Chapter 5
Propositional Logic: Binary Decision Diagrams

The problem of deciding the satisfiability of a formula in propositional logic has turned out to have many important applications in computer science. This chapter and the next one present two widely used approaches for computing with formulas in propositional logic.

A binary decision diagram (BDD) is a data structure for representing the semantics of a formula in propositional logic. A formula is represented by a directed graph and an algorithm is used to *reduce* the graph. Reduced graphs have the property that the graphs for logically equivalent formulas are identical. Clearly, this gives a decision procedure for logical equivalence: transform A_1 and A_2 into BDDs and check that they are identical. A formula is valid iff its BDD is identical to the trivial BDD for *true* and a formula is satisfiable iff its BDD is not identical to the trivial BDD for *false*.

Before defining BDDs formally, the next section motivates the concept by reducing truth tables for formulas.

5.1 Motivation Through Truth Tables

Suppose that we want to decide if two formulas A_1 and A_2 in propositional logic are logically equivalent. Let us construct systematic truth tables, where *systematic* means that the assignments to the atomic propositions are arranged in some consistent order, for example, in lexicographic order by placing T before F and varying the values assigned to the atoms from the right to the left. Now, all we have to do is to check if the truth tables for A_1 and A_2 are identical. Of course, this is very inefficient, because 2^n rows are needed for each formula with n variables. Can we do better?

DOI 10.1007/978-1-4471-4129-7_5, © Springer-Verlag London 2012

Consider the following truth table for $p \vee (q \wedge r)$, where we have numbered the rows for convenience in referring to them:

	p	q	r	$p \vee (q \wedge r)$
1	T	T	T	T
2	T	T	F	T
3	T	F	T	T
4	T	F	F	T
5	F	T	T	T
6	F	T	F	F
7	F	F	T	F
8	F	F	F	F

From rows 1 and 2, we see that when p and q are assigned T, the formula evaluates to T *regardless* of the value of r, and similarly for rows 3 and 4. The first four rows can therefore be condensed into two rows:

	p	q	r	$p \vee (q \wedge r)$
1	T	T	$*$	T
2	T	F	$*$	T

where $*$ indicates that the value assigned to r is immaterial. We now see that the value assigned to q is immaterial, so these two rows collapse into one:

	p	q	r	$p \vee (q \wedge r)$
1	T	$*$	$*$	T

After similarly collapsing rows 7 and 8, the truth table has four rows:

	p	q	r	$p \vee (q \wedge r)$
1	T	$*$	$*$	T
2	F	T	T	T
3	F	T	F	F
4	F	F	$*$	F

Let us try another example, this time for the formula $p \oplus q \oplus r$. It is easy to compute the truth table for a formula whose only operator is $\oplus$, since a row evaluates to T if and only if an odd number of atoms are assigned T:

	p	q	r	$p \oplus q \oplus r$
1	T	T	T	T
2	T	T	F	F
3	T	F	T	F
4	T	F	F	T
5	F	T	T	F
6	F	T	F	T
7	F	F	T	T
8	F	F	F	F

Here, adjacent rows cannot be collapsed, but careful examination reveals that rows 5 and 6 show the same dependence on r as do rows 3 and 4. Rows 7 and 8 are similarly related to rows 1 and 2. Instead of explicitly writing the truth table entries for these rows, we can simply refer to the previous entries:

	p	q	r	$p \oplus q \oplus r$
1	T	T	T	T
2	T	T	F	F
3	T	F	T	F
4	T	F	F	T
5, 6	F	T	$*$	(See rows 3 and 4.)
7, 8	F	F	$*$	(See rows 1 and 2.)

The size of the table has been reduced by removing repetitions of computations of truth values.

5.2 Definition of Binary Decision Diagrams

A binary decision diagram, like a truth table, is a representation of the value of a formula under all possible interpretations. Each node of the tree is labeled with an atom, and solid and dotted edges leaving the node represent the assignment of T and F, respectively, to this atom. Along each branch, there is an edge for every atom in the formula, so there is a one-to-one correspondence between branches and interpretations. The leaf of a branch is labeled with the value of the formula under its interpretation.

