Lecture Notes for

Logic and Computability

Course Number: IND04033UF

Contact

Bettina Könighofer
Institute for Applied Information Processing and Communications (IAIK)
Graz University of Technology, Austria
bettina.koenighofer@iaik.tugraz.at

Graz University of Technology

Table of Contents

5 Binary Decision Diagrams **3**

5.1 Binary Decision Diagram 3

5.2 Reduced Ordered BDDs 5

5.3 Construction of Reduced Ordered BDDs 10

5 Binary Decision Diagrams

In this chapter we introduce an efficient data structures to store Boolean formulas. One of the most frequently used data structure is called *reduced ordered binary decision diagram* (*ROBDDs*).

5.1 Binary Decision Diagram

Binary decision diagrams (BDDs) are an efficient way to represent Boolean formulas. In this chapter, we start with simple binary decision diagrams, and we will extend them until we reach the data structure of reduced ordered BDDs.

Definition - Directed Acyclic Graph A directed acyclic graph (DAG) is a directed graph that does not have any cycles. A node of a DAG is initial if there are no edges pointing to that node. A node is called terminal if there are no edges out of that node.

Definition - Binary Decision Diagram. A binary decision diagram that represents a Boolean formula f is a DAG with two *terminal nodes* that are labelled with 0 and 1. The *internal nodes* are labelled with the Boolean variables of the formula, e.g., a, b, c, $\ldots$. Each internal node has exactly two outgoing edges: one line that is labeled with a T (*the then-edge*), and the other line that is labeled with an E (*the else-edge*) or *marked with a circle.* There is a unique initial node called the *function node* labeled with f that does not has any ingoing edges and one outgoing edge to the internal variable node on the first level.

Note that within a BDD, nodes can share leaves. Therefore, BDDs are more general than trees.

Example. Figure 5.1 shows a binary decision diagram with four layers of variables a, b, c, and d.

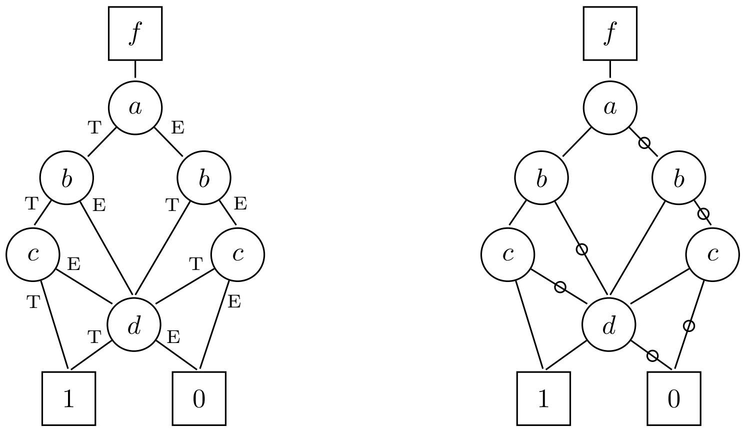

Figure 5.1: A simple BDD with different edge labeling.

Evaluating a model in a BDD

Each binary decision diagram determines a unique boolean formula in the following way. Given a model (an assignment to the variables in the formula), we start at the root of the diagram and take the T line whenever the value of the variable at the current node is *true*, otherwise, we travel along the line marked with E. The truth value of the formula is the value of the terminal node we reach.

Example. Consider the BDD given in Figure 5.1. To which truth value does the formula represented by the BDD evaluates under the given model

$$\mathcal{M} := \{a = \top,\ b = \top,\ c = \top,\ d = \top\}?$$

Solution. We start at the root node and only follow the *then-edges* and end in the terminal node marked with 1. Therefore, the model M is a satisfying assignment.

Constructing formulas from BDDs

To find the formula f in disjunctive normal form that is represented by the BDD, we need to find all paths that end in the terminal node 1. Each path forms a cube of the DNF formula: if we follow the *then-edge* the corresponding variable appears positive in the cube, if we follow the *else-edge*, we use the negated variable in the cube.

Example. Consider the BDD given in Figure 5.1. Construct the formula represented by the BDD.

