2.3.2 BDDs

Reduced Ordered Binary Decision Diagrams (ROBDDs) provide a canonical representation of Boolean functions. They often are more compact that other canonical representations like Sum-Of-Products for instance. The presentation about BDD is based on Chapter 5 from the textbook by Clarke, Grunberg, and Peled [3].

Constructing BDDs

The fundamental notion underlying this section is Shannon's expansion. For a Boolean function f and a given variable x of f, it states that function f can be decomposed into two sub-functions considering the cases where either x is false or x is true. Formally, we have the following equation:

$$f = (\neg x \wedge f|_{x=0}) \vee (x \wedge f|_{x=1})$$

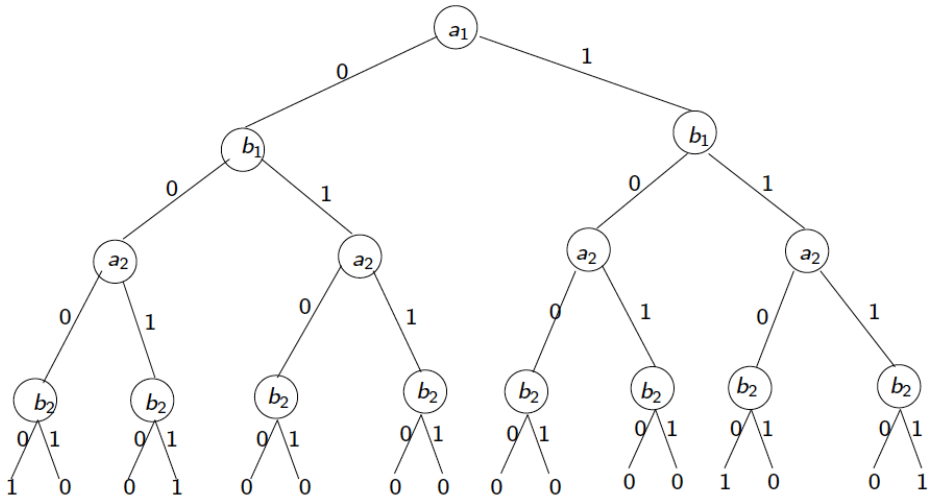

Figure 2.1: Binary decision tree for two-bit comparator.

Using Shannon's decomposition, it is possible to build a binary decision tree. Figure 2.1 shows the tree for a two-bit comparator, that is, for the following Boolean function:

$$f(a_1, a_2, b_1, b_2) = (a_1 \leftrightarrow b_1) \wedge (a_2 \leftrightarrow b_2)$$

A binary tree basically is a rooted, directed tree with terminal and nonterminal vertices. Each nonterminal vertex v is labelled by a variable noted $var(v)$. Such a vertex has two successors: a successor corresponding to the case where the variable is low (false) noted $low(v)$ and a successor corresponding to the case where the variable

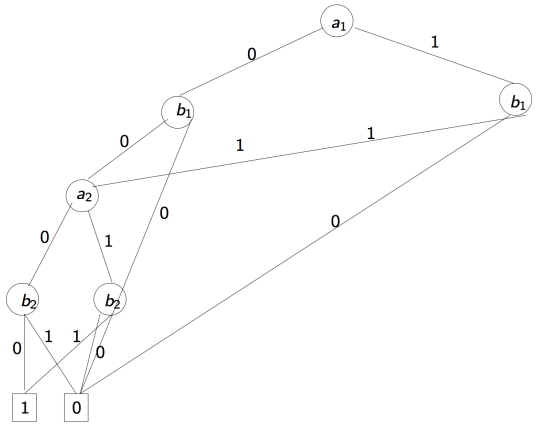

Figure 2.2: ROBDD for two-bit comparator.

is high (true) noted $high(v)$. Graphically, we often identify the successor by a 0 or 1 label on the edges. Each terminal vertex v is either 0 or 1. To know the truth value of formula for a given assignment, one traverses the tree down to a leaf by following the low successor if a variable is false or the high successor if a variable is true. As one can see, such a tree is not a very compact representation.

