VERIFUL : VERIfication using FUnctional Learning*

Rajarshi Mukherjee†
Dept. of Electrical and Computer Engineering
University of Texas at Austin
Austin TX 78712

Jawahar Jain Masahiro Fujita
Fujitsu Laboratories of America
77 Rio Robles, San Jose
CA 95134-1807

Abstract

It is well known that learning (i.e., indirect implications) based techniques perform very well in many instances of combinational circuit verification when the two circuits being verified have many corresponding internal equivalent points. We present some results on combinational circuit design verification using a powerful, and highly general learning technique called functional learning. Functional learning is based on OBDDs and hence can efficiently learn novel implications based on functional manipulation.

1 Introduction

Analysis of a logic design, constituting of problems such as, representation, verification, ATPG, etc., poses one of the most fundamental challenges in the field of computer-aided design. For example, logic verification of two different realizations of the same Boolean function during the process of circuit synthesis is of utmost importance to guarantee correctness of the circuit being implemented.

As discussed in [16], digital circuit synthesis typically consists of a sequence of atomic operations through which the circuit is altered locally to suit specific needs, keeping the functionality same. Hence, it can be argued that after each such atomic change the two versions of the circuit before and after the change remain very similar. This fact immediately lures one to try and extract these internal equivalences and use them effectively in order to simplify the problem of logic verification. Berman *et. al.* [15] proposed the first method of using these internal equivalent points to establish the equivalence of two circuits. In [15] a decomposition is found using the min/cut algorithm that facilitates decomposing the problem of verification of the whole circuit into much smaller and simpler problems. Cerny and Mauras presented in [18] further observations to establish cross-relations between two appropriate cuts in the two circuits.

Learning techniques [8, 12] can often be efficiently used to extract the internal equivalent points which are used subsequently to speed up the process of design verification. However, as shown in [15], a direct use of these equivalent points to prove the outputs of two circuits equivalent can cause the problem of *false negatives*. This happens when one attempts to compare the functionality of the two circuits using these equivalent points as *pseudo primary inputs*. It is easy to see that this kind of comparison can erroneously prove the two circuits to be functionally inequivalent even though they may actually be equivalent. This is because the interdependence of the *pseudo primary inputs* in terms of the true primary inputs is neglected.

A technique for extracting and utilizing internal equivalent points for logic verification without having to face the problem of false negatives was presented in [14]. Another interesting technique to carry out design verification using internal equivalences and observability don't cares has been presented in [17]. However, techniques such as [17] may not discover even all necessary assignments. Techniques of [14] can scale poorly with increasing circuit sizes as was found in some industrial circuits [3]. Most such techniques find it very difficult to detect relationships between points not in structural proximity. Importantly, the types of relationships that these techniques can discover are limited. For example, these methods can learn *Constant-Value Relationships*: if a constant Boolean value $v \in \{0, 1\}$ at a given gate implies another constant Boolean value at another gate. However, they cannot detect more involved relationships between a set of functions with another set of functions. For example, a gate $f = 0$ may simply imply that a disjunction taken over some given set of functions must be 1. Or, under $f = 0$, a set of gates must assume identical value. Clearly, these techniques cannot easily learn conditions implied by more complex Boolean relations among two or more gates in the circuit.

We suggest one possible solution to the combinational verification problem through a learning technique based on OBDDs. We will show that this technique can work as efficiently or even better than typical learning techniques. The capability of OBDDs to model various functions and also symbolically manipulate them is well accepted. Hence it is not surprising that OBDDs can be also easily used to derive more involved internal relationships *implications* as well. Our results show that the sizes of the OBDDs that are required to be built are extremely small. Hence, there is no memory explosion. Although not used in the present work, use of dynamic reordering [11] can make it even further efficient for most combinational circuits encountered in real life.