Solution. The first path that ends in the terminal node 1 is given by always taking the *then-edges.* This path results in the first cube $(a \wedge b \wedge c)$. The second path is given by taking the *then-edges* on level 1 and 2, the *else-edge* on level 3 and the *then-edge* on level 4. This path gives us the second cube $(a \wedge b \wedge \neg c \wedge d)$. By enumerating and encoding all paths into cubes, we end up in the following formula:

$$f = (a \wedge b \wedge c) \vee (a \wedge b \wedge \neg c \wedge d) \vee (a \wedge \neg b \wedge d) \vee (\neg a \wedge b \wedge d) \vee (\neg a \wedge \neg b \wedge c \wedge d)$$

Size of a BDD

If the root of a BDD is an internal node labeled with the variable v then it has two sub-trees: one for the value of v being *true* and another one for v having value *false.* For a Boolean formula f with n variables, in the worst case the corresponding BDD will have $2^{n+1} - 1$ nodes. A truth table would have 2^n lines. *However, binary decision diagrams often contain a lot of redundancy which we can exploit.* This makes it possible to often represent large formulas by relatively small BDDs.

5.2 Reduced Ordered BDDs

From BDD to Reduced BDDs

To reduce the size of the BDD, several optimizations can be exploited.

Optimization 1: Complement attribute and single terminal node. An edge can be negated by marking the edge with a *full circle.* By negating all edges that lead to the terminal node 0, the terminal node 0 can be removed and all edges will be redirected and negated to terminal node 1.

An example BDD using the complement attribute is shown in Figure 5.2.

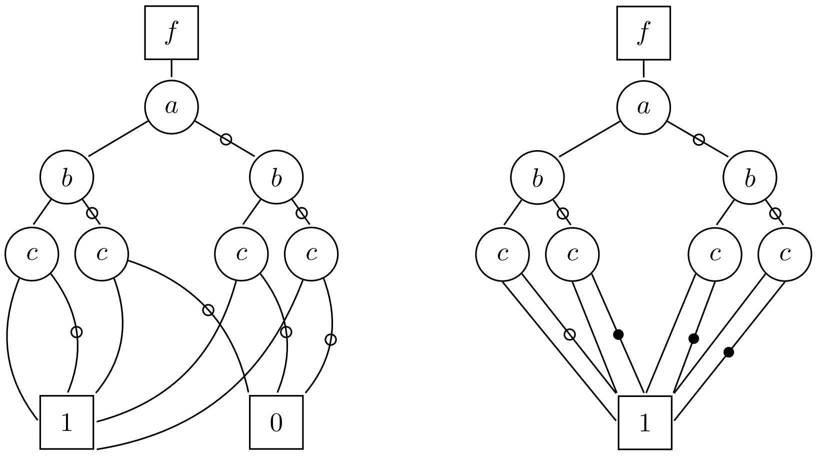

Figure 5.2: Left: Simple BDD. Right: After optimization 1: BDD *with* complemented edges.

Optimization 2 - Dangling edges. Since there is only one terminal node left, there is no need to actually connect the edges leading to the terminal node with the terminal node. We can let the edges from internal nodes from the last level dangling and know implicitly that those edges lead to the node 1. This optimization is just to make it easier to draw a BDD on paper. Figure 5.3 shows the example BDD from above with dangling edges.

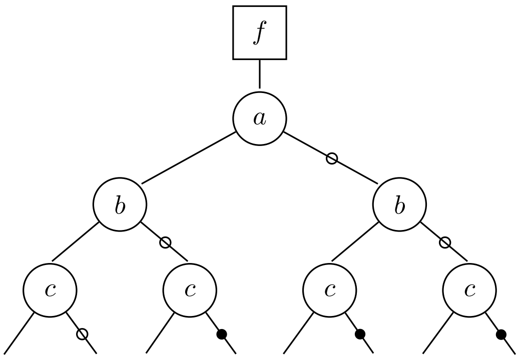

Figure 5.3: Optimization 2: BDD with dangling edges.

Optimization 3 - Removal of duplicate sub-BDDs. If two distinct nodes n and m in the BDD are the roots of structurally identical sub-BDDs, then we eliminate one of them, say m, and redirect all its incoming edges to the other one.

Consider the BDD from Figure 5.3. With the exception of the left most node labeled with c, the other three nodes labeled with c in Figure 5.3 have identical sub-trees. Therefore, applying Optimization 3 results in the BDD shown in Figure 5.4 with two c-nodes removed.

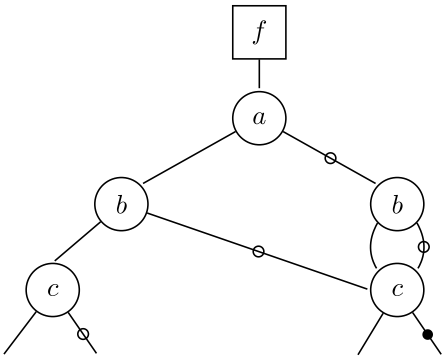

Figure 5.4: Optimization 3: BDD with no duplicate sub-BDDs.

