Department of Computer Science
University of Manchester

Manchester M13 9PL, England

Technical Report Series

UMCS–92–4–1

Behnam Bani-Eqbal

Exclusive Normal Form of Boolean Circuits

Exclusive Normal Form of Boolean Circuits [*,†]

Behnam Bani-Eqbal

Department of Computer Science
University of Manchester
Oxford Road, Manchester, UK.
behnam@cs.man.ac.uk

Technical reports issued by the Department of Computer Science, Manchester University, are available by anonymous ftp from m1.cs.man.ac.uk (130.88.13.4) in the directory /pub/TR. The files are stored as PostScript, in compressed form, with the report number as filename. Alternatively, reports are available by post from The Computer Library, Department of Computer Science, The University, Oxford Road, Manchester M13 9PL, UK.

[†]This work was supported both by the Science and Engineering Research Council Under Grant GR/F/09570.

Abstract

Recently Binary Decision Diagrams (BDD) have been proposed as an efficient representation of boolean functions [1]. Although most practical functions have a compact BDD, some useful ones do not. A prime example of these is the integer multiplier function, which was proved to have at least exponential complexity [2], and much worse in practice. We propose a modification of BDDs using the exclusive or form and show that it leads to a more compact representation for the multipliers.

1 Introduction

Boolean functions are an important area of both digital circuit design and computer science. Many different techniques have been developed to represent and manipulate these functions. A most useful method has been the binary decision diagrams (BDD) with variable orderings of R. Bryant [1]. This representation allows efficient implementation of boolean operations, including testing for equality. Since the introduction of BDDs, implementations and extensions of the algorithms have been developed by research workers and applied to complex problems, for some references on such work refer to the BDD papers in [3].

However, the BDD approach does have its problems. First, an ordering of the variables has to be fixed before the graph is determined. In fact choosing the wrong ordering could result in an exponential graph. Worse still, some functions have an exponential size BDD regardless of the ordering. The chief examples of such functions are the output bits of an n-bit integer multiplier. This was conjectured in [1] and then subsequently proved by the same author in [2] where a lower bound of $2^{n/8}$ is obtained for the n bit graph. Experimental results however indicate that the bits have much more complex BDDs (probably double exponential in size). For instance for n=8, the graph has a size of around 5000 vertices, but for n=10 this grows to more than 100 000 nodes and thus the BDD approach becomes impractical.

An examination of the integer multiplier, using the shifted partial products method indicates that the essential operation is multiple addition. Now the most appropriate boolean function for addition is the exclusive or. Furthermore the exclusive or enjoys many useful properties, and in fact any boolean function has a canonical exclusive normal form (XNF). We can therefore modify the Bryant's BDDs to take advantage of this canonical form, to obtain the exclusive decision diagram (XDD) of boolean functions. As expected we shall see that the integer product bits have a more compact XDD. Unfortunately experimental results show that this is still exponential with a growth rate of around 3.

Our method does have disadvantages over the BDD approach. In particular, the operations of OR or AND on two XDDs could result in an exponential blow-up in the worst case. However XDDs genuinely differ from the BDDs. In fact it will be shown that there are simple functions with a small XDD which convert to an exponential BDD, and conversely.

2 Representation

Throughout this paper, $\oplus$ denotes the exclusive or. To represent a boolean function f, we first write it as an OR sum of products, with a term for each 1 entry in the truth table. The sum can be rewritten to exclusive or, as only one term is true in any assignment of variables. Next, we replace the negation of any variable $\overline{x}$ with $1 \oplus x$, and the expand the result. We get an xor sum of positive terms. Then if two terms are equal then we delete both and keep going till there are no duplicates. The resulting formula is the exclusive normal form (XNF) of f.
Example: $x \vee y = xy \oplus x\overline{y} \oplus \overline{x}y = xy \oplus x \oplus y$.

Lemma 1 *Two XNF f, g are equal iff they are identical.*

Supposing $f = g$ and choose a variable x in f. We can write $f = f_1 \oplus xg_1$ and $g = f_2 \oplus xg_2$ uniquely where f_1, f_2, g_1, g_2 are in XNF and do not contain x. Putting $x = 0$, we get $f_1 = f_2$. By induction on the number of variables, f_1 and f_2 are identical. Putting $x = 1$ and xoring f_2 we get $g_1 = g_2$ which again implies $g_1 \equiv g_2$. For the start of induction, ie the no variables case, observe that the only XNF formulae with no variables are 0, 1, and the truth of the lemma is immediate.

