Incremental Reduction of Binary Decision Diagrams

R. Jacobi, N. Calazans, C. Trullemans.
Université de Louvain
Laboratoire de Microélectronique - Place du Levant, 3
B1348 Louvain-la-Neuve Belgium
Tel: (32) 10 472583 Fax: (32) 10 452272

Abstract

Binary Decision Diagrams (BDDs) became popular after the ordering constraints proposed by Bryant [1]. BDDs can provide compact canonical representations of Boolean functions. BDDs have been widely used in circuit testing, verification and synthesis [1 2 3 4]. An associated problem is to find good variable orderings to avoid the exponential explosion of the number of nodes and also to speed up the algorithms that manipulate BDDs. Some heuristic techniques are emploied to find good initial orderings based on the circuit structure. Usually, after the complete BDD is built no further reduction is tried. We present here a new method to reduce the number of nodes of a BDD *after* its construction, which we call incremental reduction. It is based on an algorithm that swap two adjacent variables in the BDD ordering. The initial ordering is obtained by an algorithm based on [2]. Our experiments have shown that an average reduction of 20% on the number of nodes of these initial BDDs can be achieved with our method.

1. Introduction

Binary Decision Diagrams are acyclic graphs obtained by applying an ordering constraint over the input variables and reduction operations on a Binary Decision Tree, as proposed by Bryant [1]. In figure 1 we exemplify the tree and diagram for a function f = ab + ac + bc. The ordering consists in associating to each level of the tree a unique input variable. Each variable has an index that identifies its level in the BDD.

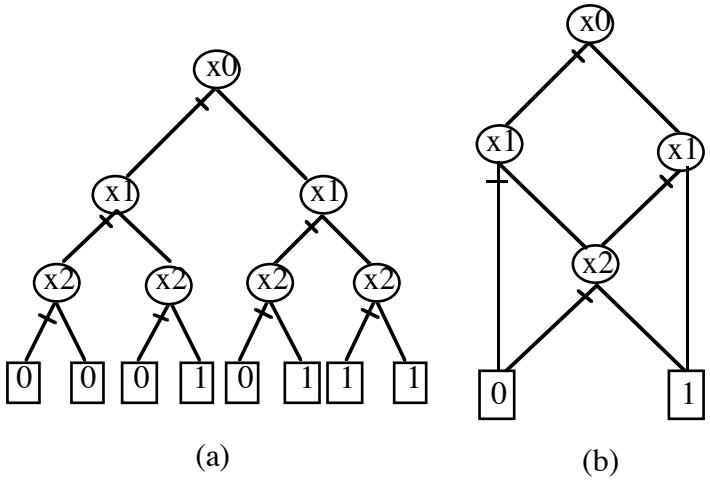

Figure 1. (a) binary decision tree (b) binary decision diagram

Reduction operations consist in eliminating redundant nodes from the binary tree. A node can be eliminated if

a. its low and high sons are equivalent
b. there is another node in the graph with the same index and with equivalent low and high sons, respectively. (1)

A BDD can be viewed as a compact representation of a truth-table. The value of the function for a given input vector is obtained by traversing the diagram, starting at the root node and taking either the low or the high son if the value of the variable is 0 or 1, respectively, up to reach a terminal node. Here we use a graphical convention in which the low branch is identified by a small mark near the node.

In section 2 we present some basic definitions. In section 3 we describe the heuristics to find a good initial ordering. Section 4 shows the incremental reduction method and algorithms. In section 5 we present some results and section 6 is the conclusion.

2. Definitions

Some concepts used in this paper are informally defined in this section.

3. Initial Ordering

The input to our system can be either an Espresso-like PLA description or a set of Boolean equations. Algebraic division techniques are used to produce a Boolean network multi-level representation of the circuit. Over this Boolean network we run a heuristic algorithm based on [2] to find a good initial ordering. In [2] a very simple algorithm is proposed that traverses the circuit structure in a depth first way and put the variables found in a list. An auxiliary heuristic is to put an input which fans-out to more than one gate before the set of inputs currently processed. This heuristic has produced good results, but is too much dependent on the circuit traversing order. We have tried to circumvent this dependency while improving the algorithm efficiency. We built lists of related (through the transitive fan-in) variables and apply some priority rules to concatenate them in a global ordering. A more detailed description of the algorithm can be find in [5].

