Lecture Notes for

Logic and Computability

Course Number: IND04033UF

by

Bettina Könighofer
Bernd Grabner
Belinda Uhl

Contact

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

March, 2021

Graz University of Technology

Table of Contents

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, ... 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 not trees and are more general.

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

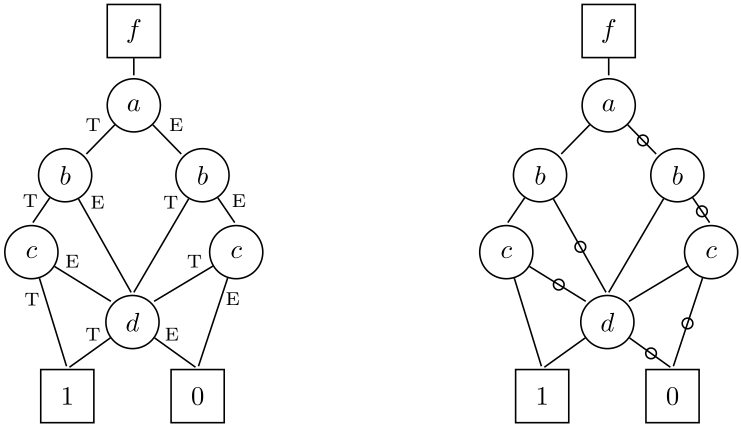


Figure 5.1: A simple BDD with different edge labeling.

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

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

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 represents 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.

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:

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

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.*

5.2 Reduced Ordered BDDs

From BDD to Reduced BDDs

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

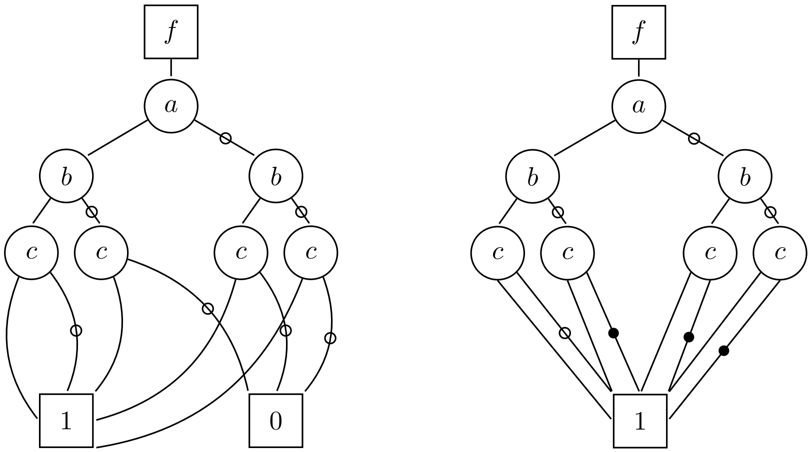


Figure 5.2: Left: Simple BDD. Right: After optimisation 1: BDD *with* complemented edges.

Optimisation 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.

Example. The introduction of the complement attribute is shown in Figure 5.3.

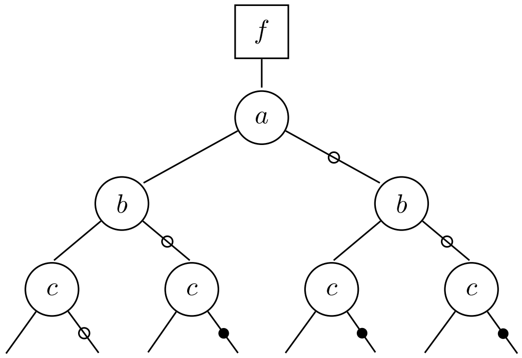


Figure 5.3: Optimisation 2: BDD with dangling edges.

Optimisation 2 - Dangling edges. Sine we only have one terminal node anymore, there is no need to actually connect the edges leading to 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.

Example. Figure 5.3 shows the BDD from above with dangling edges.

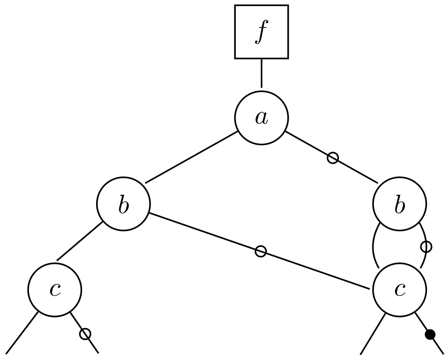


Figure 5.4: Optimisation 3: BDD with no duplicate sub-BDDs.

Optimisation 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.

Example. 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 Optimisation 3 results in the BDD shown in Figure 5.4 with two c-nodes removed.

