Chapter 11

Quantified Boolean Formulas

Contents

In this chapter we discuss an extension of propositional formulas, known as *quantified boolean formulas*. Quantified boolean formulas have the same expressive power as propositional formulas, but in general allow for a more succinct representation of problems. Quantified boolean formulas can be used to present finite-state combinatorial problems, such as planning or games. In addition, they are extensively used in symbolic model checking algorithms studied later in this book.

11.1 Quantified Boolean Formulas

In this section we introduce quantified boolean formulas as an extension of propositional formulas and discuss their syntax and semantics. We generalize the notions of validity and

satisfiability to quantified boolean formulas and show that, unlike propositional formulas, checking a truth value of a formula in an interpretation is as hard as satisfiability or validity checking. Then we discuss in length the notions of free and bound variables and difficulties created by bound variables in defining the notion of substitution of a formula for a boolean variable.

11.1.1 Syntax and Semantics

In this chapter we will refer to boolean variables simply as *variables*. Quantified boolean formulas extend propositional formulas by allowing quantification on variables.

DEFINITION 11.1 (Quantified Boolean Formula) The definition of *quantified boolean formulas* is obtained from Definition 3.1 on page 30 of propositional formulas by adding the following items:

(6) If p is a boolean variable and F is a formula, then $\forall pF$ and $\exists pF$ are formulas.

The symbol $\forall$ is called the *universal quantifier* and $\exists$ is called the *existential quantifier*. We also call *quantifiers* expressions $\forall p$ and $\exists p$. If a formula F contains $\forall p$ (respectively $\exists p$), we say that p is *universally quantified* (respectively, *existentially quantified*) in F. ❑

Intuitively, $\forall pF$ means that F is true *for all* possible values for p, i.e., both when p is true and p is false. Likewise, $\exists pF$ means that F is true *for some* possible value for p.

As in the case of propositional formulas, we will often omit parentheses in formulas. For the connectives used in quantified boolean formulas we will use the same priorities as in Figure 3.1 on page 30, but additionally assume that the quantifiers have the same priority as negation $\neg$. For example, the formula $\forall p \neg p \rightarrow q$ represents $(\forall p \neg p) \rightarrow q$.

The semantics of quantified Boolean formulas is defined by extending the semantics of propositional formulas to quantifiers. To define this extension, let us introduce a new notation. Let I be an interpretation, p an variable, and b a boolean value. Then by $I[p \leftarrow b]$ we denote the interpretation defined as follows:

$$I[p \leftarrow b](q) \stackrel{\text{def}}{=} \begin{cases} I(q), & \text{if } p \neq q; \\ b, & \text{if } p = q. \end{cases}$$

It is easy to see that $I[p \leftarrow b](p) = b$ and that $I[p \leftarrow b]$ agrees with I on all variables different from p.[1]

DEFINITION 11.2 (Truth) The notion of truth for quantified boolean formulas is obtained from Definition 3.2 on page 31 of truth of propositional formulas by adding the following items.

[1] The definition of $I[p \leftarrow b]$ is nearly identical to that of $I + (p \mapsto b)$, except that we do not require that I be undefined on p.

(7) $I(\forall pF) = 1$ if and only if $I[p \leftarrow 0](F) = 1$ and $I[p \leftarrow 1](F) = 1$.

(8) $I(\exists pF) = 1$ if and only if $I[p \leftarrow 0](F) = 1$ or $I[p \leftarrow 1](F) = 1$.

As in the case of propositional formulas, we will write $I \vDash F$, if F is true in I. ❑

In other words, the formula $\forall pF$ is true in I if *for all* boolean values b we have $I[p \leftarrow b] = 1$, and the formula $\exists pF$ is true in I if *there exists* a boolean value b such that $I[p \leftarrow b] = 1$. So $\forall pF$ is informally read as "for all p, F", and $\exists pF$ is read as "there exists p such that F".

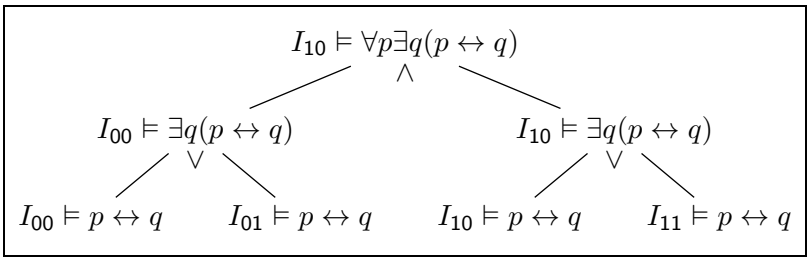
The notions of *satisfiability*, *validity* and *equivalence* are defined in the same way as for propositional formulas.

