas generated from multi-level combinational logic circuits and show how these properties allow a measure of flexibility in the search for a satisfying assignment to the formula. Furthermore, we prove the relation between the number of backtracks obtained using the DP procedure and the number of paths in the corresponding BDD. This relation allows the calculation of optimal lower bounds for the number of backtracks needed to prove the equivalence of two equivalent circuits. We start by introducing some notation to provide a concise basis for the formalization of the derived results.

Let φ be a CNF formula, and $V(\varphi)$ denote the set of variables that φ depends on. A clause $c_i \in \varphi$ is satisfied if there is some assignment to its literals such that the disjunction of the literals evaluates to *true*. The CNF formula φ is satisfied if there is a truth assignment to $V(\varphi)$ such that every clause $c_i \in \varphi$ is satisfied.

Let C-CNF denote the set of CNF formulas generated from multi-level combinational logic circuits. Let $C \in$ C-CNF be a multi-level combinational circuit, and let f be the underlying Boolean function of C. Since the logic value of all the internal and output signals of C can be derived from the values of the primary inputs, we consider the support of f to be the primary inputs only. We denote the set of primary inputs of φ by $P(\varphi)$. In addition, we introduce the variable χ to reference the primary output of the circuit. The relation between φ, $P(\varphi)$ and $V(\varphi)$ is given by the next lemma.

Lemma 1 If $\varphi \in$ C-CNF, then it is possible to find a set of variables $P(\varphi) \subset V(\varphi)$ such that φ can be satisfied by only splitting on the variables of $P(\varphi)$ in the DP procedure.

Proof Since the internal nodes of a multi-level logic circuit are functions of the primary inputs, it is possible to determine their assignment given some assignment on the set of primary input variables, $P(\varphi)$. ■

Lemma 1 enables a reduction of the search space from one in terms of all the variables of the circuit nodes to one in terms of the primary inputs. This reduces the backtracking in the DP procedure to be only in terms of the primary inputs. In a typical run of the DP procedure, a sequence of support variables will be assigned truth variables during

the search of a satisfying assignment to the whole CNF formula. This subset of currently assigned primary inputs shall be denoted by S, and their truth assignments by A_s. The restricted Boolean function that results from the application of A_s to φ shall be denoted by f_{As}.

In order to avoid fruitless searches, the DP procedure should avoid assigning truth-values to variables that can make no contribution to the satisfiability of the formula. This notion is captured by the following definition.

Definition 1 If A_s is the truth assignment of a set $S \subseteq P(\varphi)$ and $v \in P(\varphi)$ but $v \notin S$ then v is said to be redundant under A_s if $\partial f_{As} / \partial v = 0$.

Definition 1 simply states that if the restricted Boolean function f_{As} is insensitive to the change in v, then there is no point in assigning v a truth value. This notion allows us to tailor down the satisfiability of C-CNF formulas as given by the following theorem.

Theorem 1 A CNF formula $\varphi \in$ C-CNF is satisfied under a truth assignment A_s of a set $S \subseteq P(\varphi)$ if $\forall v \in (P(\varphi)\text{-}S)$: $\partial f_{As} / \partial v = 0$.

Proof If $\forall v \in (P(\varphi)\text{-}S)$: $\partial f(A_s)/\partial v = 0$, then no truth assignment to any element that belongs to $(P(\varphi)\text{-}S)$ can change the function $f(A_s)$. Since all variables in S are assigned truth values under A_s and $(P(\varphi)\text{-}S) \cup S = P(\varphi)$, then it follows from Lemma 1 that φ is satisfied. ■

Theorem 1 proves that if all the remaining unassigned primary inputs are redundant, then there is no need for additional variable assignments since the C-CNF formula is satisfiable. Another important property of C-CNF formulas is that there is no point in assigning additional variables if the primary output has been already assigned a truth value. This is proved in the next theorem.

Theorem 2 If $\varphi \in$ C-CNF and χ is assigned a truth value under a truth assignment A_s of a set $S \subseteq P(\varphi)$, then φ is satisfied.

Proof If χ is assigned a truth value, then $\forall v \in (P(\varphi)\text{-}S)$: $\partial f(A_s)/\partial v = 0$. This is true since none of the variables that belongs to $(P(\varphi)\text{-}S)$ can change the primary output χ corresponding to f. Thus, it follows from Theorem 1 that φ is satisfied. ■

Theorem 2 defines our notion of satisfiability for CSAT. In addition, it opens the possibility for the primary output variable χ to be assigned a truth-value while there exist clauses that are not evaluated to *true* and have some remaining unassigned literals. Example 1 illustrates such a case.