Optimization 4 - Removal of redundant nodes. If both outgoing edges of a node n point to the same node m, then we eliminate that node n, sending all its incoming edges to m.

Figure 5.5 gives examples from nodes that are redundant and can be removed, and notes that are not redundant due to negated edges.

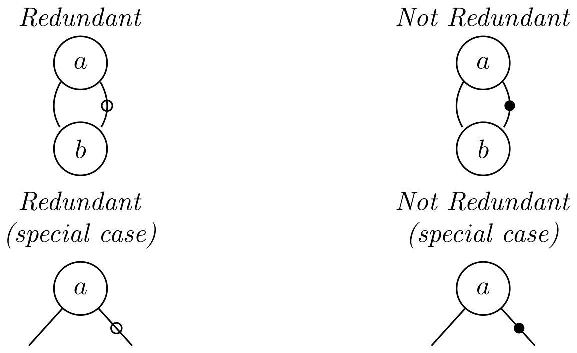

Figure 5.5: Examples of redundant and not redundant nodes.

Let us consider the BDD given in Figure 5.4. The node b on the right subtree is redundant, as its value does not affect the values of paths that go trough it. The same goes for the node c on the left, as both of its outgoing edges are not negated. The result after the reduction is given in Figure 5.6.

Applying the optimizations rules from above exhaustively until no further reductions are possible results in a Reduced BDD.

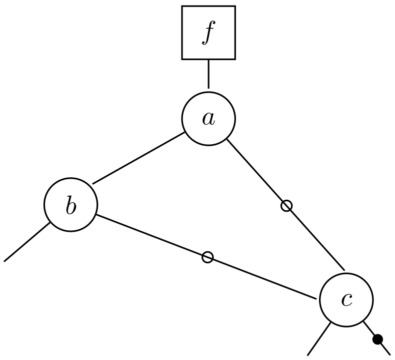

Figure 5.6: Optimization 4: BDD with no redundant nodes.

Ordered BDDs

Having multiple occurrences of a variable along a path seem rather inefficient. Therefore, ordered BDDs impose an ordering on the variables occurring along any path.

Definition - Ordered BDD. Let $[x_1,\ldots,x_n]$ be an ordered list of variables without duplications. We say that a BDD has the ordering $x_1 < \cdots < x_n$ if all variable labels of the BDD occur in that list and, for every occurrence of x_i followed by x_j along any path in the BDD, we have $i < j$. An ordered BDD (OBDD) is a BDD which has an ordering for some list of variables.

It follows from the definition of OBDDs that one cannot have multiple occurrences of any variable along a path. Note that all BDDs we considered so far were ordered.

The impact of the chosen variable ordering. For most examples, the chosen variable ordering makes a significant difference to the size of the ROBDD representing a given boolean formula. The sensitivity of the size of an ROBDD to the particular variable ordering is one of the disadvantages of ROBDDs. Although finding the optimal ordering is itself a computationally expensive problem, there are good heuristics which will usually produce a fairly good ordering. *In this lecture, we consider from now on only BDDs that are **reduced and ordered**, i.e., ROBDDs.*

The commitment to an ordering gives us a unique representation of boolean formulas as ROBDDs.

Theorem: ROBDDs give a *canonical* representations of propositional formulas. This means that for a given variable order, if two formulas f_1 and f_2 are semantically equivalent, then they will be represented via the very same ROBDD.

For example, if you consider the two formulas $f_1 = \big(a \wedge (b \vee c)\big)$ and $f_2 = \big(a \wedge (a \vee b) \wedge (b \vee c)\big)$. The two formulas are syntactically different but semantically equivalent. Therefore, they will be represented by the same ROBDD. This makes it possible to implement equivalence checking using ROBDDs in *constant time.*

Example. Check whether the given models $\mathcal{M}_1$, $\mathcal{M}_2$ and $\mathcal{M}_3$ are satisfying models or falsifying models using the ROBDD given in Figure 5.7.

- $\mathcal{M}_1: \; a = \top, b = \bot, c = \top, d = \top$
- $\mathcal{M}_2: \; a = \bot, b = \top, c = \bot, d = \bot$
- $\mathcal{M}_3: \; a = \top, b = \top, c = \bot, d = \bot$

Solution. $\mathcal{M}_1 \nvDash f$, $\mathcal{M}_2 \vDash f$ and $\mathcal{M}_3 \vDash f$.

