Efficient Solution of Systems of Boolean Equations

Scott Woods

School of Electrical & Computer Engineering
Georgia Institute of Technology
Atlanta, GA 30332-0250

Giorgio Casinovi

Info Tech & Telecommunications Lab
Georgia Tech Research Institute
Atlanta, GA 30332-0800

Abstract

This paper describes an algorithm for the efficient solution of large systems of Boolean equations. The algorithm exploits the fact that, in some cases, the composition operation of Boolean functions represented by BDD's can be performed in a very efficient manner. Thus, the algorithm tries to eliminate as many variables and equations as possible through function composition. When the system can no longer be reduced in this way, the elimination process is continued through the use of Shannon decomposition. Numerical results show that the performance of this algorithm is significantly superior to that of a previous algorithm proposed by the authors.

1 Introduction

Given a Boolean function $f(x_1, x_2, \ldots, x_n)$, the problem of determining whether there exist Boolean values for $x_1, x_2, \ldots, x_n$ such that *f* evaluates to a logic one (or a logic zero) is known as the *Boolean satisfiability problem.* This problem arises in a number of areas related to the synthesis, simulation and testing of digital networks, such as output encoding and state assignment of finite state machines [1], timing analysis and delay-fault test generation of combinational circuits [2], automatic test-pattern generation [3], and the determination of the initial state of a digital network containing feedback loops. In [4], the authors presented an algorithm to solve Boolean equations based on repeated applications of the Shannon decomposition. Its main features were its ability to take advantage of the sparsity of the system of equations to be solved, and the fact that it could determine whether the system of equations had no, one, or more than one solution.

This paper describes an improved version of that algorithm which is mathematically equivalent to it, but tries to exploit as much as possible the fact that the composition of Boolean functions represented by BDD's can, in some particular cases, be performed in a very efficient manner. Section 2 contains a description of the algorithm and some theoretical results related to it, while Section 3 describes a number of variable numbering schemes that have been tried by the authors to improve the algorithm's efficiency. Finally, some experimental results are shown and discussed in Section 4.

This work was supported in part by the National Science Foundation under grant MIP-9211163.

2 Solving Boolean equations

Throughout this section, it will be assumed that the system of equations to be solved arises from the problem of finding the initial state of a digital network described at the gate level. As will become apparent later, the algorithm that computes the solution of this system of equations relies on the network's signal flow graph. It should be noted, however, that the dependency graph of the system of equations is isomorphic to the network's signal flow graph. Therefore, the algorithm described here can be applied to any system of Boolean equations: all that is necessary to do is to replace the network's signal flow graph with the dependency graph of the system to be solved.

In order to reduce the size of the system to be solved, it is advantageous to decompose the network into its *strongly connected components.* Since it is assumed that all the gates are unidirectional, the gate-level circuit can be represented as a directed graph. An algorithm that identifies strongly connected components in a directed graph was developed by Tarjan [5]. The complexity of this algorithm is $O(n+e)$, where *e* is the number of graph edges, and *n* the number of graph vertices (i.e. the number of nodes in the network). A side benefit of using this algorithm is that it sorts the graph vertices in reverse topological order. Traversing a graph in this order ensures that each vertex is visited only after all of its predecessors have already been visited: in the case of a digital network, this corresponds to propagating signal values through the network following the signal flow graph.

Once the graph has been sorted and its strongly connected components identified by Tarjan's algorithm, signal values can be propagated through the network, starting from the primary inputs. For this purpose, a strongly connected component is regarded as one gate, and when all inputs to the component are known a special routine is called to

ICCAD '96
1063-6757/96 $5.00 © 1996 IEEE

compute the values of the nodes inside the component: as is shown below, this requires the solution of a set of Boolean equations.

Let $x_1, x_2, \ldots, x_n$ denote the logic values of the nodes in a digital circuit. For each gate in the circuit, a corresponding Boolean equation can be written that expresses the value of the gate's output as a function of its input values. Such equations have the form:

$$x_i = g_i(x_1, \ldots, x_n), \qquad i = 1, \ldots, n, \quad (1)$$

where x_i denotes the value of the output of the *i*-th gate, and g_i is the Boolean function implemented by the gate. By the laws of Boolean algebra, the system of equations (1) is equivalent to:

$$f_i(x_1, \ldots, x_n) = x_i \overline{g_i} \cup \overline{x}_i g_i = 0, i = 1, \ldots, n. \quad (2)$$