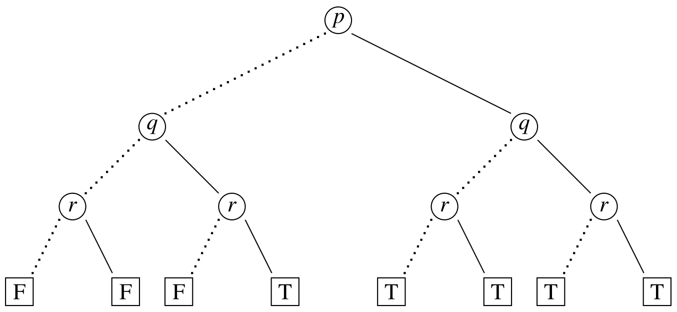


Fig. 5.1 A binary decision diagram for $p \vee (q \wedge r)$

Definition 5.1 A *binary decision diagram (BDD)* for a formula A in propositional logic is a directed acyclic graph. Each leaf is labeled with a truth value T or F. Each interior node is labeled with an atom and has two outgoing edges: one, the *false edge*, is denoted by a dotted line, while the other, the *true edge*, is denoted by a solid line. No atom appears more than once in a branch from the root to an edge.

A full or partial interpretation $\mathscr{I}_b$ for A is associated with each branch b from the root to a leaf. $\mathscr{I}_b(p) = T$ if the true edge was taken at the node labeled p and $\mathscr{I}_b(p) = F$ if the false edge was taken at the node labeled p.

Given a branch b and its associated interpretation $\mathscr{I}_b$, the leaf is labeled with $v_{\mathscr{I}_b}(A)$, the truth value of the formula under $\mathscr{I}_b$. If the interpretation is partial, it must assign to enough atoms so that the truth value is defined. ■

Example 5.2 Figure 5.1 is a BDD for $A = p \vee (q \wedge r)$. The interpretation associated with the branch that goes left, right, left is

$$\mathscr{I}(p) = F, \qquad \mathscr{I}(q) = T, \qquad \mathscr{I}(r) = F.$$

The leaf is labeled F so we can conclude that for this interpretation, $v_{\mathscr{I}}(A) = F$. Check that the value of the formula for the interpretation associated with each branch is the same as that given in the first truth table on page 96. ■

The BDD in the figure is a special case, where the directed acyclic graph is a tree and a *full* interpretation is associated with each branch.

5.3 Reduced Binary Decision Diagrams

We can modify the structure of a tree such as the one in Fig. 5.1 to obtain a more concise representation without losing the ability to evaluate the formula under all interpretations. The modifications are called *reductions* and they transform the tree into a *directed acyclic graph*, where the direction of an edge is implicitly from a node to its child. When no more reductions can be done, the BDD is *reduced*.

Algorithm 5.3 (Reduce)
Input: A binary decision diagram *bdd*.
Output: A reduced binary decision diagram *bdd'*.

- If *bdd* has more than two distinct leaves (one labeled T and one labeled F), remove duplicate leaves. Direct all edges that pointed to leaves to the remaining two leaves.
- Perform the following steps as long as possible:

  1. If both outgoing edges of a node labeled p_i point to the same node labeled p_j, delete this node for p_i and direct p_i's incoming edges to p_j.
  2. If two nodes labeled p_i are the roots of identical sub-BDDs, delete one sub-BDD and direct its incoming edges to the other node. ■

Definition 5.4 A BDD that results from applying the algorithm **Reduce** is a *reduced binary decision diagram.* ■

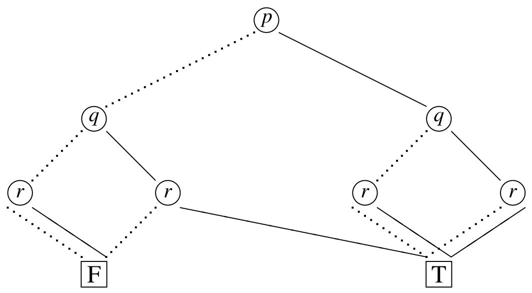
See Bryant (1986) or Baier and Katoen (2008, Sect. 6.7.3) for a proof of the following theorem:

Theorem 5.5 *The reduced BDD bdd' returned by the algorithm* **Reduce** *is logically equivalent to the input BDD bdd.*

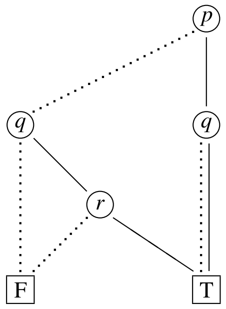
Let us apply the algorithm **Reduce** to the two formulas used as motivating examples in Sect. 5.1.