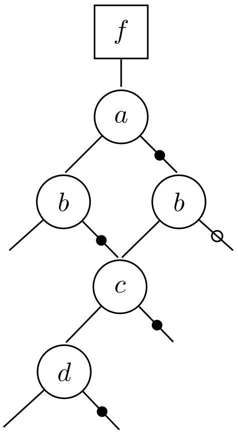

Figure 5.7: ROBDD

Constructing Formulas from ROBDDs

To find the formula in DNF that is represented by the ROBDD, we again search for all paths which corresponding assignment make the formula true. *An assignment makes the formula true, if its corresponding path has an **even number of negations** (full circles).*

Example. Consider the ROBDD in Figure 5.8. Give the formula that is represented by the ROBDD.

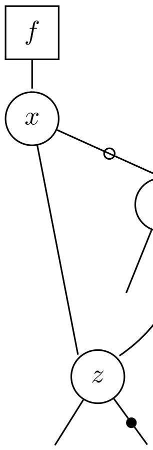

Figure 5.8: Reduced Ordered BDD.

Solution. The first path follows the else-edge from x and the then-edge from y. This path has no negations and is therefore accepting. The path that follows the else-edge from x and y and the the then-edge from z has one negation. Therefore, the corresponding assignment does not satisfy the formula. If we enumerate all *paths with an even number of negations* and transform them to cubes, we obtain the following formula in DNF:

$$f := (\neg x \wedge y) \vee (\neg x \wedge \neg y \wedge \neg z) \vee (x \wedge z).$$

5.3 Construction of Reduced Ordered BDDs

We discuss how to construct a ROBDD directly from a given formula. The algorithm for the construction of ROBDDs is based on the cofactors of the given formula.

Definition - Cofactor. The cofactors are derived from the formula f by substituting truth values for the propositional variables. For a formula f, $f_x = f[x \leftarrow 1]$ and $f_{\neg x} = f[x \leftarrow 0]$ are the **positive** and **negative** cofactors of f with respect to the variable x.

Example. Compute the positive and the negative cofactor w.r.t. the variable x for the following formula:

$$f = (x \wedge y) \vee (\neg x \wedge z).$$

Solution. Setting x to true results in the positive cofactor

$$f_x = (\top \wedge y) \vee (\bot \wedge z) = y.$$

Setting x to false gives us the negative cofactor

$$f_{\neg x} = (\bot \wedge y) \vee (\top \wedge z) = z.$$

Algorithm to Construct a ROBDD

Step 1. Compute all cofactors.

In the first step, recursively compute all cofactors with respect the given variable order. If a cofactor matches a cofactor or the negation of a cofactor that you have seen before, note that the two cofactors are the same and backtrack. If you ignore this step, your BDD that you construct will not be reduced.

For example, if $a < b < c < d$, we first compute f_a with $f_a = f[a \leftarrow 1]$. Next, we compute f_{ab} with $f_{ab} = f_a[b \leftarrow 1]$. Next, we compute f_{abc} with $f_{abc} = f_{ab}[c \leftarrow 1]$ and so on.

Step 2. Draw BDD from cofactors.

We draw a ROBDD, such that each node represents a cofactor. For root node (e.g., labeled with a) of the BDD represents the entire formula f. We can therefore connect it with a function node labeled with f. The internal node that we reach by taking the *then-edge* represents the *positive cofactor* f_a. The internal node that we reach by taking the *else-edge* represents the *negative cofactor* $f_{\neg a}$.

We construct the BDD such that one node representing a cofactor leads us to the next one in the recursive computation. If a cofactor resolves to true, we draw a danling edge. If a cofactor resolves to false, we draw a negated dangling edge. If a cofactor is equivalent to a cofactor we have already seen, we draw the edge to the node that represents this cofactor.

Step 3. Shift negations upwards.

In the final step, we follow the convention that *complemented edges marked with full circles are only allowed for else-edges.* However, the execution of step 2 might cause complemented dangling then-edges. To remove the negations on the then-edges of the BDD, we shift illegal negations upwards.

To do so, if a then-edge is negated, *we consider the origin node of the then-edge and negate all its incoming and outgoing edges.* If an edge gets two negations in this process, they cancel each other out. If one of the outgoing edges that got negated was a then-edge, we consider the origin node of this then-edge and negate all its incoming and outgoing edges. This process shifts the negations upwards until the root node is reached.

Example. Consider the following formula $f = (a \wedge b \vee \neg a) \wedge \neg c \wedge d \vee c$ and the variable order $a < b < c < d$. Construct a ROBDD that represents f.