Example 1 The CNF formula of the circuit in Figure 3 is $\varphi = (\neg a + d) \cdot (\neg b + d) \cdot (a + b + \neg d) \cdot (c + \neg z) \cdot (d + \neg z) \cdot (\neg d + \neg c + z)$. In this example, $P(\varphi) = \{a, b, c\}$, $\chi = z$ and under the partial assignment $A_s = \{c = 0\}$, where $S = \{c\}$, z is assigned the truth value *false* and $\varphi = (\neg a + d) \cdot (\neg b + d) \cdot (a + b + \neg d)$. We notice that under A_s both $\partial f_{As}/\partial a$ and $\partial f_{As}/\partial b$ equal zero, since the function output value is already determined. Furthermore, there exists a truth assignment to the set of variables $P(\varphi)\text{-}S$ that satisfies the remaining clauses. This truth assignment is however of no interest since the primary output has already been assigned a truth-value. As an illustration for Definition 1, if $A_s = \{a = 1\}$ where $S = \{a\}$ then $\varphi = (c + \neg z) \cdot (\neg c + z)$ and $\partial f_{As}/\partial b = 0$, yet z is not assigned a truth value. Thus, φ is not satisfied but b is redundant under A_s.

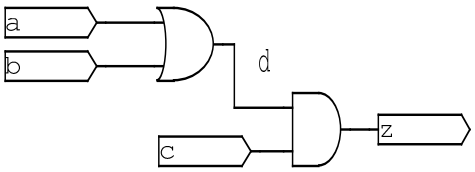

Fig 3: Circuit for example 1

We notice that Rule 1 in Section 2.1 for BDD reduction is equivalent to Definition 1 in the CSAT context and thus tracing a path from the root to the 1-terminal or 0-terminal in the BDD of circuit C is equivalent to finding a satisfying assignment to the CNF formula φ of C using the same variable ordering of the path in the BDD. This notion is captured in the following lemma.

Lemma 2 Given a BDD β and a CSAT formula φ for some logic circuit C, then under a variable ordering π and a truth assignment A_π on a certain path of β to the terminal, φ is satisfiable using the same variable ordering and truth assignment.

Proof Assume the set of primary inputs is P. Then we can partition P into two sets, S and T, such that S contains all the variables that appear in π and T contains all the remaining variables. By Rule 1 of section 2.1, T is the set of all redundant nodes. But due to the equivalence of Rule 1 and Definition 1, it follows from Theorem 1 that φ is satisfied. ■

From this perspective, the equivalence checking problem between two circuits can be viewed as a search in the decision trees of the two circuits for a path that leads to the terminal 1(0) in one but leads to the 0(1) terminal in the other. This view allows the introduction of the relation between BDDs and the DP procedure as given by the following theorem.

Theorem 3 Given a BDD β with a number of paths P and a C-CNF formula φ for some logic circuit C then if the variable ordering strategy of the DP procedure follows the same ordering for every path of β, then DP proves the equivalence of C against an equivalent version in P-1 backtracks.

Proof In order to prove the equivalence of two circuits, we have to check that every assignment that leads to 1(0) terminal in the BDD of one circuit leads to the same result in the other circuit. Consequently, if the variable ordering

for splitting in the DP procedure is the same as in the corresponding BDD, then considering an alternative path in the BDD leads to a backtrack in the DP procedure. Thus, the number of paths exceeds the total number of backtracks exactly by 1. ■

We now utilize Theorem 3 to calculate optimal lower bounds on the number of backtracks that can be obtained from the DP procedure using the primary inputs for backtracking.

Searching among the $N!$ different variable orderings for a N variable BDD helps identify the variable ordering(s) that produces the minimal number of paths. This minimal number of paths allows the calculation of a lower bound on the number of backtracks in the DP procedure as given by the following theorem.

Theorem 4 Given a DP procedure that operates using Theorem 1 and 2, then the optimal number of backtracks needed to prove the equivalence of two equivalent circuits is bounded by the number of paths in the corresponding minimal path BDD.

Proof Since the number of paths and backtracks are linked by Theorem 3 and there exists a variable ordering that minimizes the number of paths, then the optimal number of backtracks using the DP procedure can be obtained if we follow the same variable ordering for every path in the minimal path BDD. ■

The importance of Theorem 4 is that it allows benchmarking any variable ordering strategy for the DP procedure against an optimal lower bound. We now develop a variable ordering heuristic for the DP procedure that tries to trace the same variable ordering for every path in the corresponding minimal path BDD of the circuit.