Example 5.6 Figure 5.1 shows a non-reduced BDD for $A = p \vee (q \wedge r)$.

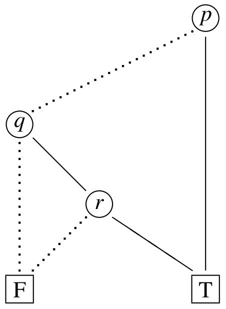
First, merge all leaves into just two, one for T and one for F:



Now we apply Step (1) of the algorithm repeatedly in order to remove nodes that are not needed to evaluate the formula. Once on the left-hand side of the diagram and twice on the right-hand side, the node for r has both outgoing edges leading to the same node. This means that the partial assignment to p and q is sufficient to determine the value of the formula. The three nodes labeled r and their outgoing edges can be deleted and the incoming edges to the r nodes are directed to the joint target nodes:



Step (1) can now be applied again to delete the right-hand node for q:



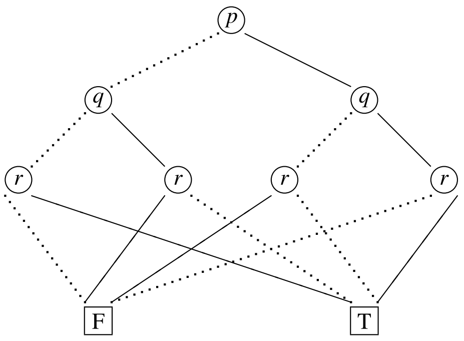
Since neither Step (1) nor Step (2) can be applied, the BDD is reduced.

There are four branches in the reduced BDD for $p \vee (q \wedge r)$. The interpretations associated with the branches are (from left to right):

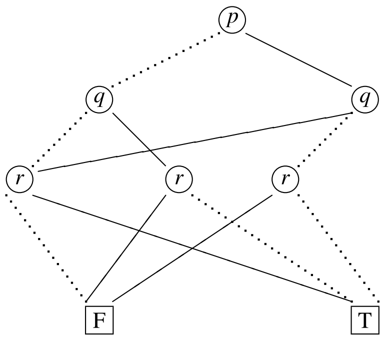
$$\begin{array}{lll}
\mathscr{I}_{b_1}(p) = F, & \mathscr{I}_{b_1}(q) = F, & \\
\mathscr{I}_{b_2}(p) = F, & \mathscr{I}_{b_2}(q) = T, & \mathscr{I}_{b_2}(r) = F, \\
\mathscr{I}_{b_3}(p) = F, & \mathscr{I}_{b_3}(q) = T, & \mathscr{I}_{b_3}(r) = T, \\
\mathscr{I}_{b_4}(p) = T. & &
\end{array}$$

The interpretations $\mathscr{I}_{b_1}$ and $\mathscr{I}_{b_4}$ are partial interpretations, but they assign truth values to enough atoms for the truth values of the formula to be computed. ■

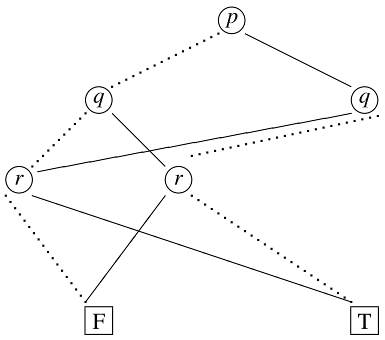
Example 5.7 Consider now the formula $A' = p \oplus q \oplus r$. We start with a tree that defines full interpretations for the formula and delete duplicate leaves. Here is the BDD that results:



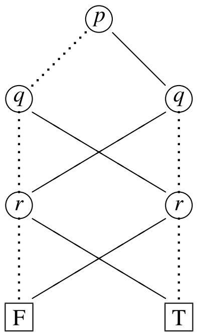
The reduction of Step (1) is not applicable, but examination of the BDD reveals that the subgraphs rooted at the left and right outermost nodes for r have the same structure: their F and T edges point to the same subgraphs, in this case the leaves $\boxed{F}$ and $\boxed{T}$, respectively. Applying Step (2), the T edge from the rightmost node for q can be directed to the leftmost node for r:



Similarly, the two innermost nodes for r are the roots of identical subgraphs and the F from the rightmost node for q can be directed to the second r node from the left:



Neither Step (1) nor Step (2) can be applied so the BDD is reduced. By rearranging the nodes, the following symmetric representation of the BDD is obtained:



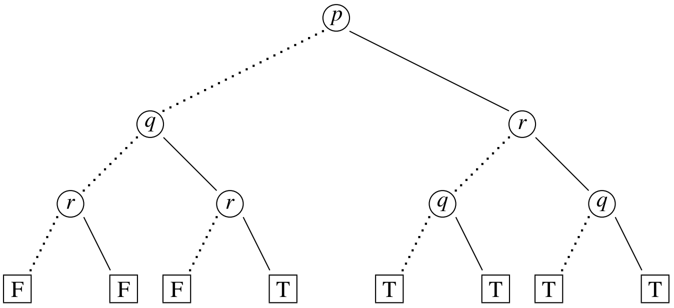
Check that the truth values of A' under the interpretations associated with each branch correspond to those in the reduced truth table on page 97. ■

5.4 Ordered Binary Decision Diagrams

The definition of BDDs did not place any requirements on the order in which atoms appear on a branch from the root to the leaves. Since branches can represent partial interpretations, the set of atoms appearing on one branch can be different from the set on another branch. Algorithms on BDDs require that the different orderings do not contradict each other.

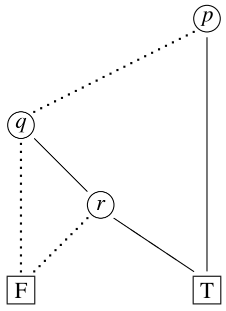
Definition 5.8 Let $\mathcal{O} = \{\mathcal{O}_A^1, \ldots, \mathcal{O}_A^n\}$, where for each i, $\mathcal{O}_A^i$ is a sequence of the elements of $\mathcal{P}_A$ (the set of atoms in A) defined by $<^i_{\mathcal{P}_A}$, a total relation that orders $\mathcal{P}_A$. $\mathcal{O}$ is a *compatible set of orderings for* $\mathcal{P}_A$ iff for all $i \neq j$, there are no atoms p, p' such that $p <^i_{\mathcal{P}_A} p'$ in $\mathcal{O}_A^i$ while $p' <^j_{\mathcal{P}_A} p$ in $\mathcal{O}_A^j$. ■

Example 5.9 Here is a BDD that is the same as the one in Fig. 5.1, except that the orderings are not compatible because q appears before r on the left branches, while r appears before q on the right branches:



■

Example 5.10 Consider again the reduced BDD for $p \vee (q \wedge r)$:



The four branches define three distinct orderings of the atoms:

$$\{(p,q),\ (p,q,r),\ (p)\},$$

but the orderings are compatible. ■

Definition 5.11 An *ordered binary decision diagram (OBDD)* is a BDD such that the set of orderings of atoms defined by the branches is compatible. ■

The proofs of the following theorems can be found in Bryant (1986).

Theorem 5.12 *The algorithm* ***Reduce*** *constructs an OBDD if the original BDD is ordered. For a given ordering of atoms, the reduced OBDDs for logically equivalent formulas are structurally identical.*

The theorem means that a reduced, ordered BDD is a *canonical* representation of a formula. It immediately provides a set of algorithms for deciding properties of formulas. Let A and B be formulas in propositional logic; construct reduced OBDDs for both formulas using a compatible ordering of $\{\mathscr{P}_A, \mathscr{P}_B\}$. Then:

- A is satisfiable iff $\boxed{T}$ appears in its reduced OBDD.
- A is falsifiable iff $\boxed{F}$ appears in its reduced OBDD.
- A is valid iff its reduced OBDD is the single node $\boxed{T}$.
- A is unsatisfiable iff its reduced OBDD is the single node $\boxed{F}$.
- If the reduced OBDDs for A and B are identical, then $A \equiv B$.

The usefulness of OBDDs depends of course on the efficiency of the algorithm **Reduce** (and others that we will describe), which in turn depends on the size of reduced OBDDs. In many cases the size is quite small, but, unfortunately, the size of the reduced OBDD for a formula depends on the ordering and the difference in sizes among different orderings can be substantial.

Theorem 5.13 *The OBDD for the formula*:

$$(p_1 \wedge p_2) \vee \cdots \vee (p_{2n-1} \wedge p_{2n})$$

has $2n+2$ *nodes under the ordering* $p_1, \ldots, p_{2n}$, *and* 2^{n+1} *nodes under the ordering* $p_1, p_{n+1}, p_2, p_{n+2}, \ldots, p_n, p_{2n}$.