Optimisation 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.

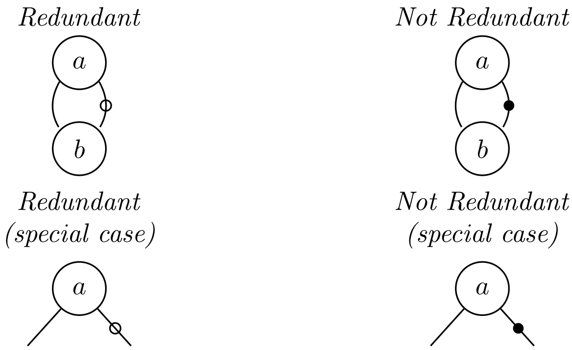


Figure 5.5: Examples of redundant and not redundant nodes.

Example. Let us consider the BDD given in Figure 5.4. The node b on the right sub-tree 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.

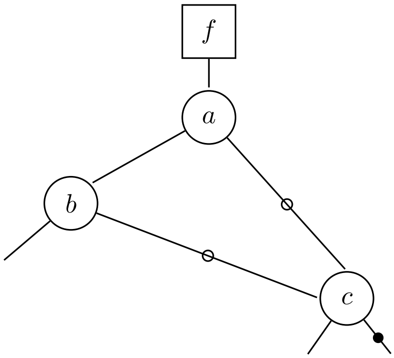


Figure 5.6: Optimisation 4: BDD with no redundant nodes.

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

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 $[x_1,\dots,x_n]$ 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 know 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.

Example. 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 and equivalence checking can be implemented using ROBDDs in *constant time.*

Constructing Formulas from ROBBDs

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 5.7. Give the formula that is represented by the ROBDD.

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,

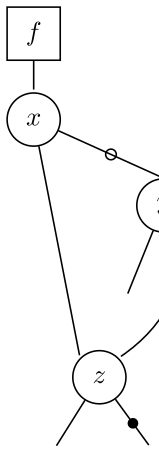


Figure 5.7: Reduced Ordered BDD.

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:

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

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.

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. We continue shifting the negation upwards until we reach the root node and no negated then-edges remain.

Example 1: Construction of a ROBDD

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.

Step 1. Compute all Cofactors.

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

$$f_{a\neg b\neg c} = \bot$$
$$f_{\neg a} = \neg c \wedge d \vee c = f_{ab}$$

Note that $f_{nega} = f_{ab}$.

Step 2. Draw BDD from Cofactors. The BDD is shown in Figure 5.8.

Step 3. Shift negations Upwards. Since there are no negated then-edges, there is nothing more to be done and Figure 5.8 represents the final ROBDD.

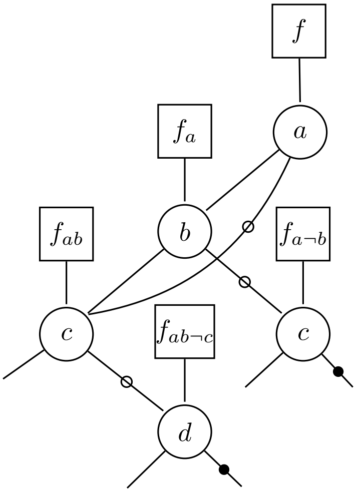


Figure 5.8: ROBDD for Example 1.

Example 2: Construction of a ROBDD

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.

Step 1. Compute all Cofactors.

$$f = (a \wedge \neg c) \vee \big(\neg a \wedge (b \vee (\neg b \wedge c))\big)$$
$$f_a = \neg c$$
$$f_{ab} = \neg c = f_a$$
$$f_{abc} = \bot$$
$$f_{ab\neg c} = \top$$
$$f_{a\neg b} = \neg c = f_a$$
$$f_{\neg a} = b \vee (\neg b \wedge c)$$
$$f_{\neg ab} = \top$$
$$f_{\neg a\neg b} = c = \neg f_{ab}$$

In this formula, we have a few interesting redundancies. The value of b does not have any effect on f_a, therefore we can ignore this two cofactors. Furthermore, $f_{\neg a \neg b}$ yielded c, which is the opposite of f_{ab}.

Step 2. Draw BDD from Cofactors. The BDD is given in Figure 5.9.

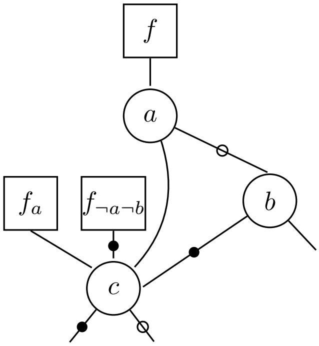


Figure 5.9: BDD with illegal compliments for Example 2.

