Functional Learning: A new approach to learning in digital circuits

Rajarshi Mukherjee Jawahar Jain* Dhiraj K. Pradhan
Dept. Of Computer Science
Texas A&M University
College Station, TX 77843

Abstract

Recently, learning-based techniques, which are extremely effective in finding test vectors for hard to detect faults and in detecting redundancies, have been proposed as an efficient alternative to the traditional branch-and-bound techniques for test generation. This paper presents *functional learning*, a new OBDD-based learning technique, that uses implication procedures. Functional Learning is complete - given enough time, it will determine all the uniquely implied values in the circuit from the current situation of value assignments. The most attractive feature of functional learning is its ability to extract, maintain and manipulate novel information regarding the circuit in compact OBDD representations for Boolean functions.

1 Introduction

Test generation has been proved to be an NP-complete problem [1]. Recently, learning techniques like *Socrates* [4][5][6] and *Recursive Learning* [7][15] have been proposed as an efficient alternative to the decision tree based approaches like PODEM and FAN. The learning techniques involve the temporary injection of logic values at arbitrary signals in the circuit and the subsequent examination of its logical consequences. These techniques prune the search space for test generation by identifying *necessary assignments* based on a given situation of value assignments in a circuit. In this paper we present *functional learning* [11], a new OBDD-based method that can efficiently identify all necessary assignments in a circuit. This method is complete. Given sufficient time, it can identify *all* necessary assignments for a given situation of value assignments in a circuit. Since the problem of identification of all the necessary assignments in a circuit is NP-complete, any method that can identify all necessary assignments must have an exponential time complexity.

The rest of the paper is organized as follows. The principle of *learning* has been illustrated in section 2. A brief overview of *Socrates* and *Recursive Learning* has been presented in section 3. The concept of *functional learning* is introduced in section 4 and the algorithm *functional_implication* is presented in section 4.1. The results of learning operations on an ISCAS 85 benchmark circuit and XOR tree circuits using *functional learning* are presented and analysed in section 5. Concluding discussions and directions for further research have been presented in section 6.

2 The concept of Learning

Learning operations are based on the *Law of contrapositum* which is stated below.

Law of contrapositum: If a Boolean value A on wire a implies a Boolean value B on wire b, then by the *law of Contrapositum* $\overline{B}$ on b implies $\overline{A}$ on a.

The concept of *learning* has been illustrated in the Fig. 1. The Boolean value 0 on the wire f can be justified in two ways: $d = 0$ and $e = 0$. As shown in the figure, $b = 1$ for both these justifications. Therefore, $f = 0$ implies $b = 1$. By the *law of contrapositum*, $b = 0$ implies $f = 1$.

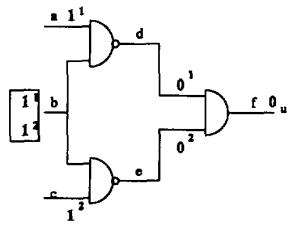

Figure 1: *Indirect implication*

*Computer Engineering Research Center, ENS 403, University of Texas at Austin, Austin TX 78712

0-8186-5440-6/94 $03.00 © 1994 IEEE

3 Overview of Socrates and Recursive Learning

Socrates [4][5] carries out static learning based on the law of contrapositum. In the learning phase, a Boolean 1 and a Boolean 0 are injected at all signals in the circuit. Then, learning operations are carried out after a complete forward and backward implication. The learning phase constitutes the preprocessing phase for the test generation algorithm. The values that are learned during the preprocessing phase are used to prune the search space during the actual test generation using FAN. Due to the reduction in the search space, Socrates is able to identify redundant faults efficiently and can generate tests for hard to detect faults much faster than FAN alone.

It was shown in [7][15] that the learning is incomplete in *Socrates*. By using *precise implications*, *recursive learning* is able to identify *all* the *necessary assignments* in a circuit, given enough time. No backtracks occur during test generation, since *recursive learning* is able to identify all the necessary assignments for fault detection by calling the learning routines recursively to an appropriate depth of recursion.

4 Functional Learning

This section illustrates the basic concept of the proposed *functional learning*.