Fortunately, you can generally use heuristics to choose an efficient ordering, but there are formulas that have large reduced OBDDs under any ordering.

Theorem 5.14 *There is a formula A with n atoms such that the reduced OBDD for any ordering of the atoms has at least* 2^{cn} *nodes for some* $c > 0$.

5.5 Applying Operators to BDDs

It hardly seems worthwhile to create a BDD if we start from the full binary tree whose size is about the same as the size of the truth table. The power of BDDs comes from the ability to perform operations directly on two reduced BDDs. The algorithm **Apply** recursively constructs the BDD for $A_1 \, op \, A_2$ from the reduced BDDs for A_1 and A_2. It can also be used to construct an initial BDD for an arbitrary formula by building it up from the BDDs for atoms.

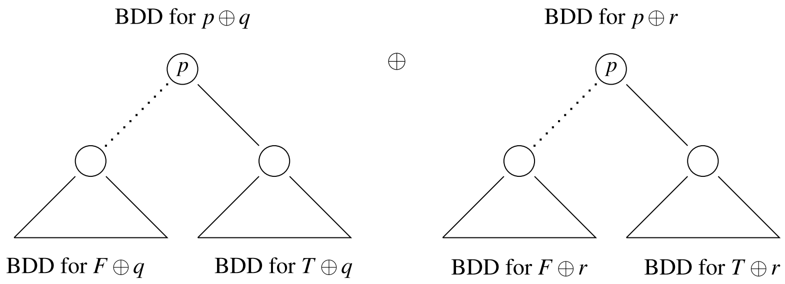
The algorithm ***Apply*** *works only on ordered BDDs.*

Algorithm 5.15 (Apply)
Input: OBDDs bdd_1 for formula A_1 and bdd_2 for formula A_2, using a compatible ordering of $\{\mathscr{P}_{A_1}, \mathscr{P}_{A_2}\}$; an operator *op*.
Output: An OBDD for the formula $A_1 \; op \; A_2$.

- If bdd_1 and bdd_2 are both leaves labeled w_1 and w_2, respectively, return the leaf labeled by $w_1 \; op \; w_2$.
- If the roots of bdd_1 and bdd_2 are labeled by the same atom p, return the following BDD: (a) the root is labeled by p; (b) the left sub-BDD is obtained by recursively performing this algorithm on the left sub-BDDs of bdd_1 and bdd_2; (c) the right sub-BDD is obtained by recursively performing this algorithm on the right sub-BDDs of bdd_1 and bdd_2.
- If the root of bdd_1 is labeled p_1 and the root of bdd_2 is labeled p_2 such that $p_1 < p_2$ in the ordering, return the following BDD: (a) the root is labeled by p_1; (b) the left sub-BDD is obtained by recursively performing this algorithm on the left sub-BDD of bdd_1 and on (the entire BDD) bdd_2; (c) the right sub-BDD is obtained by recursively performing this algorithm on the right sub-BDD of bdd_1 and on (the entire BDD) bdd_2.
  This construction is also performed if bdd_2 is a leaf, but bdd_1 is not.
- Otherwise, we have a symmetrical case to the previous one. The BDD returned has its root labeled by p_2 and its left (respectively, right) sub-BDD obtained by recursively performing this algorithm on bdd_1 and on the left (respectively, right) sub-BDD of bdd_2. ■

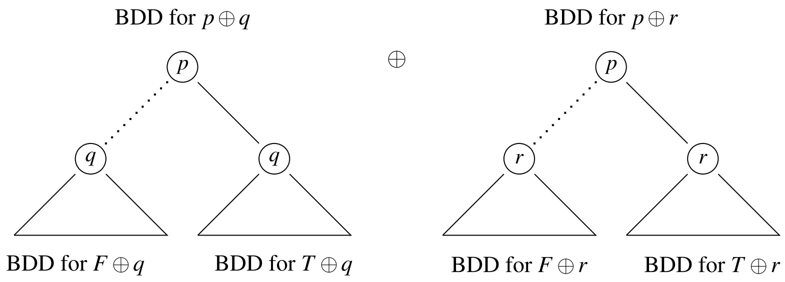
We now work out a complete example of the application of the **Apply** algorithm. It is quite lengthy, but each step in the recursive algorithm should not be difficult to follow.