We can evaluate a quantified boolean formula in an interpretation using Definition 11.2 of truth, but the evaluation is not as straightforward as in the case of propositional formulas.

EXAMPLE 11.3 Let us consider an example: evaluation of the formula $\forall p\exists q(p \leftrightarrow q)$ in the interpretation $I = \{p \mapsto 1, q \mapsto 0\}$. Let us introduce a notation which is convenient for this example. For any two boolean values b_1, b_2 denote by $I_{b_1b_2}$ the interpretation $\{p \mapsto b_1, q \mapsto b_2\}$. In this notation I can be denoted as I_{10}. By the definition, we have

$$I_{10} \vDash \forall p\exists q(p \leftrightarrow q) \Leftrightarrow \boxed{\begin{array}{l} I_{00} \vDash \exists q(p \leftrightarrow q) \\ I_{10} \vDash \exists q(p \leftrightarrow q) \end{array} \text{ and }} \Leftrightarrow \boxed{\begin{array}{l} \boxed{\begin{array}{l} I_{00} \vDash p \leftrightarrow q \\ I_{01} \vDash p \leftrightarrow q \end{array} \text{ or}} \\ \text{and} \\ \boxed{\begin{array}{l} I_{10} \vDash p \leftrightarrow q \\ I_{11} \vDash p \leftrightarrow q \end{array} \text{ or}} \end{array}}$$

This evaluates to true, so the formula is true in I_{10}. This formula evaluation process can be depicted as an AND-OR tree as follows:



Note the essential difference between the evaluations of propositional and quantified boolean formulas. To evaluate a quantified boolean formula, we build and AND-OR tree, which is unnecessary for propositional formulas. In fact, evaluation of quantified boolean formulas is a PSPACE-complete problem, while propositional formulas can be evaluated in polynomial time.

Observe what happens when we evaluate $\forall p\exists q(p \leftrightarrow q)$ in a different interpretation. It is not hard to see that the result will be the same. Indeed, let us extend the notation $I_{b_1b_2}$

by using wildcards _ for b_1, b_2. If we use _ for b_i, it means that we do not know the value b_i in I. For example, $I_{\_\_}$ stands for an arbitrary interpretation (both the values b_1 and b_2 are unknown). We have

$$I_{\_\_} \models \forall p \exists q (p \leftrightarrow q) \Leftrightarrow \boxed{\begin{array}{l} I_{0\_} \models \exists q (p \leftrightarrow q) \\ I_{1\_} \models \exists q (p \leftrightarrow q) \end{array} \text{ and }} \Leftrightarrow \boxed{\begin{array}{l} \boxed{\begin{array}{l} I_{00} \models p \leftrightarrow q \\ I_{01} \models p \leftrightarrow q \end{array} \text{ or }} \\ \\ \text{and} \\ \\ \boxed{\begin{array}{l} I_{10} \models p \leftrightarrow q \\ I_{11} \models p \leftrightarrow q \end{array} \text{ or }} \end{array}}$$

Therefore, the formula $\forall p \exists q (p \leftrightarrow q)$ is true in all interpretations $I_{\_\_}$, hence it is valid. ❑

11.1.2 Free and Bound Variables

The truth value of a propositional formula F in an interpretation depends, in general, on the truth values of all variables p occurring in F. As one can see from Example 11.3, for quantified boolean formulas the situation is different: the truth value of F in I does not depend on the values in I of variables occurring in quantifiers, called the *bound variables* in F.

DEFINITION 11.4 (Subformula, Polarity, Free and Bound Occurrences) The notion of subformula for quantified boolean formulas is obtained from that of Definition 3.7 on page 35 for propositional formulas by adding the following item:

(6) The formula F_1 is the immediate subformula of the formulas $\forall p F_1$ and $\exists p F_1$.

The notions of *position* and *polarity* for quantified boolean formulas are obtained from that of Definition 4.10 on page 50 by adding the following item. Let $F|_\pi = G$ and G has the form $\forall p G_1$ or $\exists p G_1$. Then $\pi.1$ is a position in F, $F|_{\pi.1} \stackrel{\text{def}}{=} G_1$ and $pol(F, \pi.1) \stackrel{\text{def}}{=} pol(F, \pi)$. We will use the notions of *positive* and *negative occurrences* of subformulas in the same way as we did for propositional formulas, see Definition 4.10 on page 50.