In this paper we will chiefly focus on demonstrating that functional learning is an efficient tool for detecting internal correspondences. Since constant-value relationships are the only kind which existing learning techniques can detect, our chief task in this paper is to show that functional learning can be as efficient as exist-

*The first author was supported in part during the early part of this work by the ONR under grant N00014-92-J-1366 and the National Science Foundation under grant MIP-9218238

†Currently also with Fujitsu Laboratories of America, 77 Rio Robles, San Jose CA 95134-1807

ing learning techniques in the determination of constant-value relationships.

2 On Functional Learning

The learning techniques involve the temporary injection of logic values at arbitrary signals in a digital circuit and the subsequent examination of its logical consequences. Socrates [8] carried out *static learning* and the method of learning was further improved in [12]. Functional learning, introduced in [1] is easily seen as the superset of all the previous learning methods [8, 12]. Functional learning is a *complete* method, i.e. given sufficient time, it can identify *all* necessary assignments from a given situation of value assignments in a circuit. The concept of functional learning is explained below with the help of an example. We will illustrate the added capability of functional learning using an example in Figure 3.

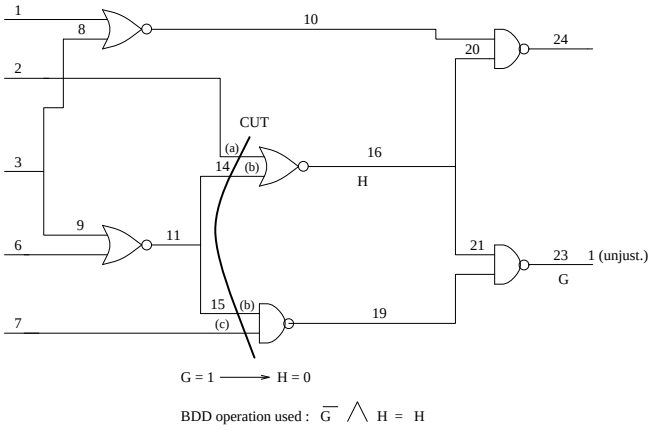

Figure 1: *Functional learning*

Consider the wire 23 in the circuit shown in Figure 1 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 Figure 2 [1]. 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.

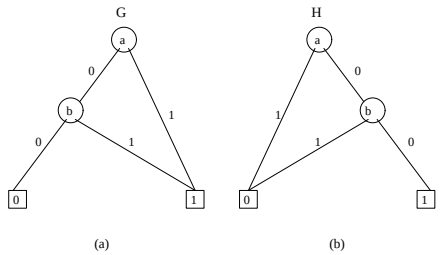

Figure 2: *OBDDs for wire 23 and wire 16*

The theorem that governs the learning operations in functional learning is presented below [1].

Theorem 2.1 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* $G = 1 \Rightarrow H = 0$.

As shown in [1], the above two operations suffice in checking any simple constant-value relationship between two points G *and* H.

As shown in the Figures 4, if a Boolean 1 is injected at the gate G in the circuit shown in the Figure 3, then the gate becomes unjustified. The OBDD for this wire in terms of (a,b,c) is shown in the Figures 4(a), and the OBDDs for both the wires H, and I, in terms of (a,b,c,d) are shown in the Figure 4(b). The result of the AND operation between the OBDD G and the OBDD H is OBDD R and is shown in Figure 4(c). It can be seen that there is no unique learning but we learn that under the condition G = 1, both Y1 and Y2 must carry identical Boolean values. Note that checking for the equivalence of two OBDDs is a constant time operation.