Example 5.16 We construct the BDD for the formula $(p \oplus q) \oplus (p \oplus r)$ from the BDDs for $p \oplus q$ and $p \oplus r$. In the following diagram, we have drawn the two BDDs with the operator $\oplus$ between them:

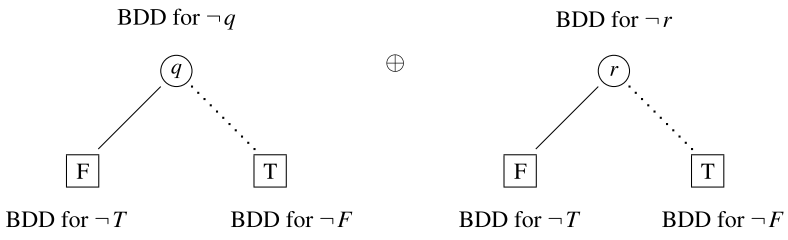


The sub-BDDs will be BDDs for the four subformulas obtained by substituting T and F for p. Notations such as $F \oplus r$ will be used to denote the formula obtained by partially evaluating a formula, in this case, partially evaluating $p \oplus r$ under an interpretation such that $\mathscr{I}(p) = F$.

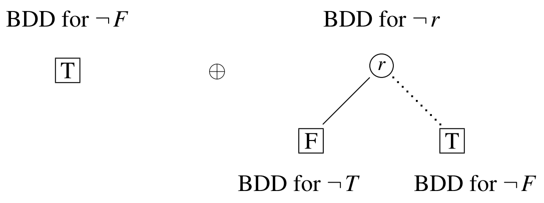
Since there is only one atom in each sub-BDD, we know what the labels of their roots are:



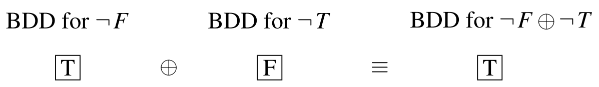
Let us now take the right-hand branch in both BDDs that represent assigning T to p. Evaluating the partial assignment gives $T \oplus q \equiv \neg q$ and $T \oplus r \equiv \neg r$. To obtain the right-hand sub-BDD of the result, we have to compute $\neg q \oplus \neg r$:



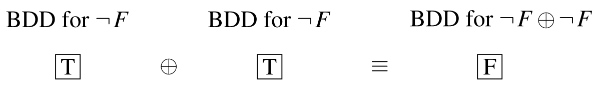
The recursion can be continued by taking the right-hand branch of the BDD for $\neg\, q$ and assigning F to q. Since the BDD for $\neg\, r$ does not depend on the assignment to q, it does not split into two recursive subcases. Instead, the algorithm must be applied for each sub-BDD of $\neg\, q$ together with the *entire* BDD for $\neg\, r$. The following diagram shows the computation that is done when the right-hand branch of the BDD for $\neg\, q$ is taken:



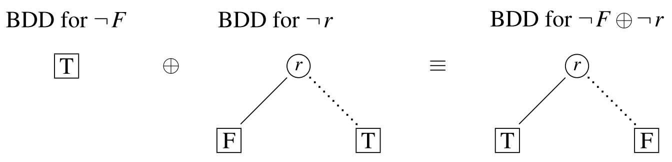
Recursing now on the BDD for $\neg\, r$ also gives base cases, one for the left-hand (true) branch:



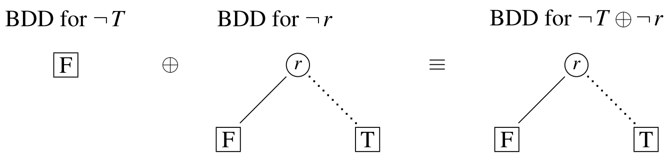
and one for the right-hand (false) branch:



When returning from the recursion, these two results are combined:



Similarly, taking the left-hand branch of the BDD for $\neg\, q$ gives:



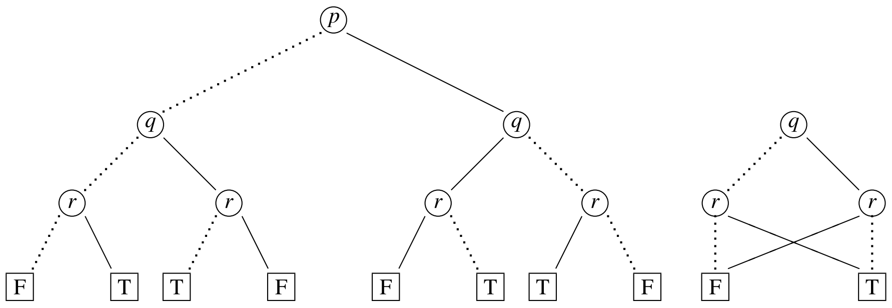
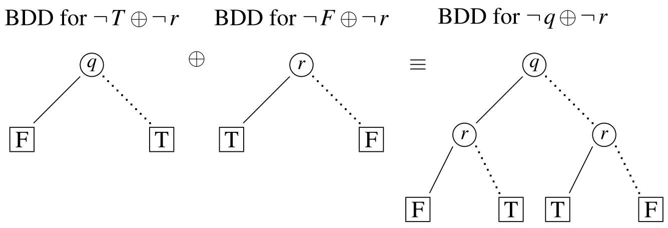


Fig. 5.2 BDD after the **Apply** and **Reduce** algorithms terminate

Returning from the recursion to the BDD for $\neg q$ gives:


BDD for $\neg T \oplus \neg r$
BDD for $\neg F \oplus \neg r$
BDD for $\neg q \oplus \neg r$
q
F
T
$\oplus$
r
T
F
$\equiv$
q
r
r
F
T
T
F

The BDD obtained upon termination of the algorithm is shown in Fig. 5.2 and to its right is the BDD that results from reducing the BDD. Check that this is the reduced BDD for $q \oplus r$:

$$(p \oplus q) \oplus (p \oplus r) \;\equiv\; (p \oplus p) \oplus (q \oplus r) \;\equiv\; \mathit{false} \oplus (q \oplus r) \;\equiv\; q \oplus r.$$

■

5.6 Restriction and Quantification *

This section presents additional important algorithms on BDDs.

5.6.1 Restriction

Definition 5.17 The *restriction* operation takes a formula A, an atom p and a truth value $w = T$ or $w = F$. It returns the formula obtained by substituting w for p and partially evaluating A. Notation: $A|_{p=w}$. ■

Example 5.18 Let $A = p \vee (q \wedge r)$; its restrictions are:

$$\begin{aligned} A|_{r=T} &\equiv p \vee (q \wedge T) \equiv p \vee q, \\ A|_{r=F} &\equiv p \vee (q \wedge F) \equiv p \vee F \equiv p. \end{aligned}$$

■

The correctness of the algorithm **Reduce** is based upon the following theorem which expresses the application of an operator in terms of its application to restrictions. We leave its proof as an exercise.

Theorem 5.19 (*Shannon expansion*)

$$A_1 \; op \; A_2 \equiv (p \; \wedge \; (A_1|_{p=T} \; op \; A_2|_{p=T})) \;\; \vee \;\; (\neg \, p \; \wedge \; (A_1|_{p=F} \; op \; A_2|_{p=F})).$$

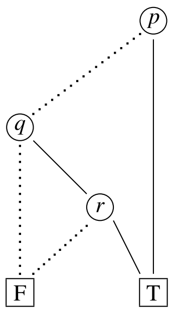
Restriction is very easy to implement on OBDDs.

Algorithm 5.20 (Restrict)
Input: An OBDD *bdd* for a formula A; a truth value w.
Output: An OBDD for $A|_{p=w}$.
Perform a recursive traversal of the OBDD:

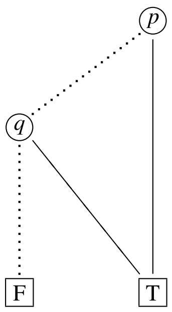
- If the root of *bdd* is a leaf, return the leaf.
- If the root of *bdd* is labeled p, return the sub-BDD reached by its true edge if $w = T$ and the sub-BDD reached by its false edge if $w = F$.
- Otherwise (the root is labeled p' for some atom which is not p), apply the algorithm to the left and right sub-BDDs, and return the BDD whose root is p' and whose left and right sub-BDDs are those returned by the recursive calls. ■

The BDD that results from **Restrict** may not be reduced, so the **Reduce** algorithm is normally applied immediately afterwards.