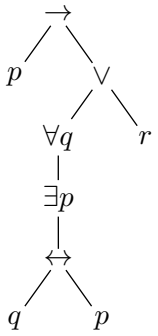
Let p be a boolean variable and $F|_\pi = p$. We say that the occurrence of p at the position π in F is *bound* if π can be represented as a concatenation of two strings $\pi_1 \pi_2$ such that $F|_{\pi_1}$ has the form $\forall p G$ or $\exists p G$ for some G. Otherwise the occurrence of p in F is said to be *free*. If a variable p has some free (respectively, bound) occurrences in F, we call p a *free* (respectively *bound*) variable of F. A quantified boolean formula F is called *closed* if it has no free variables. ❑

As an example, consider the formula $p \to \forall q \exists p (q \leftrightarrow p) \vee r$. The variable p has both free and bound occurrences in it: the first occurrence of p is free, while the last one is bound. The variable q has only bound occurrences in the formulas, and the variable r only free

occurrences. Therefore, p and r are the free variables of this formula, while p and q are the bound ones.

In the sequel we will denote by $\exists\forall$ a symbol which stands for either $\forall$ or $\exists$. When we use several occurrences of this symbol in the same context, all occurrences will stand for the same quantifier, i.e., either $\forall$ or $\exists$ but not both. We may also use subscripts $\exists\forall_i$ to distinguish between different quantifiers.

The notions of free and bound occurrences of variables are best illustrated when the formula is represented in a tree-like form. An occurrence of a variable p is bound, if above this occurrence in the tree there is a node $\exists\forall p$. Consider, for example, the parse tree for the formula $p \to \forall q \exists p (q \leftrightarrow p) \vee r$:



It can easily be seen that, for example, the occurrence of the variable q in a leaf of the tree is bound, since above it the tree contains $\forall q$.

11.1.3 Truth and Satisfiability

By straightforward induction on the definition of truth, it is not hard to argue that the value of a quantified boolean formulas only depends on the values of its free variables.

LEMMA 11.5 *Let I_1 and I_2 be two interpretations which agree on all free variables of F, i.e., for all free variables p of F we have $I_1(p) = I_2(p)$. Then $I_1 \vDash F$ if and only if $I_2 \vDash F$.*

PROOF. The proof is by induction on F. We will only consider the case when F has the form $\forall pG$, other cases are similar. Let V be the set of all free variables of F. Then the set of free variables of G is a subset of $V \cup \{p\}$. Therefore, for every boolean value b $I_1[p \leftarrow b]$ agrees with $I_2[p \leftarrow b]$ on all free variables of G, this fact will be used when we apply the induction hypothesis below. We have

$I_1 \vDash \forall pG \Leftrightarrow$
(by the definition of truth)
$I_1[p \leftarrow 0](G) = 1$ and $I_1[p \leftarrow 1](G) = 1 \Leftrightarrow$
(using the induction hypothesis)
$I_2[p \leftarrow 0](G) = 1$ and $I_2[p \leftarrow 1](G) = 1 \Leftrightarrow$
(by the definition of truth)
$I_2 \vDash \forall pG$. ❑

This lemma implies the following consequence for closed formulas.

LEMMA 11.6 *For closed quantified boolean formulas the notions of truth, validity and satisfiability coincide in the following sense: for every interpretation I and closed formula F the following propositions are equivalent: (i) $I \models F$; (ii) F is satisfiable; and (iii) F is valid.*

PROOF. Evidently, if F is valid, then $I \models F$, and if $I \models F$ then F is satisfiable. Therefore is enough to prove that satisfiability of F implies its validity. Suppose that F is satisfiable, then for some interpretation I_1 we have $I_1 \models F$. Take an arbitrary interpretation I_2. Since F has no free variables, then I_1 and I_2 agree on all free variables of F. By Lemma 11.5 we have $I_2 \models F$. Since I_2 was arbitrary, the formula F is valid. ❑

In general, satisfiability and validity of quantified boolean formulas can easily be seen a special case of satisfiability (or, equivalently, validity) of closed formulas.

LEMMA 11.7 *Let F be a quantified boolean formula with free variables $p_1, \ldots, p_n$. Then F is satisfiable (respectively, valid) if and only if the formula $\exists p_1 \ldots \exists p_n F$ (respectively, $\forall p_1 \ldots \forall p_n F$) is satisfiable.* ❑

11.1.4 Monotonic and Equivalent Replacement Theorems

Our next aim is to establish analogues of the equivalent and monotonic replacement theorems. Monotonic Replacement Theorem 4.13 can be reformulated literally for quantified boolean formulas.