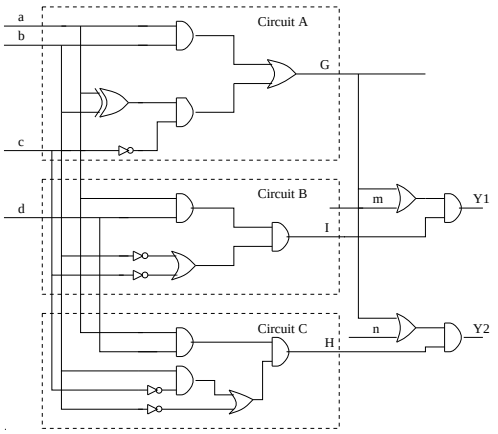

Figure 3: *Functional learning*

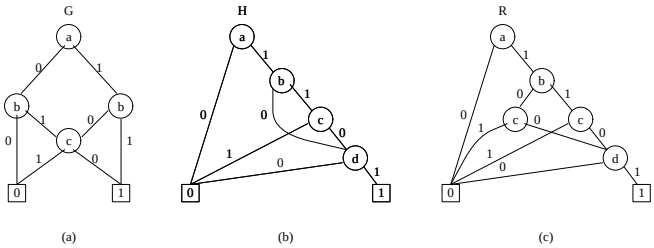

Figure 4: *BDDs in Circuit of Figure 3. (a) BDD G; (b) BDD H; (c) BDD* $G \wedge H$

2.1 Precise marking of potential learning area

Given an unjustified wire, functional learning first chooses an appropriate *cut*. The present technique for choosing a *cut* is based on a simple breadth-first traversal of the transitive fan-in cone of the unjustified wire, starting at the unjustified wire. Next, the OBDD for the unjustified wire is built in terms of the *cut* variables. Once the OBDD is built, appropriate AND operations,

as explained in the previous section, must be performed in order to learn indirect implications. But, the wires where learning will be possible under the given situation of value assignments in the circuit are not known before hand. In order to precisely demarcate the *potential learning areas* in the circuit a preprocessing of the OBDD is carried out.

Definition 2.1 *A justification vector in an OBDD is a path from the root variable in the OBDD to that terminal node whose value is equal to the value of the unjustified wire.*

During the preprocessing of the OBDD for the unjustified wire, a constant k number of justification vectors are extracted. These vectors are applied to the circuit and a complete implication is carried out. After the application of all the k justification vectors, the wires that carry common Boolean values for all the consistent justification vectors are marked as the *potential learning areas*. Only these wires are subjected to the learning operations using the procedure explained earlier. The number of justification vectors that need to be applied in order to demarcate the potential learning areas with high amount of precision is a matter of heuristic. The process of justification vector evaluation is stopped if either no new wires are eliminated from the list of potential learning areas after two successive evaluations or if a user-defined upper limit on the number of justification vectors to be extracted is reached or if evaluation of all the justification vectors in the OBDD is completed.

In the case of a larger OBDD for which a complete enumeration of all the justification vectors is not feasible, the procedure outlined above gives the wires which comprise the *potential learning area*. Once the *potential learning area* has been marked, learning operations based on the theorem stated above are carried out at these wires.

3 Algorithm for Verification

As is customary in such approaches, the problem of design verification has been converted to the problem of checking for satisfiability. The two circuits to be verified are joined at their primary inputs. Their corresponding primary outputs are fed in pairs to 2-input XOR gates. This new circuit will be henceforth referred to as the *composite circuit*. Thus, if the two circuits are to be proven inequivalent, all that needs to be done is to prove the satisfiability of the output of any of the above XOR gates. In this work it is assumed that there are no external don't cares in the circuits. However, this work can be extended to incorporate external don't cares as well. In this context, results presented in [19] can be easily incorporated.

The complete flow diagram for the logic verification algorithm is shown in Figure 5. In the learning phase, indirect implications from one circuit to the other circuit are learned by injecting Boolean values at different wires in the circuit.