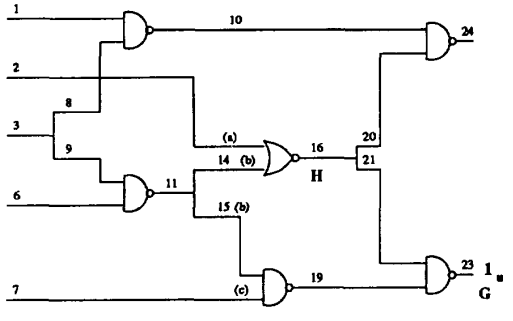

Figure 2: *Functional learning*

Consider the wire 23 in the circuit shown in Fig. 2 to be unjustified to a 1. G and H are the OBDDs for the wires 23 and 16 respectively, built in terms of the pseudo inputs a, b and c. The two OBDDs are shown in Fig. 3. The result of the AND operation between $\overline{G}$ and H is H. This implies that when G is a Boolean 1, H is a Boolean 0. Hence, it is learned that a Boolean 1 on the wire 23 implies a Boolean 0 on the wire 16. By forward implication, it is learned that the wire 24 must carry a Boolean 1. A Boolean 0 on the wire 16 and a Boolean 1 on the wire 24 are the *necessary conditions* for a Boolean 1 on the wire 23.

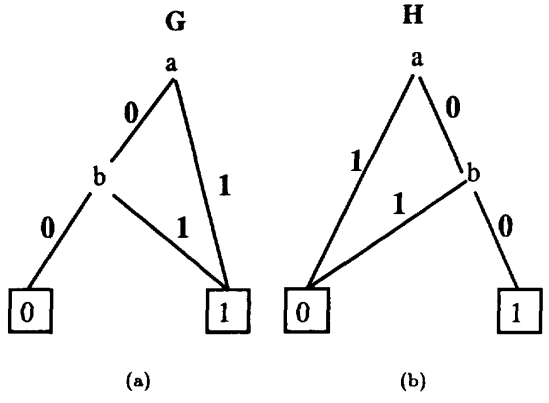

Figure 3: OBDDs for wire 23 and wire 16

In this paper we discuss in detail the algorithm *functional_implication*, which illustrates the basic principle of *functional learning*. We present here some definitions that will be required for the explanation of the algorithm.

Definition 4.1 *An unjustified wire, u is a wire in the circuit that carries a signal value, but all or some of the inputs to the gate which feeds this unjustified wire are unspecified such that there can be one or more combinations of value assignments to the unspecified inputs that can cause a conflict at the output of the gate.*

Definition 4.2 *The* own-cone *owc_w of a wire w in a circuit is the cone that contains all the wires in the circuit that can be reached using a simple backward tracing procedure along the wires, starting from w itself.*

Definition 4.3 *The* other-cone, *otc_w of a wire w is a cone that has some wires in common with owc_w. These wires fan out from the owc_w into otc_w.*

Definition 4.4 *S_G is the functional space which comprises all the 2^n combinations of the input variables of the Boolean function G.*

Definition 4.5 *S_{G_1} is the set of all the combinations of the input variables of the Boolean function G for*

which the value of the function is Boolean 1. S_{G_0} is the set of all the combinations of the input variables of the Boolean function G for which the value of the function is Boolean 0.

A theorem which forms the crux of *functional learning* is stated and proved using set theoretic notation. Let us consider two OBDDs G and H.

Theorem 4.1 **a.** *If $G = 1 \Rightarrow H = 1$ then $G \bigwedge H = G$. Conversely, if $G \bigwedge H = G$, then $G = 1 \Rightarrow H = 1$. By the law of contrapositum $H = 0 \Rightarrow G = 0$.*

b. *If $G = 1 \Rightarrow H = 0$ then $\overline{G} \bigwedge H = H$. The converse is also true. That is if $\overline{G} \bigwedge H = H$, then $H = 1 \Rightarrow G = 0$. By the law of contrapositum $G = 1 \Rightarrow H = 0$.*

c. *If $G = 0 \Rightarrow H = 0$ then $G \bigwedge H = H$. The converse is also true. That is if $G \bigwedge H = H$ then $H = 1 \Rightarrow G = 1$. By the law of contrapositum $G = 0 \Rightarrow H = 0$.*

d. *If $G = 0 \Rightarrow H = 1$ then $\overline{G} \bigwedge H = \overline{G}$. The converse is also true. That is if $\overline{G} \bigwedge H = \overline{G}$, then $G = 0 \Rightarrow H = 1$. By the law of contrapositum $H = 0 \Rightarrow G = 1$.*

Proof:

We prove the case **a**. Two Boolean functions G and H are considered. In this proof it is assumed that $S_G = S_H$. Since, $G = 1 \Rightarrow H = 1$, $S_{G_1} \subset S_{H_1}$. So, it immediately follows that $G \bigwedge H = G$.