THEOREM 11.8 (Monotonic Replacement) *Let F_1 be a subformula of a formula G_1 and $F_1 \to F_2$ be valid. Let the formula G_2 be obtained from G_1 by replacement of one or more positive (respectively, negative) occurrences of F_1 by F_2. Then the formula $G_1 \to G_2$ (respectively $G_2 \to G_1$) is valid.*

PROOF. **Later**. ❑

In other words, the truth value of a formula is monotonic on subformulas occurring positively and anti-monotonic on subformulas occurring negatively.

In a similar way we can prove an analogue of Equivalent Replacement Theorem 3.9.

THEOREM 11.9 (Equivalent Replacement) *Let F_1 be a subformula of a formula G_1 and $F_1 \equiv F_2$. Let the formula G_2 be obtained from G_1 by replacement of one or more occurrences of F_1 by F_2. Then $G_1 \equiv G_2$.* ❑

Note that Monotonic Replacement Lemma 4.12 on page 54 does not hold for quantified boolean formulas because of the bound variables. Indeed, consider the interpretation $I = \{p \mapsto 1, q \mapsto 1\}$. Then $I \models q \to p$. We also have $I \models \forall p(q)$ but not $I \models \forall p(p)$. This

example also shows that Lemma 3.8 on equivalent replacement does not hold for quantified boolean formulas, since $I \models p \leftrightarrow q$ but $I \not\models \forall p(p) \leftrightarrow \forall p(q)$.

11.1.5 Substitutions

In some algorithms of this chapter, we will have to replace formulas by other formulas. When we replace subformulas in quantified boolean formulas, a special care is required to deal with bound variables.

Free occurrences of a boolean variable p in a formula F correspond to a "parameter" of F. For example, the formula $\exists q(\neg p \leftrightarrow q)$ expresses that there is a boolean value which is equivalent to the negation of p. In this formula p can be considered a parameter. We can replace p by a formula, e.g., $p_1 \wedge p_2$, and obtain a formula which expresses that there is a truth value equivalent to the negation of $p_1 \wedge p_2$, namely $\exists q(\neg(p_1 \wedge p_2) \leftrightarrow q)$. Thus, replacement of p by a formula can be regarded as substitution of an expression for a parameter. We cannot, however, immediately formalize substitution of variables by formulas as simply replacement of variables, as two kinds of problems arise from such a naive formalization.

The first problem appears when we try to replace a bound variable: the result may be not a formula at all. For example, if we replace in $\exists q(\neg p \leftrightarrow q)$ the variable q by $\neg q$ we obtain the expression $\exists \neg q(\neg p \leftrightarrow \neg q)$ which is not a formula at all. This suggests that only free occurrences of variables can be replaced. However, this restriction is not enough.

The second problem arises when the substituted formula contains free variables which become bound after the substitution. To illustrate this problem, consider again the formula $\exists q(\neg p \leftrightarrow q)$. Evidently, if we substitute a formula F for p, we expect to obtain a formula which expresses that there exists a boolean value q equivalent to the negation of F. Let us try to replace p by a formula F with free occurrence of q, for example, by q itself. We obtain the formula $\exists q(\neg q \leftrightarrow q)$. This formula means that there exists a boolean value equivalent to its own negation, which is not what we intended. Even worse, the formula $\exists q(\neg p \leftrightarrow q)$ is valid while the formula $\exists q(\neg q \leftrightarrow q)$ is unsatisfiable. The problem in this case was created by the fact that a free variable of F becomes bound after after we substitute F for p. Let us introduce a notion that guarantees that a substitution preserves the meaning of a formula.

DEFINITION 11.10 (Formula Free for a Variable) Let F and G be quantified boolean formulas and p be a variable. We say that G is *free for p in F* if after the replacement in F of all free occurrences of p by G no free occurrence of a variable in G becomes bound. More strictly, this can be formulated as follows. G is *not* free for p in F if there exist paths π_1, π_2 and a formula F_1 such that $F|_{\pi_1.\pi_2} = p$, the occurrence of p at the position $\pi_1.\pi_2$ in F is free, $F|_{\pi_1} = \exists \forall q F_1$, and q is a free variable of G. ❑

For example, the formula q is *not* free for p in $\exists q(\neg p \leftrightarrow q)$. In general, a formula G is free for p in $\exists q(\neg p \leftrightarrow q)$ if and only if q is not a free variable of G.

If F, G are quantified boolean formulas and p is a variable, we write F_p^G to denote the formula obtained by replacing in F all *free* occurrences of p by G. When we use this notation, we always assume that G is free for p in F.

What if we need to substitute a formula G for a variable p in F so as to preserve the meaning of parameters, but G is not free for p in F? In this case we can change F into an equivalent formula F' so that G is free for p in F', and then substitute G for p in F'. The aim of the transformation changing F into F' is to replace bound variables in F' which are free in G by other bound variables. This operation is called *renaming bound variables*. We will give quite a restrictive definition of renaming bound variables, since the definition in its most general form is rather complex. However, this restrictive definition is enough for our purpose.