This system, in turn, can be reduced to the single equation:

$$F^1(x_1, \ldots, x_n) = \bigcup_{i=1}^{n} f_i(x_1, \ldots, x_n) = 0. \quad (3)$$

The simplest way to solve eqn. (3) is to express F^1 in terms of its Shannon decomposition with respect to x_1, and to rewrite the equation as:

$$x_1 F^1_{x_1}(x_2, \ldots, x_n) \cup \overline{x}_1 F^1_{\overline{x}_1}(x_2, \ldots, x_n) = 0.$$

It can be shown [6, p. 58] that the values of x_1 that solve this equation are exactly those that satisfy the inequalities:

$$F^1_{\overline{x}_1}(x_2, \ldots, x_n) \leq x_1 \leq \overline{F}^1_{x_1}(x_2, \ldots, x_n).$$

Of course, values of x_1 that satisfy both inequalities exist if and only if $F^1_{\overline{x}_1} \leq \overline{F}^1_{x_1}$. By the laws of Boolean algebra, this last inequality is equivalent to the equation $F^1_{\overline{x}_1} F^1_{x_1} = 0$. Therefore eqn. (3) can be solved if and only if the following equation can be solved:

$$F^2(x_2, \ldots, x_n) = F^1_{\overline{x}_1}(x_2, \ldots, x_n) F^1_{x_1}(x_2, \ldots, x_n) = 0. \quad (4)$$

Since function F^2 does not depend on x_1, the number of unknowns has been reduced by one from the original equation. Recursive application of this technique reduces the original system of equations to one equation in the unknown x_n, whose solutions (if they exist) are determined by inequalities of the type: $a^n \leq x_n \leq b^n$, where a^n and b^n are Boolean constants. If $a^n b^n \neq 0$, the system of equations (3) has no solutions. If $a^n b^n = 0$, the system can be solved by back-substituting the values of $x_n, x_{n-1}, \ldots, x_{i+1}$ into the inequalities:

$$F^i_{\overline{x}_i}(x_{i+1}, \ldots, x_n) \leq x_i \leq \overline{F}^i_{x_i}(x_{i+1}, \ldots, x_n). \quad (5)$$

In principle, this algorithm allows the computation of all the solutions of the original set of equations. From a practical standpoint, its main drawback is that the starting point is a function that depends on all the unknowns. Since the number of unknowns can be quite large, it may be impracticable, if not impossible, to handle the functions generated by the algorithm. For this reason, modifications to the algorithm are needed to keep the size of the resulting BDD's, and the complexity of the algorithm, to a manageable level. In [4], the authors proposed the following modification, which takes advantage of the sparsity of the equations. Let:

$$S_i = \{j : f_j \text{ depends on } x_i \text{ but not on } x_1, \ldots, x_{i-1}\},$$

and define:

$$\begin{aligned} G^i(x_i, \ldots, x_n) &= \bigcup_{j \in S_i} f_j(x_i, \ldots, x_n) \\ F^1(x_1, \ldots, x_n) &= G^1(x_1, \ldots, x_n) \\ F^{i+1} &= (F^i_{x_i} F^i_{\overline{x}_i}) \cup G^{i+1}. \end{aligned} \quad (6)$$

It can be shown [4] that F^i depends only on $x_i, \ldots, x_n$ and that the solutions of eqns. (2) and (3) can still be computed through inequalities (4), using the functions F^i as defined in eqn. (6). The advantage of this modifications is that, before variable x_i is eliminated, only those functions that depend explicitly on x_i are included in F^i, and the size of the BDD representing F^i is thus reduced. Experimental results obtained by the authors have shown that this algorithm can handle systems containing hundreds of unknowns in reasonable times.

A further simplification to the algorithm can be made in the following way: if the function g_1 in eqn. (1) does not depend on x_1, the variable x_1 can be eliminated by substituting g_1 for it in the remaining equations:

$$x_i = g_i(g_1(x_2, \ldots, x_n), x_2, \ldots, x_n) \qquad i = 2, \ldots, n.$$

Of course, the substitution has to be performed only for those functions g_i that depend explicitly on x_1. The connection between this algorithm and the one described earlier is illustrated by the following theorem.