4. Incremental Reduction

The basic idea is to find new orderings by swapping two adjacent variables in the BDD variable ordering. The new ordering can lead to a smaller diagram. The incremental reduction is divided into two main operations: *swap* and *local-reduction*. It has a local nature: only the nodes at the swapping levels can be eliminated, as enunciated by the following theorems.

Theorem 1: Let **F** be a Boolean function of the input variables $X = \{x_0, x_1, \ldots, x_{n-1}\}$. Let B_1 be a BDD of **F**, constructed using the input variables ordering $O_1 = <a_0, a_1, \ldots, a_{n-1}>$, where $\{a_i\}$ is a permutation of $\{x_i\}$. Let $O_2 = <a_0, a_1, \ldots, a_i, a_{i-1}, \ldots, a_{n-1}>$ be another ordering obtained from O_1 by exchanging two adjacent variables a_{i-1}, a_i. The BDD B_2 of **F**, corresponding to this new ordering, differs from B_1 only at levels i and i+1.

Proof: in the final paper.

Theorem 2: Let B_2 be the BDD obtained by the the application of the swap function on indices i and i+1 of a reduced BDD B_1. It can be shown that *only the nodes at level i+1* can be eliminated by the application of the reduction operations (1) *after* swapping.

Proof: in the final paper.

4.1 Swap

To explain how swap can be implemented, we will analyze all the possible subgraphs configurations in levels i and i+1 and its respective transformations.

The swap is based on the independence of the cofactor operation with respect to the order of the cofactored variables. The BDD can be built by successively cofactoring the function up to hit a terminal value. Consider the function $f(X_n)$ where $X_n = \{x_0, x_1, ... x_{n-1}\}$. We can build the BDD by successively calculating f_{x0}, f_{x0x1}, etc. Considering that $f_{x0x1} = f_{x1x0}$, we can verify the equivalence between subgraphs of figure 3. The subgraphs represent possible configurations of nodes in the swapping levels i, i+1. In short, the transformations consist in creating new nodes and exchanging some sons between the nodes.

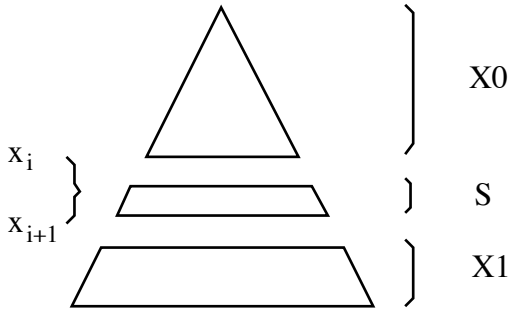

Figure 2. BDD partition.