We can convert an XNF formula into a directed acyclic graph by a procedure essentially similar to the BDD case. Each node is labelled by a variable. The non-terminal nodes have two outgoing edges labelled by 0 and 1. The procedure goes as for BDDs. We must first specify an ordering of the variables. Then we construct the graph starting from a node corresponding to f and labelled by the first variable. To grow the children of a node labelled by the variable x and corresponding to the formula f, we decompose f in the XNF form $f = f_1 \oplus xf_2$ where f_1 and f_2 are XNF. The children are two nodes corresponding to f_1 and f_2 and labelled by the next variable from x in the ordering. The edge to f_1 is labelled by 0 and that to f_2 by 1. The growth process shares duplicate nodes so that if either of f_1 or f_2 has been dealt with before then only an edge to that node is added. A branch will terminate when the function is constant true or false which correspond to the special terminal nodes 0 or 1. When all the branches have terminated, the resulting graph is the reduced xor decison graph (XDD) of f.

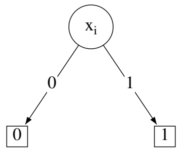
It is easy to see how to retrieve a function from its xor graph. Also a function can be evaluated by traversing its xor graph. This however is distinct form BDDs. The graph is traversed from the root as the variable assignments are evaluated. The difference from BDD evaluation lies in that a list of nodes must be kept. This list starts with just the root node. The partially evaluated function is the xor of all the node (functios) on the list. If the next variable is assigned 0 then every node in the set is replaced by its child connected via the 0 edge. However if the assignment is 1, then the nodes are replaced with both their children. The list can be simplified by deleting any two identical nodes in it. Once all the variables are examined, the list is either empty or consists of 0 which yields a value of 0, or contains 1 which yields a value of 1.

We note that in the production of the XDD, duplicate subgraphs were avoided by re-using a node when its function is identical with the one at hand. As for BDDs, we call such an XDD reduced. However when we perform a boolean operation on two (reduced) XDDS, a nonreduced graph may result. We can reduce the graph by a similar procedure as that described in [1]. Also the following is proved similarly to the theorem 1 of [1].

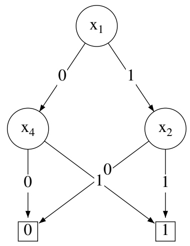
Lemma 2 *For any boolean function there is a unique reduced xor graph denoting it, and any other xor graph denoting f has more vertices.*

3 Examples

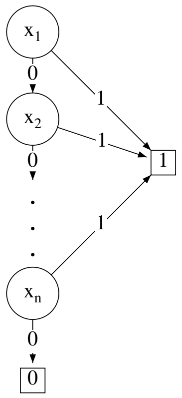
We have taken these examples from [1].



x_i



$x_1x_2 \oplus x_4$



$x_1 \oplus x_2 \oplus \cdots x_n$

4 Boolean Operations on XDD

In this section we consider algorithms to perform boolean operations on the xor graphs. We will not consider the data structures as they are similar to the BDD case. The reduction algorithm, which is purely dependent on the graph structure is identical to the BDD case. Let us consider the boolean operations. Suppose that we have two graphs G_1 and G_2 representing F_1 and F_2. The operations of negation and xor works straighforwardly. For negation we note that $\neg F = 1 \oplus F$. However AND, OR operations are not so easy. Consider AND first. Suppose that x is the first variable and decompose $F_1 = f_1 \oplus xg_1$, $F_2 = f_2 \oplus xg_2$. Then $F_1F_2 = f_1f_2 \oplus x(f_1g_2 \oplus f_2g_1 \oplus g_1g_2)$. So in the next iteration we need to find the xor sum of 3 products. Carrying on, in general the product graph will have nodes corresponding to an arbitrary sum of products of nodes from G_1 and G_2. Thus the result is of exponential size. As for the BDD case we use a table to remember which sums have been seen. The OR function is dealt with via the formula: $F_1 + F_2 = F_1 \oplus F_2 \oplus F_1F_2$. There are examples to show that the AND operation can result in a blow-up in the worst case.

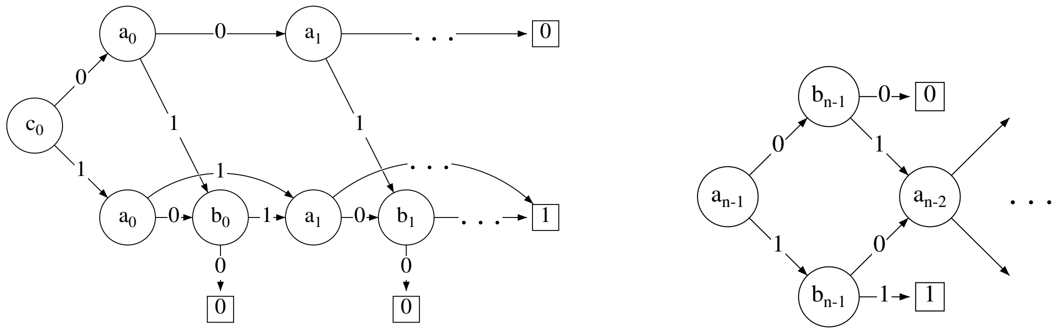
5 Arithmetic Circuits

Let us consider addition first: $A + B + c_0$ where $A = a_{n-1} \cdots a_0$ and $B = b_{n-1} \cdots b_0$ and c_0 is the carry-in. We will consider the graph of the carry function c_n emerging from the top bit, under the two orderings $c_0a_0b_0a_1b_1\cdots b_{n-1}$ and the reverse one. We use the following relations:

$$c_n = c_0(a_0 \oplus b_0)(a_1 \oplus b_1)\cdots \oplus a_0b_0(a_1 \oplus b_1)(a_2 \oplus b_2)\cdots \oplus a_1b_1(a_2 \oplus b_2)\cdots \oplus \cdots \oplus a_{n-1}b_{n-1}$$

$$c_n = a_{n-1}b_{n-1} \oplus (a_{n-1} \oplus b_{n-1})c_{n-1}$$

We get these graphs for the two given orderings:



The integer multiplier is obviously much more complex. Let us set $P = A \times B$, where the i-th bit of P is p_i. We are interested in the p_{n-1}, being the most complex output bit. We can in fact compute its exclusive normal form directly from the partial products summation, using:-

Lemma 3 *Given booleans b_i, the n-th bit of the sum $\sum b_i$ is the XOR of all the distinct AND combinations of 2^n variable subsets of b_i.*

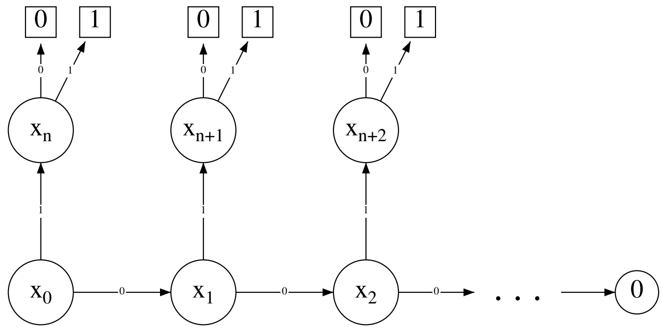
We have written a program to calculate the XDD for p_{n-1} with $n = 1\cdots 9$. We found that the smallest graph was obtained by ordering the variables as $a_0a_1\cdots a_{n-1}b_{n-1}\cdots b_1b_0$ or the same ordering with *a, b* exchanged. The widest part of the graph is reached roughly after $3n/2$ variables, with the width:-

n	1	2	3	4	5	6	7	8	9
width	3	5	8	13	24	58	154	490	1588

We can see from this data that the tree grows exponentially, at the rate of less than 4. We conjecture that these observations are generally true.

6 Differences to BDD

The compact XDDs represent a class of functions distinct from those functions with a compact BDD. We can quite easily construct a function with a compact XDD whose BDD is exponential. The reason is that a list of nodes needs to be kept in order to evaluate an xor graph. The value of the function is the xor of the functions of the nodes on the list. Now, as all the different valuations of a subset of the variables are taken, so the list can range over all subsets of the nodes at a particular level , thus forcing an exponential size BDD. Consider the following XDD of $2n$ variables $x_0\cdots x_{2n-1}$



Now an assignment of truth values to $x_0 \cdots x_{n-1}$ results in the function reducing to $\bigoplus x_{n+i}$ where the i correspond to those x_i assigned to one. These are all distinct functions. Thus the BDD with the same ordering has 2^n nodes on the n-th level.

7 Conclusions

We have presented a new representation of boolean functions via xor graphs and shown that it has some of the useful properties of binary decision diagrams of [1], eg each function has a unique graph. However there are some conspicuous shortcomings, chiefly that the boolean operations of AND, OR may result in exponential graphs. In spite of this the method does have its uses, eg functions related to additions have as simple xor graphs as their BDDs. In particular we have applied our technique to the integer multiplier and found that the graph grows much more reasonably than is the case with the BDDs. We have also noted that the class of functions represented compactly by xor graphs is distinct from the compact BDD class.

References

[1] R. E. Bryant, Graph-Based Agorithms for Boolean Function Manipulation. *IEEE Transactions on Computers, Vol C-35:8, 1986, pp 677 - 691*

[2] R. E. Bryant, On the Complexity of VLSI Implementations and Graph Representations of Boolean Functions with Applications to Integer Multiplication. *IEEE Transactions on Computers, Vol 40:2, 1991, pp 205 - 213*

[3] IMEC-IFIP International Workshop on Applied Formal Methods For Correct VLSI Design. November 1989, Belgium.