Theorem 1 *Let the functions f_i, F^1 and F^2 be as defined in eqns. (2), (3) and (4), respectively. If g_1 does not depend on x_1, then:*

$$F^2(x_2, \ldots, x_n) = \bigcup_{i=2}^{n} f_i(g_1(x_2, \ldots, x_n), x_2, \ldots, x_n).$$

Proof: Consider the Shannon decomposition of f_i:

$$f_i = x_1 f_{i,x_1} \cup \overline{x}_1 f_{i,\overline{x}_1}.$$

Since $x_1 = g_1(x_2, \ldots, x_n)$, and g_1 does not depend on x_1, it follows that:

$$\begin{aligned}\bigcup_{i=2}^{n} f_i(g_1(x_2,\ldots,x_n),x_2,\ldots,x_n) = \\ = \bigcup_{i=2}^{n} g_1 f_{i,x_1} \cup \overline{g}_1 f_{i,\overline{x}_1} = \\ = g_1\left(\bigcup_{i=2}^{n} f_{i,x_1}\right) \cup \overline{g}_1\left(\bigcup_{i=2}^{n} f_{i,\overline{x}_1}\right).\end{aligned}$$

On the other hand, the Shannon cofactors of f_1 are: $f_{1,x_1} = \overline{g}_1$ and $f_{1,\overline{x}_1} = g_1$. From the definition of F^1 and the properties of the Shannon decomposition, it follows that:

$$\begin{aligned}F^1_{x_1} &= \bigcup_{i=1}^{n} f_{i,x_1} = \overline{g}_1 \cup \left(\bigcup_{i=2}^{n} f_{i,x_1}\right) \\ F^1_{\overline{x}_1} &= \bigcup_{i=1}^{n} f_{i,\overline{x}_1} = g_1 \cup \left(\bigcup_{i=2}^{n} f_{i,\overline{x}_1}\right).\end{aligned}$$

Then, from eqn. (4):

$$\begin{aligned}F^2 = \left[\overline{g}_1 \cup \left(\bigcup_{i=2}^{n} f_{i,x_1}\right)\right]\left[g_1 \cup \left(\bigcup_{i=2}^{n} f_{i,\overline{x}_1}\right)\right] = \\ = \overline{g}_1\left(\bigcup_{i=2}^{n} f_{i,\overline{x}_1}\right) \cup g_1\left(\bigcup_{i=2}^{n} f_{i,x_1}\right) \\ \cup \left(\bigcup_{i=2}^{n} f_{i,\overline{x}_1}\right)\left(\bigcup_{i=2}^{n} f_{i,x_1}\right).\end{aligned}$$

But the last term in the equation above is redundant, because, if $g_1 = 0$:

$$\left(\bigcup_{i=2}^{n} f_{i,\overline{x}_1}\right) \cup \left(\bigcup_{i=2}^{n} f_{i,\overline{x}_1}\right)\left(\bigcup_{i=2}^{n} f_{i,x_1}\right) = \left(\bigcup_{i=2}^{n} f_{i,\overline{x}_1}\right),$$

and, if $g_1 = 1$:

$$\left(\bigcup_{i=2}^{n} f_{i,x_1}\right) \cup \left(\bigcup_{i=2}^{n} f_{i,\overline{x}_1}\right)\left(\bigcup_{i=2}^{n} f_{i,x_1}\right) = \left(\bigcup_{i=2}^{n} f_{i,x_1}\right).$$

Hence:

$$F^2 = \overline{g}_1\left(\bigcup_{i=2}^{n} f_{i,\overline{x}_1}\right) \cup g_1\left(\bigcup_{i=2}^{n} f_{i,x_1}\right),$$

which proves the theorem's statement. □

This theorem shows that, if g_1 does not depend on x_1, eliminating x_1 by substituting the function g_1 for it, is mathematically equivalent to eliminating it through Shannon decomposition. algorithm. Of course, if the function $g_2(g_1(x_2,\ldots,x_n),x_2,\ldots,x_n)$ does not depend on x_2, the same technique can be used to eliminate x_2 from the remaining equations. This procedure can be repeated until the system of equations is reduced to:

$$x_i = g_i(x_p,\ldots,x_n), \qquad i = p,\ldots,n, \quad (7)$$

where each function g_i does depend on x_i. From a circuit standpoint, this corresponds to looking at the original network as a block of combinational logic with *p* feedback loops which correspond to variables $x_p, \ldots, x_n$, as shown in Fig. 1. This system can be solved by resorting to the method described in eqns. (1–3).