Step 3. Shift negations Upwards.

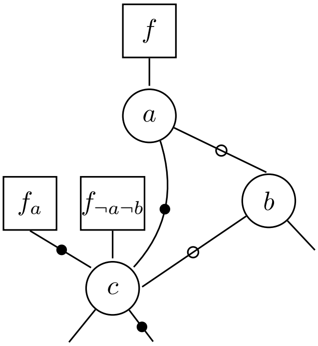


Figure 5.10: BDD of Example 2 with updated complements.

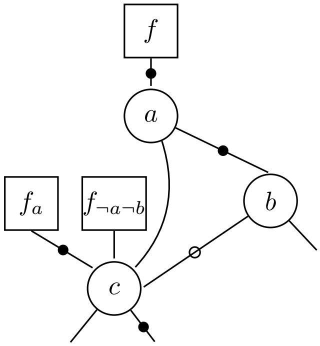


Figure 5.11: Final ROBDD of Example 2.

The ROBDD in Figure 5.9 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 5.10. To remove this negation, we repeat the same procedure with the edges of a. The result and final ROBDD can be seen in Figure 5.11.

5.4 Exercises

5.4.1 Examples

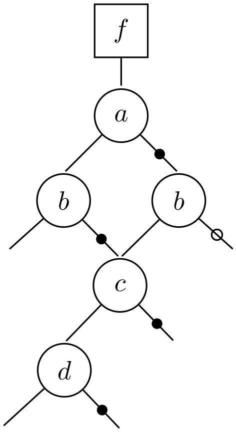
Example 1. (a) Use the following BDD to check if the function it represents evaluates to *true* or *false* with the following variable assignments.

i. $a = \top, b = \bot, c = \top, d = \top$

ii. $a = \bot, b = \top, c = \bot, d = \bot$

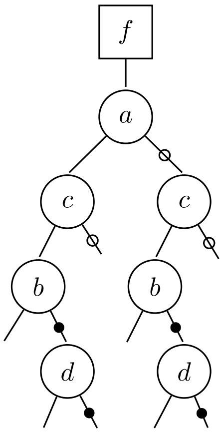
iii. $a = \top, b = \top, c = \bot, d = \bot$

(b) Find a propositional formula for the function f that is represented by the following BDD.

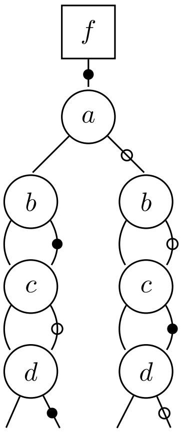


Example 2. Convert the following BDD into a RoBDD with variable order $a < c < b < d$. After converting, find a propositional formula for the function f that is represented by the following BDD.

(a)



(b)



Example 3. Construct a RoBDD with the given formula. Use complemented edges and a node for $\top$ as the only constant node. To simplify the drawing, you may assume that dangling edges point to the constant node. Write down all cofactors and mark them in the graph, that you draw.

(a) $f = (a \to b) \land (a \lor \neg b \lor c) \land (\neg a \lor \neg c)$
Use the following variable order: $a < b < c$.

(b) $f = \neg c \land (a \oplus b)$
Use the following variable order: $c < a < b$.

(c) $f = a \lor (a \leftrightarrow c) \lor (b \land c) \lor (b \land (d \oplus \neg c)) \lor (d \land (b \to a))$
Pick a sensible variable order and explain, why you chose it.

5.4.2 Solutions

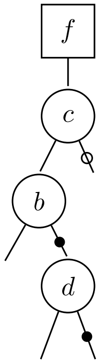
Solution 1. (a) i. *false*

ii. *true*

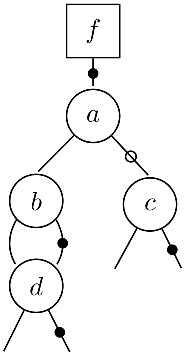
iii. *true*

(b)
- **with Option 1:**
  $f = \big(a \wedge (b \vee (\neg b \wedge \neg (c \wedge d)))\big) \vee \big(\neg a \wedge \neg ((b \wedge c \wedge d) \vee \neg b)\big)$
- **with Option 2:**
  $f = (a \wedge b) \vee (a \wedge \neg b \wedge c \wedge \neg d) \vee$
  $(a \wedge \neg b \wedge \neg c) \vee (\neg a \wedge b \wedge c \wedge \neg d) \vee (\neg a \wedge b \wedge \neg c)$