4 Dynamic variable ordering strategy for the DP procedure

From the previous section, we conclude that the variable ordering strategy should differ for every path of the decision tree, and furthermore result in no splitting on a redundant variable under the current partial assignment. In this section, we propose a structural method that avoids redundant splitting and tries to make the fewest possible splittings to satisfy φ. The method is based on the following theorem.

Theorem 5 If a bounded gate lies on every path from the primary output χ to the unassigned primary input $v \in (P(\varphi)\text{-}S)$ under a current partial assignment A_s for a set $S \subset P(\varphi)$, then $\partial f_{A_s}/\partial v = 0$.

Proof The existence of a bounded gate (a gate with a specified output) in every path to χ implies that no value assigned to v can affect the function of the circuit since these bounded gates will suppress the propagation of the logic value. Thus, since $f(A_s)$ remains unchanged under any assignment to v, $\partial f_{A_s}/\partial v = 0$.■

In order to utilize Theorem 5, we proposed a DP variable ordering strategy that is a modification of a BDD variable ordering heuristic [13]. The method starts by assigning a weight of 1.0 to the primary output, and continues by propagating this weight to the primary inputs in the following manner: divide the output weight of each gate among its inputs, accumulating the weight of the fan-out branches into the fan-out stem. Next, the primary input with the highest weight is chosen as the next variable to be split in the DP procedure. By unit clause propagation, we exclude all the gate outputs of the bounded literals from the next cycle of weight calculations.

The proposed weight assignment method assigns a weight of zero to every redundant primary input under the current partial assignment. Furthermore, the strategy splits on the largest weight input in an effort to obtain a minimal number of splittings for satisfying the CSAT formula. We illustrate the weight calculation procedure by the following example.

Example 2 Suppose we are proving the circuit in Figure 4.a against an equivalent version of it. In this case we have to compare if their BDDs are isomorphic or using the DP

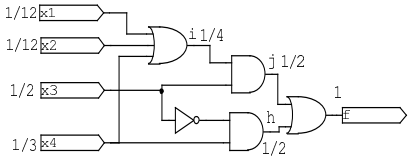

Fig 4a: Initial weights.

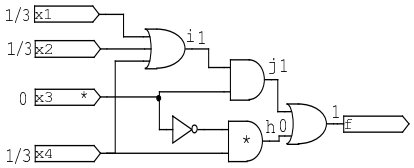

Fig 4b: Weights after x3 equals 1.

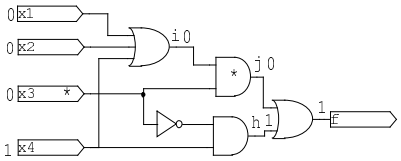

Fig 4c: Weights after x3 equals 0.

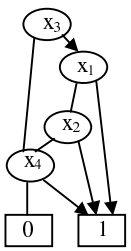

Figure 4d: Minimal path BDD.

Circuit	Output Name	TEGUS Greedy Search		Proposed Approach		Difference		Minimal Path BDD
		Backtracks	Time (seconds)	Backtracks	Time (seconds)	Backtracks	Time	#Paths
c0432	370gat	Abort	771.00	943200	185.00	95.3%	76.01%	894606
	432gat	Abort	770.00	816012	235.00	95.9%	69.48%	798906
	329gat	1646605	22.50	81912	5.94	95.0%	73.60%	57513
c0499	od1	Abort	1686.00	Abort	10351.00	-	-	20000000
	od18	Abort	1685.00	Abort	10245.00	-	-	23000000
	od31	Abort	1691.00	Abort	10627.00	-	-	23000000
c0880	864	649741	12.30	164656	27.50	74.7%	-123.58%	89937
	850	37225	0.60	24171	3.45	35.1%	-475.00%	21981
	874	Abort	498.00	1083156	278.00	94.6%	44.18%	532793
c1355	1353	Abort	1921.00	Abort	8501.00	-	-	21000000
	1354	Abort	1892.00	Abort	8427.00	-	-	23000000
	1355	Abort	1984.00	Abort	8602.00	-	-	23000000
c1908	57	Abort	718.00	147457	11.90	99.3%	98.34%	134221
	60	Abort	1708.00	28161	20.10	99.9%	98.82%	26638
	66	Abort	2058.00	14249	10.40	99.9%	99.49%	12638
c3540	399	24550	0.73	4528	1.34	81.6%	-83.56%	1636
	384	6107715	652.00	70997	57.20	98.8%	91.23%	65535
	387	1582929	213.00	17312	17.00	98.9%	92.02%	16456
C5315	688	Abort	300.00	23974	13.00	99.9%	95.67%	13919
	843	Abort	3383.00	211181	162.00	98.9%	95.21%	103569
	818	85989	3.50	6537	2.18	92.4%	37.71%	2489
C7522	373	16	0.01	10	0.01	37.5%	0.00%	10
	376	40952	1.30	15416	4.93	62.4%	-279.23%	7152
	359	2474096	114.00	585584	285.00	76.3%	-150.00%	196592
c6288	2223	157	0.01	102	0.01	35.0%	0.00%	102
	3895	174315	12.20	106423	108.00	38.9%	-785.25%	109668
	4591	2794411	255.00	1705027	2689.00	39.0%	-954.51%	1793496