At each gate a Boolean value is injected such that the gate becomes unjustified. For example a Boolean 0 is injected at the output of an AND gate and a Boolean 1 at the output of an OR gate. Similarly, both a Boolean 0 and a Boolean 1 are injected at the output of an XOR/XNOR gate. This phase is carried out using a

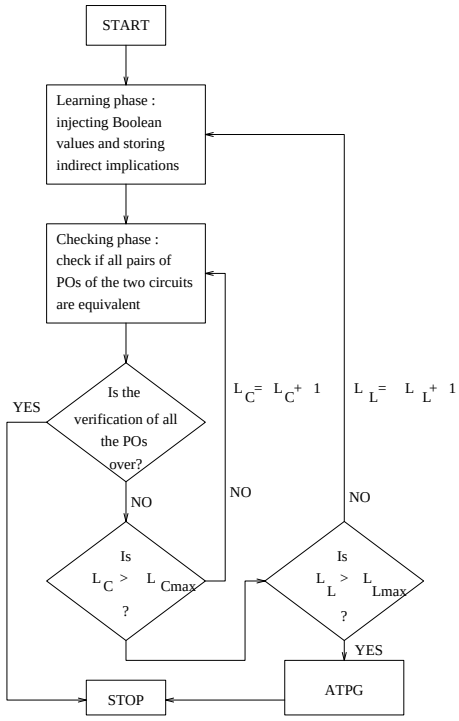

Figure 5: *Flow Diagram for Verification*

certain initial level of learning L_L. In our implementation, we start with an initial learning level of 1. All the indirect implications learned during this phase are stored along with the data structure of the wire from which it was learned. In this phase it is important to have finished processing all the wires in the transitive fan-in of the present wire. This is needed to ensure that during the processing of the present wire we can make use of the pre-stored implications of the wires in its transitive fan-in to speed up the process of learning indirect implications.

In the checking phase, we check for the equivalence of the corresponding primary outputs of the two circuits. In this phase a Boolean 1 is injected at the output of the appropriate XOR gate and prestored implications are used to try and prove a conflict of this situation of value assignment. An OBDD for the output of the XOR gate is built in terms of an appropriate cut at a level L_C in the two circuits. Next, the indirect implications among the cut variables are utilized in order to try and prove that all the paths in the OBDD starting from the topmost variable and terminating at the 1 node are inconsistent. If a conflict is proved, then the two corresponding primary outputs are equivalent. This phase is initiated with an initial level $L_C = 1$. If the two outputs cannot be proved to be equivalent, then L_C is incremented and the process is repeated till L_C exceeds a preset L_{Cmax}. If after this phase, we still have primary outputs remaining that have not been shown to be equivalent, the learning level for the learning phase is incremented and learning for indirect implications is carried out with a higher precision. If the learning level L_L for learning phase exceeds a preset L_{Lmax}, then for the remaining primary outputs we use an ATPG tool that tries to generate a test for the fault s-a-0 at the

output of the appropriate XOR gate at the output of the composite circuit. The ATPG tool makes use of all the pre-stored indirect implications, thus effectively reducing the search space for a test. The two corresponding primary outputs are equivalent if the fault is proved to be redundant. Otherwise, a test vector is generated which is the distinguishing vector for the two circuits. If the ATPG tool fails to prove the fault to be redundant and also fails to find a test vector for the fault, then VERIFUL aborts. It should be noted that the learning phase is by itself a complete algorithm for logic verification. Given enough time, the learning phase itself can prove the circuits to be equivalent or inequivalent as the case may be.

Now, we briefly discuss some other interesting characteristics of functional learning. However, detailed proofs of the theorems have been omitted due to space constraint.

Assume that $S_f(L)$ and $S_r(L)$ are the set of conditions learned respectively by the functional learning (FL) technique when operating at a distance L from an initial value assignment (or "operating at level L" as we will use in the following), and recursive learning (RL) technique at recursion level L. It can be proved that

Theorem 3.1 $S_f(L) \supseteq S_f(L-1) \cup \ldots \cup S_f(L-2) \cup S_f(1)$

An important difference between FL and RL techniques can be noticed in following. Assume $t_f(L), t_r(L)$ are the time required to learn operating at level L in FL, and recursion level L in RL. Hence $T_f(L) = \sum_{i=1}^{i=L} t_f(i)$ is the time required if the method begins by learning at each level beginning from level 1. However, in FL one can choose to begin at level L and bypass the efforts required at earlier levels.