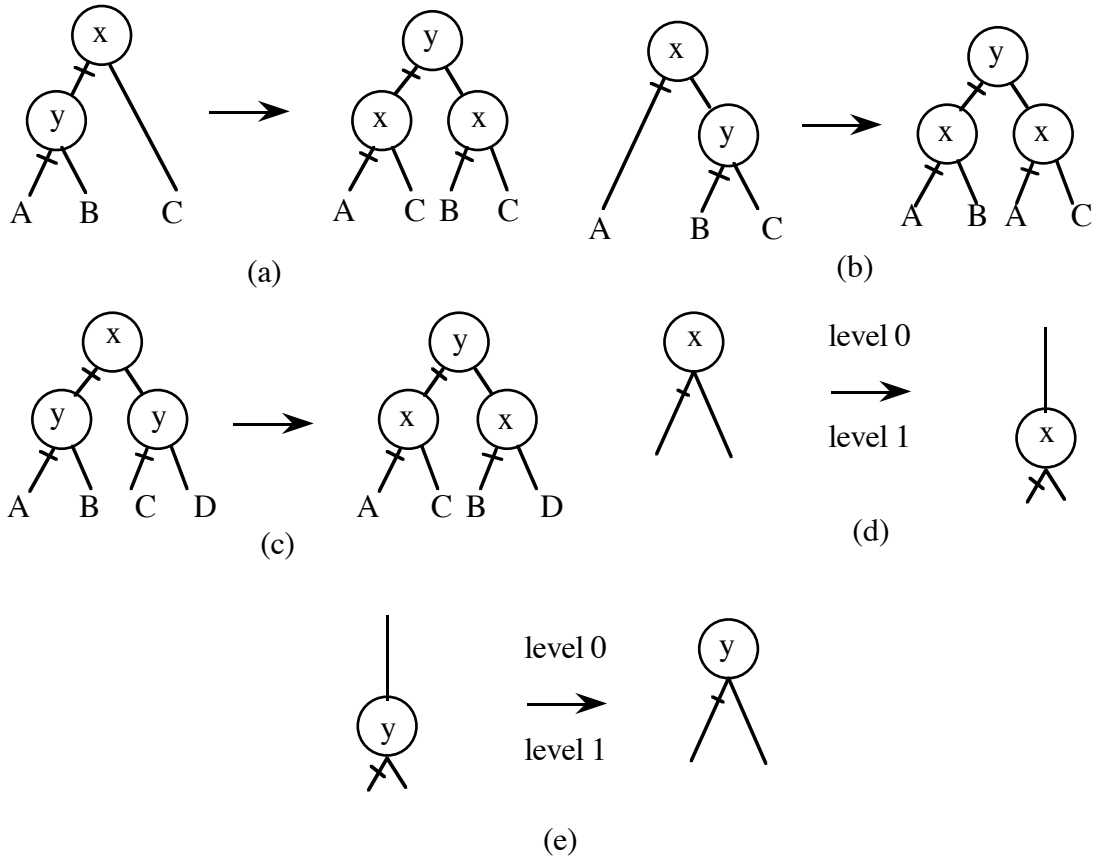

Figure 3. Swapping cases for the subgraphs

4.2 Local Reduction

After swapping the variables we may have redundant nodes that must be eliminated. As the only nodes that can be eliminated are those from level i+1, we developed an incremental reduction function based on [1] that is applied to only one level. Redundant nodes can be generated at level i+1 due to the subgraphs rearrangement produced by the swap. The algorithm looks for nodes with equivalent low/high sons and for isomorphic subgraphs. It builds a list of keys and associates each key to a unique node. A list with the redundant nodes is built and these nodes are replaced by its equivalent nodes in the BDD.

4.3 Algorithms

We present here the two basic algorithms developed to implement the incremental reduction operation: *swap-index* and *local-reduce*. A pseudo-PASCAL code for these algorithms is shown in figures 4 and 5. We give detailed explanations in the final paper.

4.4. Heuristics

Two basic heuristics are employed when trying to reduce BDDs with swap: *swap-all-red* and *swap-run-down*. *Swap-all-red* evaluate first all possible swappings without changing the MBD, and execute the swapping at the level which produces the greater reduction. This procedure is repeated until no more reduction is achieved. *Swap-run-down* start at level 0 and execute all the swappings that does not increase the BDD. The pseudo-code for these algorithms and more details in the final paper. We will describe also some improvements that have increased the throughput of the heuristics by one order of magnitude.

5. Results

In our multi-level logic synthesis system we have extended the BDDs to represent multiple-output incompletely specified functions, which we called Modified Binary Diagrams [5]. The incremental reduction method has been extended to deal with MBDs and we have obtained comparable results.

We have run the swap heuristics on a set of benchmark examples generated with MisII [6]. These results were obtained in a Mac II computer, the algorithms programmed in common LISP. The programs where also ported to a Sun 3/80 and a Microvax 3600, where similar results were obtained. The results of the examples are presented in Table 1. The initial ordering obtained by a LISP version of the algorithm proposed by [2] and by our extension is shown in the first two columns. The results for the incremental reduction refers to the orderings obtained by our algorithm. An asterisk indicate a multiple output circuit. We can verify an average reduction of 20 per cent for the MBDs and 25 per cent for the BDDs size.