Bryant [2] showed how to obtain a compact and canonical representation of Boolean functions by placing two restrictions on binary diagrams. The first restriction imposes an ordering among the variables. This means that variables must always appear in the same order along all paths from the root to terminals. Second, there should be no isomorphic sub-trees or redundant vertices in the diagram. To achieve the first restriction, a total order $<$ is imposed among the variables. We require that if any vertex u has a nonterminal successor v, $var(u) < var(v)$. The second restriction is achieved by recursively applying the following rules:

- *Remove duplicate terminals:* Keep only one terminal vertex for each label (0 or 1) and redirect all edges to eliminated vertices to the remaining ones.

- *Remove duplicate nonterminals:* If two nonterminals vertices u and v are labelled with the same variables, that is, $var(u) = var(v)$, eliminate one of the vertices and redirect edges to the eliminated vertex to the remaining one.

- *Remove redundant tests:* If the two successors of a nonterminal vertex v are equal, that is, $low(v) = high(v)$, remove v and redirect incoming edges of v to $low(v)$.

Starting with an ordered diagram, the reduced version is obtained by applying these rules until the size can no longer be reduced. We then obtained a Reduced Ordered Binary Decision Diagram (ROBDD). Bryant [2] showed that this reduction can be done in a bottom-up manner and in time that is linear in the size of the original BDD.

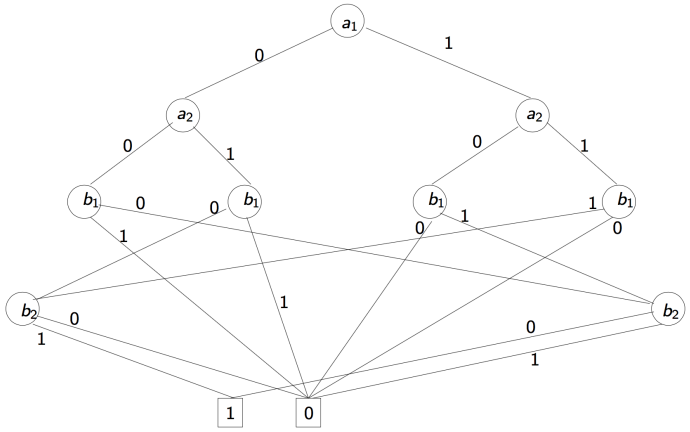

Figure 2.3: ROBDD for two-bit comparator.

Figure 2.2 shows the ROBDD obtained from the decision tree in Figure 2.1. This BDD has only 6 nodes.

The variable ordering has a strong impact on the size of the ROBDD. For instance, choosing the ordering $a_1 < b_1 < a_2 < b_2$ the 2-bit comparator results in the ROBDD shown in Figure 2.3, which is larger than the one shown in Figure 2.2. This larger BDD has 9 nodes.

ROBDDs are canonical given a variable ordering. This means that checking equivalence between two ROBDDs reduce to checking equality between two diagrams. Validity reduces to showing equivalence to the trivial diagram composed of one terminal labelled by 0.

To further illustrate the construction of BDDs we consider the following Boolean function:

$$f(a, b, c) = a \vee (b \wedge c)$$

We will use Shannon's expansion to build a Binary Decision Tree for this Boolean function. To apply the expansion we need to determine an ordering of the variables. Let us consider the ordering $a < b < c$.

We first decompose according to a, this gives the following equation:

$$f(a, b, c) = (\neg a \wedge f(0, b, c)) \vee (a \wedge f(1, b, c))$$

The first part of the tree from this equation is pictured in Figure 2.4.

We can recursively proceed with decomposing each sub-formula. For instance, we can expand the subtree $f(0, b, c)$ and obtain:

$$f(0, b, c) = (\neg b \wedge f(0, 0, c)) \vee (b \wedge f(0, 1, c))$$

A similar expansion is obtained for $f(1, b, c)$. We then insert these terms in the tree and obtain the tree pictured in Figure 2.5.