Theorem 3.2 *In Recursive learning, to learn conditions in $S_r(L)$ - at recursion level L, one needs to spend at least $T_r(L) = \sum_{i=1}^{i=L} t_r(i)$ units of time. However, in functional learning, only $t_f(L)$ units of time can suffice in providing the lower bound on disovering $S_f(L)$.*

Ignoring the arguments about BDD variable ordering, an RL method apparently has a higher computational complexity. As shown in [1], for some functions, FL takes time polynomial in number of gates analyzed but RL requires exponential time resources. The reverse is not true. It does not appear possible one can require polynomial time for an RL method but exponential time for the FL method. Interestingly, our experiments show that FL performs equally well as RL based verification even on c6288, a multiplier, having an exponential representation using OBDDs.

Note, typically a BDD has exponentially many path in it. Hence it is wiser to write the conjunction of all learned conditions themselves as a BDD. Hence learning $a = 0 \Rightarrow b = 1$ can be written as $(\overline{a} \wedge b) \equiv \overline{a}$. Let each of the learning condition c_i be expressed as a BDD, and LC represent the set of such BDDs. Let λ be some cutset such that it separates the output F_1 of circuit C_1 and output F_2 of circuit C_2 from the primary inputs. Consider $F_1(\lambda)$ and $F_2(\lambda)$ to be the corresponding output OBDDs when expressed in terms of gates at the cut λ. In other words, the gates on cut λ are now the (pseudo) inputs through which outputs F_1 and F_2 are expressed. Let $\mathsf{LC}(\lambda)$ be the set of learning condition BDDs, again expressed in terms of cut-points (gates) on λ.

Theorem 3.3 *F_1 and F_2, corresponding output functions of the circuits C_1 and C_2 are equivalent if $(F_1(\lambda) \oplus F_2(\lambda)) \wedge c_i$, $c_i \in \mathsf{LC}(\lambda)$ is not satisfiable.*

For a path enumeration mechanism, following simple observation is quite useful.

Observation: A scheme working by path extraction can easily exhaustively analyze a function without extracting all paths since we wish to test for satisfiability only by determining whether a conflict exists on all satisfiable paths. Let ω be some cutset separating the root and the terminals in BDD $F_1(\lambda) \oplus F_2(\lambda)$.

Theorem 3.4 *If we (breadth-first) enumerate all paths from root leading up to cutset ω within BDD $F_1(\lambda) \oplus F_2(\lambda)$ then $F1$ and $F2$ are not equivalent if there is a conflict on each such path.*

4 Results

In this section the results on ISCAS 85 benchmark circuits are presented. These results are extremely encouraging and prove that functional learning based verification can be a very powerful tool that designers can use to verify their designs during the synthesis procedure.

We present here the results where the redundant combinational circuits are verified against their non-redundant versions. To obtain a correct perspective of our results, note that the corresponding space as well as times required for verifications using OBDDs is about three orders of magnitude higher for circuits such as c6288! The experiments have been carried out on a Sun Sparc Station 10. The times reported are in seconds.[1]

The results show that functional learning is an extremely fast and efficient method to extract internal equivalences and indirect implications in digital logic. Note, Theorems 3.3 and 3.4 were *not* employed for the results obtained in this paper; only simple path enumeration was used in the checking phase. An added advantage of functional learning that we intend to utilize in our future work is its ability to work on circuits where parts of the logic are not represented as logic gates but simply as Boolean functions. This is because of its capability to manipulate information on Boolean functions by using BDDs. Note that the other existing learning techniques [8, 12] do not have this capability. We also intend to use additional relationships (indirect implications which are not constant-value relations) that are easily described in our approach.