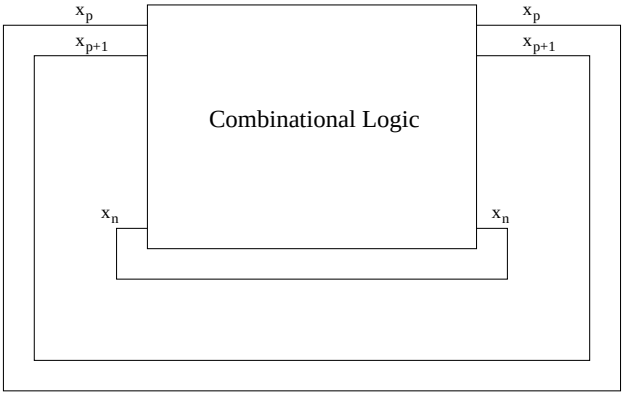

Figure 1. Network corresponding to the reduced set of equations

From a practical standpoint, the advantage of this algorithm is that initially the elimination of a variable is accomplished through one function composition operation, instead of a Shannon decomposition followed by an AND operation. If the functions are represented by BDD's, and the variable ordering is chosen appropriately [7], the composition operation can be performed in a very efficient manner. The next section describes a number of variable numbering algorithms that have been experimented with by the authors.

3 Ordering strategies

It is well known [7] that the size of a BDD representing a Boolean function is affected by the order in which the variables are numbered. Since the complexity of most operations on BDD's is related to the BDD size, it is important to try to keep the size of the BDD's as small as possible throughout the entire variable elimination process. Moreover, since the goal of the algorithm described in the previous section is to eliminate as many variables as possible through function composition, it is also important to number the variables so that the composition operation can

be performed in as efficient a manner as possible. Several variable numbering schemes have been suggested to achieve those goals. This section describes three different implementations of the algorithm described in the previous section, based on three different variable numbering and elimination strategies. In all implementations, the first step consists of identifying the variables corresponding to the feedback loops in a strongly connected components, as shown in Fig. 1. The strongly connected component can now be regarded as a block of combinational logic, whose primary inputs are the feedback loop variables.

The first implementation relies on the numbering strategy suggested in [8]. This method is based on a traversal of the network graph in depth-first order, from primary outputs to primary inputs, with a number of heuristic modifications intended to optimize the size of the resulting BDD. This algorithm yields a numbering for the primary inputs to the networks; BDD's for the primary outputs can then be obtained by repeated applications of the *BDDApply()* function following the network's signal flow graph.

The result of this operation is a set of BDD's representing the functions g_i of eqn. (7). Further simplification of this system of equations is possible if some of the functions g_i do not depend on x_i: in this case the variable x_i can be eliminated from the set of equations by variable substitution through the *BDDCompose()* function, as explained in the previous section. When all possible variable substitutions have been performed, the resulting system of equations is reduced to a single Boolean equation (eqn. (3)) and then solved by repeated applications of the Shannon decomposition, as described at the beginning of Section 2.

The two other implementations of the Boolean equations solution algorithm are based on the variable-numbering strategies suggested in [9]. In the first of the two methods described in this reference, nodes are numbered according to their *level* in the network: this is similar to numbering nodes according to a reverse breadth-first traversal of the network graph. The second method, called *fanin* by the authors, is a combination of depth-first and breadth-first numbering strategies. Both methods assign numbers to all nodes in the networks, including internal nodes, and not just to primary inputs, as was the case for the algorithm described in [8].

Once all variables have been assigned a number, BDD's are built for all the gates in the strongly connected component. This is equivalent to generating a system of Boolean equations of the type shown in eqn. (7), where the number of variables is now equal to the number of nodes contained in the strongly connected component. Of course, a large number of them can be eliminated by variable substitution through the *BDDCompose()* function. When no more variable substitutions can be performed, the system of equations is once again solved by Shannon decomposition.

4 Experimental results

The three implementations described in the previous section were tested on the 1989 ISCAS benchmarks. BDD manipulation was performed using the routine library developed at the ECE Department of Carnegie-Mellon University [10]. The results are reported in Table I: all CPU times refer to a Sparcstation 20/5 running SunOS 4.1.4. The first column contains the size of the largest strongly connected component found in the circuit: this is an upper bound on the size of the system of equations to be solved. The second column reports the time necessary to find the initial state of the network using the algorithm described in [4], which is based on Shannon decomposition. The other three columns refer to the three implementations of the algorithm described in Section 3. It can be observed that the performance of the algorithm described in this paper is vastly superior to the one based on Shannon decomposition for any ordering strategy. The best performance, especially on the largest benchmarks, is given by the first implementation described in Section 3. The superior performance of that implementation does not necessarily mean that the ordering strategy suggested in [8] is better than those described in [9]. Another factor to be taken into account is that the first implementation uses both *BDDApply()* and *BDDCompose()* to build the BDD's, while the other two rely exclusively on *BDDCompose()*. In fact, execution profiles have shown that anywhere from 30 to 85% of the total CPU time is spent in the *BDDCompose()* function. Thus, the efficiency of the composition operation can become the dominant factor in determining the efficiency of the overall algorithm.