In order to prove the converse, we assume that $G \bigwedge H = G$. This $\Rightarrow S_{G_1} \subset S_{H_1}$. This implies that $G = 1 \Rightarrow H = 1$. The cases **b**, **c** and **d** can be similarly proved. □

The following is a direct consequence of the *theorem* 3.1.

Theorem 4.2 *All the learning instances (stated in the above theorem) can be derived by carrying out just two different OBDD AND operations of the OBDDs G and H. They are $G \bigwedge H$ and $\overline{G} \bigwedge H$.*

4.1 Functional implication

Functional implication is the technique used for determining all the necessary assignments that are implied by the already made signal assignments in the circuit. In this section we present the steps of the algorithm *functional_implication*. Then, we illustrate its application on a simple circuit.

Definition 4.6 *l_w is the level of the wire w in a circuit where all the primary input wires are assigned the level 0. The level of any other wire w in the circuit is given by $l_w = max(l_{w_{in}}) + 1$, where w_{in} is an input to the gate whose output is the wire w.*

Definition 4.7 *Given u and owc_u, a cut is defined as a set of wires each at a different level l_{cut}, where $l_{cut} < l_u$. The cut variables are used to build the OBBDs of the wires ahead of it. In functional_implication the cut is chosen such that it includes at least one fanout point.*

The choice of l_{max}, the maximum number of levels for which learning is carried out, decides the amount of learning. Usually, l_{max} is chosen to have a good compromise between the amount of learning and the time taken to learn. The steps of the algorithm are enumerated below.

functional_implication

Step 1. • Inject the initial objective into the circuit.
Step 2. • Find all unjustified wires.
Step 3. • For each unjustified wire build OBDDs using appropriate cuts and learn.
Step 4. • After each learning go to **Step 2.**
Step 5. • Repeat Steps 2, 3 and 4 till either all unjustified wires get justified or the maximum learning level is exceeded.

Consider the circuit shown in Fig. 4. The wire 24 is unjustified to a Boolean 1. Learning operations using the cuts shown in the figure yield the necessary conditions : a Boolean 1 on the wires 18, 46, 54 and 62 respectively.

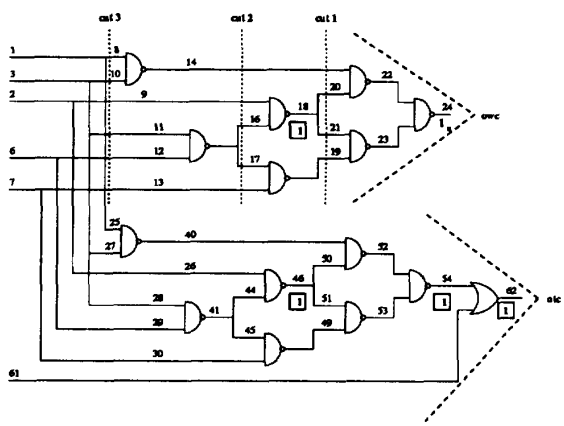

Figure 4: *Example of functional_implication*

In order to reduce the size of the OBDDs built in the *otc*, we have used the efficient technique of *graph*

abstraction. The pseudo input variables that are not common to both the *otc* and the *owc* are abstracted out by introducing dummy variables, as illustrated in Fig. 5.

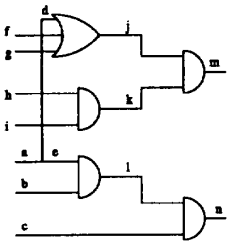

Figure 5: *Graph Abstraction*

Consider the input cone of the wire n as the *owc* and the input cone of the wire m as the *otc*. Sice the pseudo variable a is common to both the cones, while building the OBDDs for the wires in the *otc*, all the variables except for a are abstracted out. For example, the OBDD for the wire j will contain the variable a and a dummy variable v, where v is the abstraction of the variables f and g.