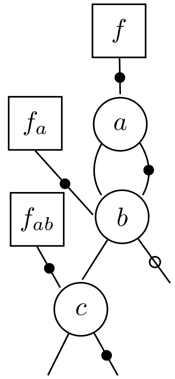
Solution 2. (a)
- **with Option 1:**
  $f = c \wedge (b \vee (\neg b \wedge \neg d)) \vee \neg c$
- **with Option 2:**
  $f = (c \wedge b) \vee (c \wedge \neg b \wedge \neg d) \vee \neg c$



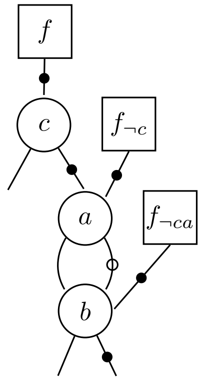
(b)
- **with Option 1:**
  $f = \neg\big(a \wedge ((b \wedge d) \vee (\neg b \wedge \neg d)) \vee (\neg a \wedge c)\big)$
- **with Option 2:**
  $f = \neg\big((a \wedge b \wedge d) \vee (a \wedge \neg c \wedge \neg d) \vee (\neg a \wedge c)\big)$



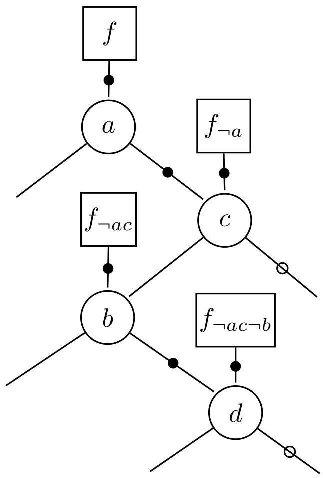
Solution 3. (a) $f = (a \to b) \wedge (a \vee \neg b \vee c) \wedge (\neg a \vee \neg c)$



(b) $f = \neg c \wedge (a \oplus b)$



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



5.4.3 Self-Evaluation

These examples will be solved online during class.

Example 1. Construct a (reduced and ordered) BDD for the formula $\phi = (a \rightarrow b) \vee (\neg a \rightarrow b)$,using alphabetic variable order. Use complemented edges and a node for "true" as the only constant node. To simplify drawing, you may assume that "dangling" edges point to the constant node. Write down all cofactors that you compute to obtain the final result.

Example 2. Construct a (reduced and ordered) BDD for the formula $f = (\neg x \vee \neg y) \wedge (x \wedge (y \vee c))$, using alphabetic variable order. Use complemented edges and a node for "true" as the only constant node. To simplify drawing, you may assume that "dangling" edges point to the constant node. Write down all cofactors that you compute to obtain the final result.

Example 3. Construct a (reduced and ordered) BDD for the formula $f = (\neg x \wedge \neg y) \vee (x \wedge y)$, using alphabetic variable order. Use complemented edges and a node for "true" as the only constant node. To simplify drawing, you may assume that "dangling" edges point to the constant node. Write down all cofactors that you compute to obtain the final result.

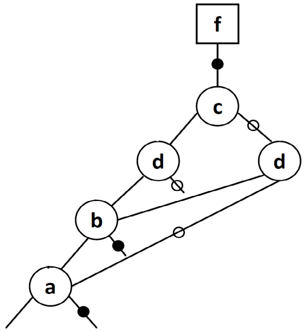


Figure 5.12: BDD

Example 4. Give the correct propositional formula f that is represented by the BDD in Figure 5.12.

Example 5. Assume that you have already constructed a BDD for a given formula and variable order. What can happen, if you change the variable order and you draw the BDD for the same formula with the new order again?

Example 6. Using BDDs, how can you perform a negation of a formula in constant time?

Example 7. For each of the following statements, state whether it constitutes an advantage of BDDs as a data structure, a disadvantage, or neither. Mark **a**dvantages with “A”, **d**isadvantages with “D”, and items which are **n**either with “N”. (Note: Consequently, items which are factually wrong, or have nothing to do with BDDs, should be marked with “N”!)

- ☐ Checks for entailment can be done in constant time.
- ☐ Using complemented edges, negation can be performed in constant time.
- ☐ Equivalence checks can be performed in constant time (assuming that the BDDs for the formula to check are already available).
- ☐ The size of a BDD may depend significantly on the variable order, which is hard to optimize.
- ☐ Logic operations, such as conjunction or disjunction, can be performed in polynomial time.

Declaration of Sources

Chapter 1 was based on the following book.

Michael Huth, Mark Dermot Ryan: *Logic in Computer Science: Modelling and Reasoning about Systems.* June 2004. Cambridge University Press. ISBN:978-0-521-54310-1