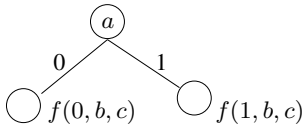

Figure 2.4: First part of the binary decision tree.

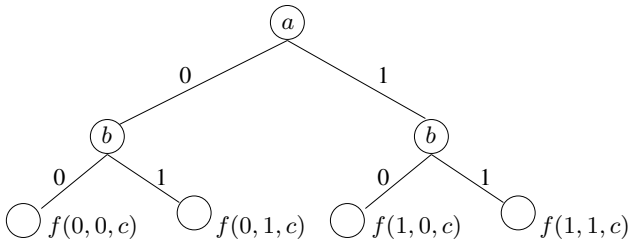

Figure 2.5: The binary decision tree continued ...

We proceed with the four leaves and then compute the values of f where all literals have been assigned a definite value. We obtain the Binary Decision Tree pictured in Figure 2.6.

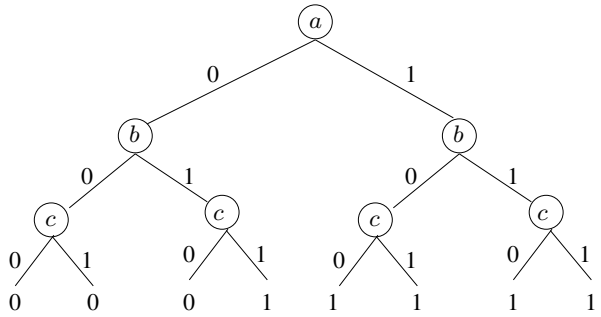

Figure 2.6: The binary decision tree

We apply the rules proposed by Bryant [2] to transform this BDT into a Reduced Ordered BDD. We first apply the rule "remove redundant tests"[1], this means that we eliminate all nodes where the value at the end of the left and the right branches is the same. There are three cases where this applies. We obtain the graph in Figure 2.7.

The same rule applies once more to the true branch of a. After applying this rule, we apply the rule "remove duplicate terminals" and obtain the ROBDD in Figure 2.8.

The size of a ROBDD depends on the variable ordering. The following question illustrates this point.

[1]The order in which the rules are applied does not matter.

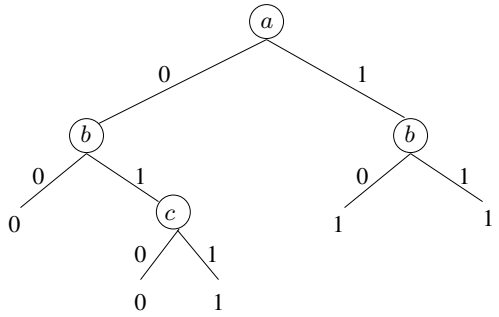

Figure 2.7: Reducing the BDT with "remove redundant tests".

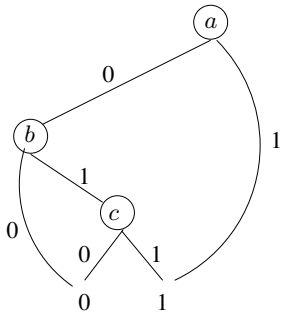

Figure 2.8: Reducing the BDT to a ROBDD.

Exercise 2.3.2. For which ordering(s) is the size of the ROBDD for our function f maximal? (Note: The size is counted in the number of nodes and edges)?

Finding an optimal ordering is infeasible and there exist functions with no optimal ordering, like bitwise multiplication. In such cases, explicit algorithms might perform better than symbolic ones.

Operations on BDDs

The main advantage of ROBDDs is that they can be manipulated efficiently. In particular, checking that two ROBDDs are equal requires constant time. Negation is also very easy and simply requires to negate the two leaves[2] of a ROBDD.

Other operations are performed using the Apply algorithm proposed by Bryant [2] and that can be used to compute any 2-argument logical operation. The main idea is to use Shannon's decomposition to break problems into sub-problems.