5 Conclusion

In this paper we have presented the preliminary results on VERIFUL, a design verification tool based on functional learning, an extremely powerful learning technique for digital logic, which was first introduced in [1]. We have conducted experiments on the ISCAS 85 benchmark circuits. These experiments indicate that

[1] Note, the number of prestored implications are indirect implications except a small number of dispensable, direct implications that our preliminary version of program stores.

Table 1: *Verification of some benchmark circuits*

Circuits being verified	L_L	L_C	# prestored implications	CPU time (min:sec)
c432 vs. c432nr	1	2	976	00:0.49
c499 vs. c499nr	1	1	1056	00:1.14
c1355 vs. c1355nr	1	1	2616	00:3.50
c499 vs. c1355	1	1	3834	00:1.94
c1908 vs. c1908nr	1	2	5336	00:5.76
c2670 vs. c2670nr	1	1	7498	00:48
c3540 vs. c3540nr	1	1	28541	06:05
c5315 vs. c5315nr	1	1	17126	06:57
c6288 vs. c6288nr	1	1	9194	00:24.5
c7552 vs. c7552nr	1	1	69379	31:51

functional learning based verification can become a very powerful and versatile tool for use during circuit synthesis. Further, in this paper we have also briefly pointed out theoretically the superiority and greater power of functional learning over the other existing learning techniques [8, 12]. Many other interesting properties of functional learning can also be shown as well as the fact that these techniques can easily be integrated with conventional OBDD based verification methods to yield very powerful verification methodologies [2]. After further improved integration of current programs with a sophisticated ATPG tool, our subsequent research will be directed towards the application of functional learning to the fields of OBDD based sequential circuit testing [13] and verification, and optimization of both combinational and sequential logic circuits. We also intend to use various new kinds of learnings such as indirect implications between sets of functions and generalized relationships in a circuit that can be easily extracted by our method.

References

[1] Mukherjee R., Jain J., Pradhan D. K., "Functional Learning: A new approach to learning in digital circuits", Proc. IEEE VLSI Test Symp., pp. 122- 127, April 1994

[2] Jain J., Mukherjee R., Fujita M., "Advanced Verification Techniques Based on Learning" Submitted for Publication.

[3] Entrena L., Cheng K-T., "Sequential Logic Optimization By Redundancy Addition And Removal", ICCAD, 1993, pp. 310-315.

[4] Goel P., "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits", IEEE Transactions on Computers, vol. C-30, pp. 215-222, Mar. 1981.

[5] Malik S. *et al.*, "Logic Verification using Binary Decision Diagrams in a Logic Synthesis Environment", ICCAD, 1988, pp. 6-9.

[6] Fujita M., Fujisawa H., Kawato N., "Evaluation and Improvements of Boolean Comparison Method Based on Binary Decision Diagrams", ICCAD, 1988, pp. 2-5.

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

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

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

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

[11] Rudell R. L., "Dynamic Variable Ordering for Ordered Binary Decision Diagrams", Proc. ICCAD, 1993, pp. 42-47.

[12] 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.

[13] Moondanos, J. and Abraham, J. A., "Sequential Redundancy Identification Using Verification Techniques", Proc. Int. Test Conf., pp. 197-205, 1992.

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

[15] Berman C. L., Trevyllian L. H., "Functional Comparison of Logic Designs for VLSI Circuits", ICCAD, 1989, pp. 456-459.

[16] Aas E. J., Klingsheim K., Steen T., "Quantifying Design Quality: A Model and Design Experiments", Proc. of EURO ASIC 1992, pp. 172-177.

[17] Brand D., "Verification of Large Synthesized Designs", ICCAD, 1993, pp. 534-537.

[18] Cerny E., Mauras C., "Tautology Checking Using Cross-Controllability and Cross- Observability Relations", ICCAD, 1990, pp. 34-38.

[19] Shiple T. R., Hojati R., Brayton R. K., "Heuristic minimization of BDDs using don't cares", DAC, 1994, pp. 225-231.