Table 1: Comparing Greedy search to BDD-based variable ordering for the DP procedure.

procedure, we have to check that every path results in the same output assignment in both circuits. Applying the previous procedure of weight calculations to minimize the number of paths being compared, we trace the different paths of the BDD of the circuit in figure 4.a in the manner shown below.

From the initial weights given in figure 4.a, we split on x_3 since it has the highest weight. Suppose x_3 is assigned the *true* value. Then by unit clause propagation, gate *h* becomes bounded to the truth value *false*. The weight calculation procedure is executed again but this time by setting the weights of bounded gates to *zero*. Figure 4.b illustrates the new weights. Next, x_1, x_2 or x_3 can be chosen to split on; the resultant search tree will always produce identical number of backtracks. Notice that bounded gates are marked with a '*' in the corresponding figure. After finishing this half of the decision tree, x_3 is flipped and assigned to *false*. The new calculated weights are shown in Figure 4.c. Since x_4 has the highest weight, we split on x_4 thus reaching the terminals.

We now construct the minimum path BDD corresponding to this circuit as shown in Figure 4.d. We notice in the BDD that the splitting choices made by the previous procedure traverse this BDD in exact ordering for each path, thus producing the minimal number of backtracks, 5, corresponding to the minimal number of paths, 6.

As was just illustrated, the proposed method has produced optimal results for the proposed example; however, the method is not in general optimal since it depends on the structure of the circuit. As shall be demonstrated in the next section, the proposed method is capable of achieving near optimal results.

5 Experimental Results

In this section, we present experimental results for our proposed approach. The experiments have been carried out on a PC with an Intel Pentium 233 Mhz processor and 64MB of physical memory. We have used TEGUS [2] as a SAT solver and the CUDD [14] decision diagram package.

In our experiments we consider the performance of the proposed approach to verify the equivalence of the ISCAS'85 benchmark circuits [15] against their non-redundant version. We benchmark our approach against the greedy search approach of TEGUS. Table 1 gives the results of such a comparison. Column 1 lists the circuit name. Column 2 provides the name of the output under verification. Columns 3 & 4 give the number of backtracks and time needed to prove the output using the TEGUS approach. Columns 5 & 6 give the number of backtracks and time needed to prove the output using the proposed approach. The possible "Abort" notation in columns 3 & 5 denotes reaching the 20 million backtrack threshold without reaching a successful resolution; the subsequent time columns in that case denote the time needed to reach this limit. Columns 7 & 8 report the percentage decrease in the number of backtracks and time, respectively. Column 9 provides the near minimal number of paths in the corresponding BDD. In obtaining the number of paths, we only consider a subspace of the possible variable orderings. We compare the number of paths that result from sifting [16] the corresponding BDD to a minimum size and picking the minimal path. For space considerations, we consider only three outputs from each circuit.

Comparing the greedy approach to the proposed variable ordering strategy, we observe that the proposed approach

results on the average in 90% decrease in the number of backtracks. The time needed to complete the backtracks varies from reduction in 13 cases with an average of 70% decrease to increases in about 7 cases with an average increase of about 4 times. While at first glance the time results look inconclusive, it can be easily noted that the increases are essentially associated with low-backtrack, provable outputs, since in these cases the recurring weight assignment costs cannot be amortized across the small number of backtracks. Our approach greatly improves the performance of large search tree cases, the main focus of practical interest. We also notice that within the limit of 20 million backtracks, TEGUS fails to complete on 7 outputs while the proposed approach proves them in a relatively low number of backtracks.