Theorem 4.3 *Let S be the set of all the pseudo variables that are common between the owc_u and the otc_u. If in the otc_u, A is a dominator for S, then if no value has been learned in the input cone of A including at A itself, no learning can take place in the otc_u at all.*

Proof:

Since the fanout pseudo variables are first in the ordering list of the pseudo inputs for the otc_u, after the OBDD for S is built, no new common pseudo variables are added on to the OBDD. The only variables that are involved in further OBDD building are the pseudo variables that are not common to the two cones. So, if no unique value has been learned in the input cone of S then no unique value can be learned anywhere else in the otc_u. □

The following is an extension of the above theorem.

Theorem 4.4 *Let $Dom(S)$ be the set of wires that dominates the members of S. Specifically, let $Dom(S) = a_1, \ldots, a_p$. If the set of intersections for the direct implications for each of the 2^p assignments generated over the members in $Dom(S)$ is equivalently an empty set, no learning is possible.*

The utility of the above theorem can be easily seen in the light of the following corollary.

Corollary 4.1 *If no learning is possible within otc_u, then learning is not possible within any cone C, where $otc_u \subset C$, if $Dom(S)_C \equiv Dom(S)_{otc_u}$.*

The results of the above two theorems and the corollary allow us to identify structures in the circuit where no learning can take place.

5 Results

In this section we present in the Table 1 the results of learning operations carried out on some of the higher numbered wires in one ISCAS 85 benchmark circuit. In the Table 2 results of learning on an artificial XOR tree circuit have been presented. Learning has been carried out for three levels on the ISCAS benchmark circuits. The times are in seconds. The experiments have been conducted on a SPARC station 1. The learning values presented in the table show the cumulative number of wires at which unique values are learned by carrying out learning operations till and inclusive of the present learning level.

The results show that the time for learning need not grow in an exponential fashion as the number of levels of learning is increased. This feature makes *functional learning* very powerful when applied to circuits with a large number of levels.

We also constructed two XOR trees joined at the primary inputs and of different number of levels as shown in Fig. 6. We injected a Boolean 1 or a Boolean 0 at the output of one tree and carried out *recursive learning* and *functional learning*. The results in the Table 2 show that *functional learning* performed faster than *recursive learning* in identifying a Boolean 1 or a Boolean 0 at the output of the other tree.

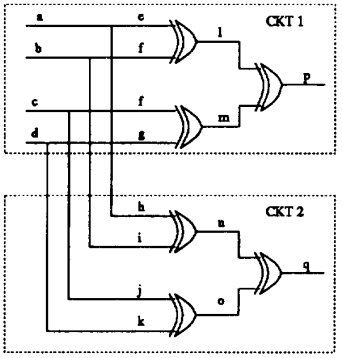

Figure 6: *XOR twin tree*

A complete ATPG tool incorporating FAN and *functional learning* has been implemented. Due to

wire no.	*wire value*	*gate type*	*time*			*learning*		
			L = 1	2	3	L = 1	2	3
432	0	NAND	0.05	0.07	0.05	12	12	12
432	1	NAND	0.02	0.04	0.08	0	0	1
431	0	NAND	0.04	0.08	0.08	16	16	16
431	1	NAND	0.03	0.03	0.04	0	0	0
430	0	NAND	0.02	0.07	0.12	27	27	27
430	1	NAND	0.04	0.06	0.05	0	0	0
429	0	NAND	0.07	0.08	0.1	39	39	39
429	1	NAND	0.04	0.04	0.06	0	0	0
428	0	NAND	0.04	0.05	0.1	33	33	33
428	1	NAND	0.02	0.03	0.04	0	0	0
425	0	NAND	0.04	0.06	0.1	41	41	41
425	1	NAND	0.04	0.06	0.06	2	2	2
422	0	NAND	0.04	0.05	0.06	34	34	34
522	1	NAND	0.03	0.04	0.05	2	2	2
421	0	NAND	0.04	0.06	0.05	0	0	0
421	1	NAND	0.03	0.04	0.09	3	3	3
416	0	NAND	0.04	0.04	0.04	0	0	0
416	1	NAND	0.02	0.08	0.1	51	51	51
414	0	NAND	0.03	0.06	0.07	12	12	12
414	1	NAND	0.03	0.05	0.06	0	0	0