Solution. *Step 1. Compute all cofactors.* Note that $f_{nega} = f_{ab}$.

$$\begin{aligned}
f &= (a \wedge b \vee \neg a) \wedge \neg c \wedge d \vee c\\
f_a &= b \wedge \neg c \wedge d \vee c\\
f_{ab} &= \neg c \wedge d \vee c\\
f_{abc} &= \top\\
f_{ab\neg c} &= d\\
f_{ab\neg cd} &= \top\\
f_{ab\neg c\neg d} &= \bot\\
f_{a\neg b} &= c\\
f_{a\neg bc} &= \top\\
f_{a\neg b\neg c} &= \bot\\
f_{\neg a} &= \neg c \wedge d \vee c = f_{ab}
\end{aligned}$$

Step 2. Draw BDD from cofactors. The BDD is shown in Figure 5.9.

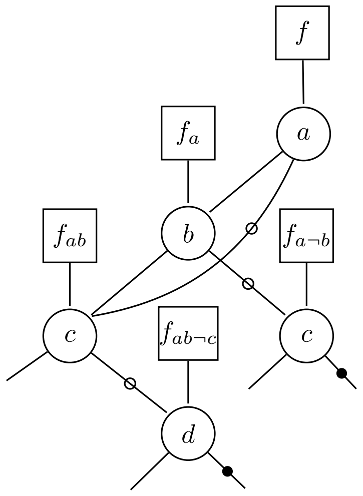

Figure 5.9: ROBDD.

Step 3. Shift negations upwards. Since there are no negated then-edges, there is nothing more to be done and Figure 5.9 represents the final ROBDD.

Example. Consider the following formula $f = (a \wedge \neg c) \vee \big(\neg a \wedge (b \vee (\neg b \wedge c))\big)$ and the variable order $a < b < c$. Construct a ROBDD that represents f.

Solution. *Step 1. Compute all Cofactors.*

$$\begin{aligned}
f &= (a \wedge \neg c) \vee \big(\neg a \wedge (b \vee (\neg b \wedge c))\big)\\
&f_a = \neg c\\
&\quad f_{ac} = \bot\\
&\quad f_{a\neg c} = \top\\
&f_{\neg a} = b \vee (\neg b \wedge c)\\
&\quad f_{\neg ab} = \top\\
&\quad f_{\neg a \neg b} = c = \neg f_{ab}
\end{aligned}$$

Note that $f_a = \neg c$. The variable b does not appear in f_a and setting b to a certain value will not have an effect in f_a. Therefore, we skip the cofactor of b and immediately compute the f_{ac} and $f_{a\neg c}$.

Step 2. Draw BDD from Cofactors. The BDD is given in Figure 5.10.

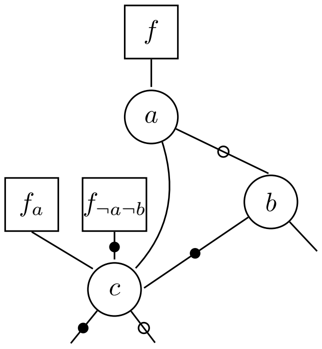

Figure 5.10: ROBDD with illegal compliments.

Step 3. Shift negations Upwards. The ROBDD in Figure 5.10 has an illegal complement, i.e., the then edge of the internal node c is negated. Therefore, we negate all edges of the c node, which double-negates the illegal complement and makes it *true* again. This way, the problem is pushed upwards and has to be repeated, until no then-edge is left negated. By removing the negation of the then-edge of c, we create a negation at the then-edge from a. This is illustrated in Figure 5.11. To remove this negation, we repeat the same procedure with the edges of a. The result and final ROBDD can be seen in Figure 5.12.

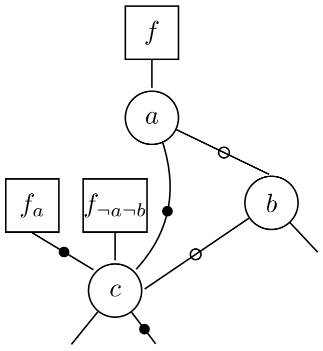

Figure 5.11: Intermediate step: ROBDD with updated complements.

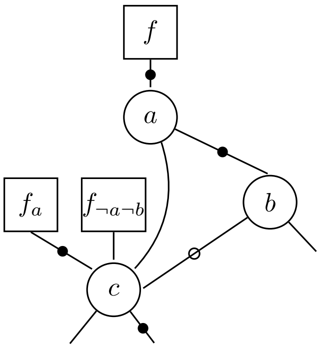

Figure 5.12: Final ROBDD.