As proved in Section 3, the minimum number of paths that can be obtained from the corresponding BDD is a lower bound on the number of backtracks that can be obtained in the DP procedure. The number of paths provides an ability to benchmark various ordering techniques and also provides an insight on the performance of the DP procedure. For example, the huge number of paths in case of the c0499 & c1355 circuits (they are functionally equivalent) suggests that they are hard-to-prove outputs using the DP procedure. In addition, in circuits, like the c6288, where the variation of the number of paths with respect to the variable ordering is small, the dynamic weight assignment technique constitutes a time consuming step with respect to the overall time, as the number of backtracks varies slightly with respect to the variable splitting strategy[2].

6 Conclusions

In this paper we have studied the relation between the search tree of the DP procedure and the BDD of the corresponding function, establishing that the number of paths from the root node to the terminals of the BDD is directly related to the number of backtracks needed to prove the equivalence of two functionally equivalent circuits. This relation introduces the ability to calculate an optimal lower bound on the number of backtracks needed to prove the equivalence checking problem. In addition, this relation has led to the conclusion that the capture of the variable ordering of the minimal path BDD in the DP procedure implies a reduction in the number of backtracks needed to prove the problem. In order to exploit this relation, we have devised a variable ordering technique for the DP procedure that through experimental results has exhibited superior performance in terms of the number of backtracks and time needed to prove the equivalence problem. The relation between the two procedures offers novel ways of tight integration of different prover approaches in a verification tool.

[2] After numerous experiments, we concluded that there is no apparent relation between the number of nodes and paths in a given BDD.

References

[1] T. Larrabee, "*Test Pattern Generation using Boolean Satisfiability*," IEEE Trans. on Computer Aided Design, Vol. 11, No. 1, pp. 4-15, January 1992.

[2] P. R. Stephan, R. K. Brayton, and A. Sangiovanni-Vincentelli, "*Combinational Test Generation Using Satisfiability*," Technical Report UCB/ERL M92/112, Dept. of EECS, Univ. of California, Berkeley, Oct., 1992.

[3] M. Davis, G, Logemann, and D. Loveland, "*A Machine Program for Theorem Proving*," Communications of the ACM, (5):394-397, July 1962.

[4] M. Davis, and H. Putnam, "*A Computing Procedure for Quantifier Theory*," Communications of the ACM, (7):201-215, 1960.

[5] J. Silva, T. Glass, "*Combinational Equivalence Checking Using Boolean Satisfiability and Recursive Learning*," In Proc. of Design Automation, Test in Europe Conference, pp. 145-149, 1999.

[6] J. Silva, "*An Overview of Backtrack Search Satisfiability Algorithms*," in Fifth International Symposium on AI and Mathematics, January 1998.

[7] M. Prasad, P. Chong, K. Keutzer, "*Why is ATPG easy?*" In Proceedings of the 36th Design Automation Conference, pp. 22-28, 1999.

[8] J. Gu, P. W. Purdom, J. Franco, and B. W. Wah, in "*Satisfiability Problem: Theory and Applications*," DIMACS Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, pp. 19-152, 1997.

[9] R. E. Bryant, "*Graph Based Algorithms for Boolean Function Manipulation*," IEEE Trans, in Computers, Vol. C-35, No. 8, pp. 677-691, August 1986.

[10] R. E. Bryant, "*On the Complexity of VLSI Implementations and Graph Representations of Boolean Functions with Application to Integer Multiplication*," IEEE Transactions on Computers, Vol. 40, No. 2, pp. 205-213, Feb. 1991.

[11] J. Burch, V. Singhal, "*Tight Integration of Combinational Verification Methods*," ACM/IEEE Int. Conference on Computer-aided Design 1998, pp. 570-576, San Jose, CA, USA.

[12] S. Reda, A. Salem, "*Combinational Equivalence Checking using Binary Decision Diagrams and Boolean Satisfiability*," In Proc. of Design Automation, Test in Europe Conference, pp. 122-126, 2001.

[13] S. Minato, *Binary Decision Diagrams and applications for VLSI CAD*, Kluwer Academic Publishers, 1996.

[14] F. Somenzi, CUDD: Colorado University Decision Diagram Package. ftp://vlsi.colorado.edu/pub.

[15] F. Brglez and H. Fujiwara, "A *Neural Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortran*," In Intl. Symp. on Circuits and Systems, 663-698, June 1985.

[16] R. Rudell, "*Dynamic Variable Ordering for Ordered Binary Decision Diagrams*," In Proc. of Int. Conf. On Computer Aided Design, pp. 42-47, 1993