Table 1: *Learning on c432*

levels	T_{fl}	T_{rl}
1	0.02	0.0
2	0.02	0.0
3	0.02	0.0
4	0.02	0.0
5	0.02	0.02
6	0.04	0.12
7	0.06	0.45
8	0.07	3.85
9	0.15	18.9
10	3.39	-

Table 2: *Learning on XOR twin tree*

problems with variable ordering and lack of efficient identification of the structures in the circuit where learning can occur, the performance of the tool is presently somewhat slow. But, these problems are being taken care of as part of further work. For example potential learning areas in the circuit can be determined by taking an intersection of implication traces of a small subset of justification vectors. Preliminary results indicate that this prevents unnecessary number of learning operations. Improved ordering methods such as [13] are being incorporated to keep the graph sizes in check.

6 Conclusions and further research

In this paper we have presented a new method for learning in digital circuits, called *functional learning*. We have demonstrated the power of learning of this new algorithm by carrying out learning operations on an ISCAS 85 benchmark circuit and XOR twin tree circuits. The present research is directed towards improving various features of the ATPG tool that has been implemented based on *functional learning*. Efficient variable ordering techniques are being investigated for reducing the size of the OBDDs. Work has been started on implementing a tool for design verification of digital circuits based on *functional learning*. As future research, we intend to apply the proposed learning technique to the field of combinational circuit optimization.

References

[1] Fujiwara H., Toida S., "The Complexity of Fault Detection Problems for Combinational Logic Circuits", IEEE Transactions on computers, vol. C-31, June 1982, pp. 555-560.

[2] Goel P., "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Cir-

cuits", IEEE Transactions on Computers, vol. C-30, pp. 215-222, Mar. 1981.

[3] Fujiwara H., Shimono T., "On the Acceleration of Test Generation Algorithms", Proc. 13th Int. Symp. Fault Tolerant Computing, pp. 98-105, 1983.

[4] Schulz M., Trischler E., Safert T., "SOCRATES: A highly efficient automatic test pattern generation system", Int. Test Conf., 1987.

[5] Schulz M., Trischler E., Safert T., "SOCRATES: A highly efficient automatic test pattern generation system", IEEE Transactions on Computer-Aided Design, vol. 7, NO. 1, pp. 126-136, Jan. 1988.

[6] Schulz M., Auth E., "Improved Deterministic Test Pattern Generation with Applications to Redundancy Identification", IEEE Transactions on Computer-Aided Design, vol. 8, NO. 7, pp. 811-816, July 1989.

[7] Kunz W., Pradhan D. K., "Recursive Learning: An Attractive Alternative to the Decision Tree for Test Generation in Digital Circuits", Proc. Int. Test Conf., pp. 816-825, 1992.

[8] Bryant R. E., "Graph-Based Algorithms for Boolean Function Manipulation", IEEE Transactions on Computers, vol. C-35, no. 8, Aug. 1986.

[9] Brace K. S., Rudell R. L., Bryant R. E., "Efficient Implementation of a BDD Package", Proc. 27th Design Automation Conf., pp. 40-45.

[10] Brglez F., Fujiwara H., "A Neutral Netlist of 10 Combinational Benchmark Designs and a Special Translator in Fortran", Proc. Intl. Symp. on Circuits and Systems, Special Session on ATPG and Fault Simulation, June 1985.

[11] Mukherjee R., Jain J., Pradhan D. K., "Functional Learning: A new approach to learning in digital circuits", Technical Report 94-020, Dept. of Computer Science, Texas A&M University, Feb. 1994.

[12] Kunz W., Pradhan D. K., "Accelerated Dynamic Learning for Test Generation in Combinational Circuits", accepted for publication in IEEE Transactions on Computer-Aided Design.

[13] Jain J., Bitner J., Moundanos D., Fussell D. S., Abraham J. A., "A New Scheme to Compute Variable Order for Binary Decision Diagrams", Fourth Great Lakes Symposium on VLSI, March 1994.

[14] Kunz W., "HANNIBAL: An Efficient Tool for Logic Verification Based on Recursive Learning", Proc. Int. Conf. Computer Aided Design, 1993.

[15] Kunz W., Pradhan D. K., "Recursive Learning: A Precise Implication procedure and its application to test generation in digital circuits", accepted for publication IEEE Transactions on Compter-Aided Design.