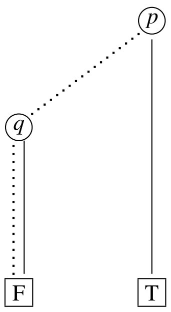
Example 5.21 The OBDD of $A = p \vee (q \wedge r)$ is shown in (a) below. (b) is $A|_{r=T}$, (c) is $A|_{r=F}$ and (d) is (c) after reduction.



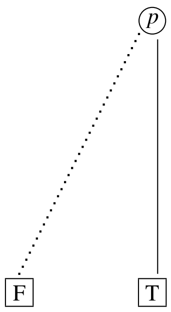
(a)



(b)



(c)



(b)

Compare the OBDDs in (b) and (d) with the formulas in Example 5.18. ■

5.6.2 *Quantification*

Definition 5.22 Let A be a formula and p an atom. The *existential quantification of* A is the formula denoted $\exists pA$ and the *universal quantification of* A is the formula denoted $\forall pA$. $\exists pA$ is true iff A is true for *some* assignment to p, while $\forall pA$ is true iff for *all* assignments to p, A is true. ■

These formulas are in an extension of propositional logic called *quantified propositional logic*. The proof of the following theorem is left as an exercise.

Theorem 5.23

$$\exists pA \equiv A|_{p=F} \vee A|_{p=T}, \qquad \forall pA \equiv A|_{p=F} \wedge A|_{p=T}.$$

Quantification is easily computed using OBDDs:

$$\begin{array}{lll} \exists pA & \text{is} & \textbf{Apply}(\textbf{Restrict}(A, p, F), \mathit{or}, \textbf{Restrict}(A, p, T)), \\ \forall pA & \text{is} & \textbf{Apply}(\textbf{Restrict}(A, p, F), \mathit{and}, \textbf{Restrict}(A, p, T)). \end{array}$$

Example 5.24 For the formulas $A = p \vee (q \wedge r)$, we can use $A|_{r=F} \equiv p$ and $A|_{r=T} \equiv p \vee q$ from Example 5.18 to compute its quantifications on r:

$$\begin{aligned} \exists r\,(p \vee (q \wedge r)) &\equiv p \vee (p \vee q) \equiv p \vee q, \\ \forall r\,(p \vee (q \wedge r)) &\equiv p \wedge (p \vee q) \equiv p. \end{aligned}$$

We leave it as an exercise to perform these computations using OBDDs. ■

5.7 Summary

Binary decision diagrams are a data structure for representing formulas in propositional logic. A BDD is a directed graph that reduces redundancy when compared with a truth table or a semantic tree. Normally, one ensures that all branches of a BDD use compatible orderings of the atomic propositions. An OBDD can be reduced and reduced OBDDs of two formulas are structurally identical if and only if the formulas are logically equivalent. A recursive algorithm can be used to efficiently compute $A\ op\ B$ given the OBDDs for A and B. BDDs have been widely used in model checkers for the verification of computer hardware.

5.8 Further Reading

Bryant's original papers on BDDs (Bryant, 1986, 1992) are relatively easy to read. There is an extensive presentation of BDDs in Baier and Katoen (2008, Sect. 6.7).

5.9 Exercises

5.1 Construct reduced OBDDs for $p \uparrow (q \uparrow r)$ and $(p \uparrow q) \uparrow r$. What does this show?

5.2 Construct reduced OBDDs for the formula $(p_1 \wedge p_2) \vee (p_3 \wedge p_4)$ using two orderings of the variables: p_1, p_2, p_3, p_4 and p_1, p_3, p_2, p_4.

5.3 How can OBDDs be used to check if $A \models B$?

5.4 Compute the Shannon expansion of $(p \rightarrow (q \rightarrow r)) \rightarrow ((p \rightarrow q) \rightarrow (p \rightarrow r))$ with respect to each one of its atomic propositions. Why do you know the answer even before you start the computation?

5.5 Prove the Shannon expansion (Theorem 5.19) and the formula for propositional quantification (Theorem 5.23).

5.6 Prove that $\exists r\,(p \vee (q \wedge r)) = p \vee q$ and $\forall r\,(p \vee (q \wedge r)) = p$ using BDDs (Example 5.24).

References

C. Baier and J.-P. Katoen. *Principles of Model Checking*. MIT Press, 2008.

R.E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35:677–691, 1986.

R.E. Bryant. Symbolic Boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24:293–318, 1992.