6. Conclusion

We have presented a method that allows the incremental reduction of the BDDs which in most of the cases results in a significant reduction of the global BDD size. This reduction allows several algorithms that manipulate BDDs, like Boolean verification, to run faster. As multi-level logic minimization systems spend most of the time running this kind of algorithms [4], this reduction can represent a significant improvement in processing time.

```
function swap-index (i0,i1: index)

var     first,          {list of nodes with index = i0}
        second,         {list of nodes with index = i1}
        fathers,        {list of fathers of a node}
        gfathers:       {list of grandfathers of a node}
                list;

begin
        first := get-brothers (Brothers-Array, i0);     {take all nodes with index i0}
        second := get-brothers (Brothers-Array, i1); {take all nodes with index i1}
        for each node n in second
        begin
          fathers := get-fathers (n, Fathers-Table);
          gfathers := {n in fathers | index (n) < i0}
          fathers := fathers - gfathers;
          if (gfathers # null  or  is-output (n))
            then
                let the original node to the gfathers or to the output function
              else {let the original node to the first father}
                   fathers := remove-first-element(fathers);
          for each f in fathers
           begin
             duplicate-node (n);
             update MBD, Fathers-Table, Brothers-Array and MBD-size;
           end
       end
      for each n in first
      begin
         il := index(low(n));
         ih := index(high(n));
         if (il=ih)
            then if (ih = i1)           {n has low and high sons at i1}
                        then  begin
                            exchange high(low(n)), low(high(n));
                            update Fathers-Table;
                        end
                        else  begin     {no high son at i1}
                            create a new node and place it at high(n);
                            update Fathers-Table, Brothers-Array, MBD-size;
                        end
              else if (ih = i1)         {n has no low son at i1}
                     then   begin               {n has high son at i1
                         create a new node;
                         place it at low(n);
                         update Fathers-Table, Brothers-Array and MBD-size;
                     end
                     else               {no sons at i1}
                         change n from level, update Brothers-Array;
      end
      update variable order in MBD-order;
end
```

Figure 4. SWAP-INDEX algorithm.

```
function  inc-reduce (i: index)
```

```
var redundat-nodes,     {nodes to be eliminated}
    reduced-lis,            {nodes that will stay}
    Keys,                   {list of node keys}
    node-lis:   list;       {nodes at level i}

begin
   node-lis := get-nodes (Brothers-Array,i);
   for each n in node-lis
   begin
      if (low(n) = high(n))
         then redundant-lis := append (<n,low(n)>, redundant-lis)
         else if (find (<low(n), high(n)> , Keys)    {isomorphic subgraphs}
                then  begin
                   new-node := get isomorphic subgraph root from Keys;
                   redundant-lis := append (<n, new-node>, redundant-lis);
                end
   end;
   reduced-lis := node-lis - redundant-lis;
   update Brothers-Array, MBD-size;
   for each pair <n,newn> in redundant-lis
      for each father in get-parents(n, Fathers-Table)
         begin
            replace n by newn in father;
            update Father-Table;
         end
   regenerate node identifiers in MBD;
end
```

Figure 5. INC-REDUCE algorithm.

Circuit	Fuji	Our	Swap	Reduction%
Duke2	132	70	52	25
X9DN	98	86	49	43
SN181	247	211	192	9
Signet	1857	2053	1491	27
Duke2*	895	463	427	8
Misex2*	147	152	115	24
P1*	275	269	209	22
Risc*	110	112	98	13
Signet*	3340	3611	2418	33
SN181	847	842	723	14
X9DN*	471	352	248	30

Table 1. Results of incremental reduction

7. References

[1] R. E. Bryant. "Graph-Based Algorithm for Boolean Function Manipulation". In: *IEEE Trans. on Computers,* vol. C-35, no. 8, Aug. 1986.

[2] M. Fujita, H. Fujisawa, N. Kawato. "Evaluation and Improvements of Boolean Comparison Method Based on Binary Decision Diagrams", In *Proceedings of the IEEE International Conference on Computer-Aided Design,* 1988.

[3] S. Malik, A. R. Wang, R. K. Brayton and A. Sangiovanni-Vincentelli. "Logic Verification using Binary Decision Diagrams in a Logic Synthesis Environment". In *Proceedings of the IEEE International Conference on Computer-Aided Design,* 1988.

[4] Y. Matsunaga, M. Fujita. "Multi-Level Logic Optimization Using Binary Decision Diagrams". In: *IEEE International Conference on Computer Aided Design,* November, 1989.

[5] N. Calazans, R. Jacobi, C. Trullemans. "Modified Binary Diagrams: an approach to the manipulation of multiple output incompletely specified functions". Submitted to ISCAS 91.

[6] R. K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli and A. Wang. "MIS: A Multiple-Level Logic Optimization System", *IEEE Trans. on Computer-Aided Design,* vol. CAD-6, no. 6, Nov. 1987.