5 Conclusion

In this paper we have shown how the performance of an algorithm for the solution of systems of Boolean equations can be greatly improved by taking advantage of the composition operation of Boolean functions represented by BDD's. Because the efficiency of the composition operation and the size of the resulting BDD's are significantly affected by the order in which the variables are numbered, three different ordering strategies were tried. Numerical results obtained on the 1989 ISCAS benchmarks show that the algorithm described here performs significantly better than a previous one developed by the authors. Although the application considered in this paper was the computation of the initial state of a digital network, the algorithm can easily be applied to any system of Boolean equations: all that is necessary is to replace the signal graph of the network with the dependency graph of the system of equations to be solved. Thus it is our belief that this algorithm has potential applications also in other areas, such as state

TABLE I
Solution times (in seconds) for ISCAS89 benchmarks

Ckt.	Max. Size	Shannon Decomp.	Apply Compose	Compose Level	Compose Fanin
s510	28	20.4	0.55	0.98	0.95
s641	12	1.25	0.03	0.97	1.00
s820	17	8.52	0.15	0.60	0.63
s953	22	3.72	0.13	0.65	0.70
s1423	133	69.16	79.13	151.01	202.1
s1488	36	> 3600	0.51	2.58	2.62
s1494	34	> 3600	0.53	2.58	2.58
s5378	14	7.75	0.08	2.35	2.36
s9234	86	> 3600	1.96	40.55	40.82
s15850	159	> 3600	6.90	242.66	276.33
s35932	37	> 3600	12.91	988.80	960.48
s38417	278	> 3600	13.10	422.52	414.62
s38584	2	> 3600	1.13	4.40	4.70

assignment, test generation, and all those problems that require the solution of large systems of Boolean equations.

6 Acknowledgement

The authors would like to thank Kanwar J. Singh, of AT&T Bell Laboratories, for suggesting the idea that led to the development of the algorithm described in this paper.

References

[1] S. Devadas and A. R. Newton, "Exact Algorithms for Output Encoding, State Assignment, and Four-Level Boolean Minimization", *IEEE Transactions on Computer-Aided Design*, vol. 10, no. 1, pp. 13–27, January 1991.

[2] P. C. McGeer, A. Saldanha, P. R. Stephan, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Timing Analysis and Delay-Fault Test Generation using Path-Recursive Funcions", in *Proceedings of the 1991 International Conference on Computer-Aided Design*, Santa Clara, CA, November 1991, pp. 180–183.

[3] T. Larrabee, "Test Pattern Generation Using Boolean Satisfiability", *IEEE Transactions on Computer-Aided Design*, vol. 11, no. 1, pp. 4–15, January 1992.

[4] Scott Woods and Giorgio Casinovi, "Gate-Level Simulation of Digital Circuits Using Multi-Valued Boolean Algebras", in *Proceedings of the 1995 international Conference on Computer-Aided Design*, Santa Clara, CA, November 1995, pp. 413–419.

[5] R. E. Tarjan, "Depth-first search and linear graph algorithms", *SIAM Journal on Computing*, vol. 1, pp. 146–160, 1972.

[6] Sergiu Rudeanu, *Boolean Functions and Equations*, North-Holland Publishing Co., Amsterdam, 1974.

[7] R. E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation", *IEEE Transactions on Computers*, vol. C-35, no. 8, pp. 677–691, August 1986.

[8] M. Fujita, H. Fujisawa, and N. Kawato, "Evaluation and Improvements of Boolean Comparison Method Based on Binary Decision Diagram", in *Proceedings of the 1988 International Conference on Computer-Aided Design*, Santa Clara, CA, November 1988, pp. 2–5.

[9] Sharad Malik, Albert R. Wang, Robert K. Brayton, and Alberto Sangiovanni-Vincentelli, "Logic Verification using Binary Decision Diagrams in a Logic Synthesis Environment", in *Proceedings of the 1988 International Conference on Computer-Aided Design*, Santa Clara, CA, November 1988, pp. 6–9.

[10] Karl S. Brace, Richard L. Rudell, and Randal E. Bryant, "Efficient Implementation of a BDD Package", in *Proceedings of the 27th ACM/IEEE Design Automation Conference*, Orlando, FL, June 1990, pp. 40–45.