Let $\star$ be an arbitrary 2-argument logical operation. Let f and f' be two Boolean functions. Let v be the current vertex of f and v' be the current vertex of f'. The algorithm basically is the following case distinction:

[2]Note that any ROBDD has only two leaves, namely, 1 for "true" and 0 for "false".

- If v and v' are both terminals, $f \star f' = value(v) \star value(v')$.
- If $var(v) = var(v')$, apply Shannon's expansion:

  $$f \star f' = \neg var(v) \wedge (f|_{var(v)\leftarrow 0} \star f'|_{var(v)\leftarrow 0}) \vee var(v) \wedge (f|_{var(v)\leftarrow 1} \star f'|_{var(v)\leftarrow 1})$$

  to break the problem into subproblems. The resulting ROBDD will have a new node w with $var(w) = var(v)$, $low(w)$ will be the ROBDD for $(f|_{var(v)\leftarrow 0} \star f'|_{var(v)\leftarrow 0})$ and $high(w)$ will be the ROBDD for $(f|_{var(v)\leftarrow 1} \star f'|_{var(v)\leftarrow 1})$.
- If $var(v) < var(v')$, $f'|_{var(v)\leftarrow 0} = f'|_{var(v)\leftarrow 1} = f'$ since f' does not depend on $var(v)$. In this case, the Shannon's expansion reduces to:

  $$f \star f' = \neg var(v) \wedge (f|_{var(v)\leftarrow 0} \star f') \vee var(v) \wedge (f|_{var(v)\leftarrow 1} \star f')$$

- If $var(v') < var(v)$, similar to previous case.

By reducing and caching intermediate results, the algorithm is polynomial. The number of sub-problems is bounded by the product of the size of the ROBDDs for f and f'.

We illustrate the *Apply* algorithm using our small example:

$$f(a) \vee f'(b, c)$$

where $f(a) = a$ and $f'(b, c) = b \wedge c$.

We consider the ordering $a < b < c$. We assume we have the ROBDDs of each function, that is, the graphs pictured in Figure 2.9.

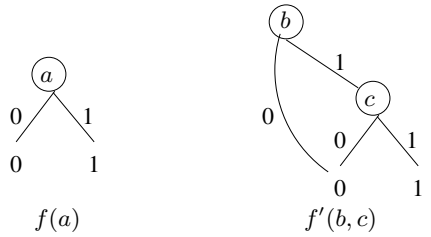

Figure 2.9: ROBDDs for f and f'.

We now use Apply and Shannon's expansion to compute the disjunction of these two ROBDDs. Since the roots of the ROBDDs are not terminal vertices and since the variables in the two roots are not the same we need to determine which of the two variables comes first in the ordering. Since we consider the ordering where $a < b$, we apply the formula from the third case of the aforementioned algorithm:

$$f(a) \vee f'(b, c) = (\neg a \wedge (f(0) \vee f'(b, c))) \vee (a \wedge (f(1) \vee f'(b, c)))$$

We now start to build a new ROBDD as illustrated in Figure 2.10.

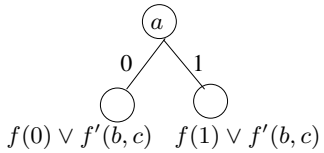

Figure 2.10: Partial ROBDDs for $f \vee f'$.

Now, we have $f(0) = 0$ and $f(1) = 1$ (since $f(a) = a$). Therefore, the left branch simplifies to $f'(b, c)$ and the right branch to 1. We already have a ROBDD for $f'(b, c)$. We simply insert it as the left branch. Finally, we obtain the ROBDD pictured in Figure 2.8 for the formula:

$$f(a, b, c) = a \vee (b \wedge c)$$

The last operation we will need on ROBDDs is existential quantification. Assume a Boolean function f and x a variable of f. Using Shannon's decomposition, we have:

$$\exists x.f = \neg x \wedge f|_{x \leftarrow 0} \vee x \wedge f|_{x \leftarrow 1}$$

That is, f is true if either f is true when x is false, or f is true when x is false. We will come back to existential quantification in Section 3.2.1 of Chapter 3.