DEFINITION 11.11 (Renaming Bound Variables) We say that a formula F' is obtained from a formula F by *renaming bound variables* if F' can be obtained from F by a sequence of the following steps. Let $\exists\forall pG$ be a subformula of F and q be a variable not occurring in F. Replace $\exists\forall pG$ in F by $\exists\forall q(G^q_p)$. ❑

EXAMPLE 11.12 Consider the formula $F = \forall p(\exists p\neg p \vee p \vee \exists p(p \to p))$. We can apply renaming of bound variables to it in the following way. First, we rename the occurrences of the variable p bound by the outermost quantifier $\forall p$ into a new variable q. We obtain the formula $\forall q(\exists p\neg p \vee q \vee \exists p(p \to p))$. Next, we rename p in $\exists p\neg p$ to a new variable r. We obtain the formula $\forall q(\exists r\neg r \vee q \vee \exists p(p \to p))$. This formula looks simpler than the original one since it is clear which quantifiers bind which occurrences of variables in it. Such formulas are called *rectified* and will be defined below. ❑

LEMMA 11.13 *Let a formula F' be obtained from a formula F by renaming bound variables. Then $F \equiv F'$.*

PROOF. **Later** ❑

DEFINITION 11.14 (Rectified Formula) A formula F is called *rectified* if (i) no variable appears both free and bound in F, and (ii) for every variable p, the formula F contains at most one occurrence of quantifiers $\forall p$ and $\exists p$ binding p. ❑

For example, the formula $\exists p\neg p \to \exists p\neg p$ is not rectified, since it contain two quantifiers binding p. The formula $\exists p\neg p \to p$ is not rectified since p has both free and bound occurrences in it. Evidently, both (i) and (ii) in Definition 11.14 can be achieved by renaming bound variables. Therefore, we have the following result.

LEMMA 11.15 *Every formula can be transformed into an equivalent rectified formula by renaming bound variables.* ❑

11.2 Normal Forms

In this section we consider two normal forms of quantified boolean formulas. For propositional formulas, we introduced the notions of CNF and clausal form. For quantified boolean

formulas, we will first introduce a notion of prenex form, that is, a form in which all quantifiers are put in front of a quantifier-free formula. The use of prenex normal forms makes it possible to generalize the splitting algorithm to quantified boolean formulas in a relatively straightforward way in Section 11.3. Then we introduce the notion of conjunctive normal form (CNF) analogous to the notion of CNF for propositional formulas. It will be used in Section 11.4 to define a DLL procedure for quantified boolean formulas. We do not consider the notion of clausal normal form here. In fact, one can represent the structure-preserving clausal normal form transformation as an equivalence-preserving transformation on quantified boolean formulas, see Exercise 11.16.

11.2.1 Prenex Form

In this section we introduce a class of quantified boolean formulas, called *prenex formulas*. Prenex formulas are obtained by putting quantifiers in front of a propositional formula.

DEFINITION 11.16 (Prenex Form) A quantified boolean formula F is said to be in *prenex form*, or simply a *prenex formula* if it has the form $\exists\forall_1 p_1 \ldots \exists\forall_n p_n G$, where G is a propositional formula. A formula F is a *prenex form of a formula* G if F is prenex and $F \equiv G$. ❑

For example, the formula $\exists p \forall q(p \rightarrow q)$ is prenex, while the formula $\exists p(p \rightarrow \forall r r)$ is not, since it has a quantifier $\forall r$ in the range of $\rightarrow$. For every formula of the form $\exists\forall_1 p_1 \ldots \exists\forall_n p_n G$, we will refer to $\exists\forall_1 p_1 \ldots \exists\forall_n p_n$ as a *quantifier prefix*.

In the sequel we will write $\between$ to denote $\wedge$ or $\vee$ in the same way as we used $\exists\forall$ to denote $\forall$ or $\exists$. For example, $p \between q$ may denote either $p \wedge q$ or $p \vee q$. The following algorithm converts formulas without equivalences into their prenex form. We will show later how one can get rid of equivalences in quantified boolean formulas.

ALGORITHM 11.17 (Prenexing) Let F be a formula. By Lemma 11.15 we can rename bound variables in F to make it rectified, so we assume that F is rectified. The transformation of F into its prenex normal form is made using the rewrite rules of Figure 11.1. The rules are applied to F until we obtain a normal form, i.e., no rule is applicable. ❑

EXAMPLE 11.18 Consider the following rectified formula: $\exists q(q \rightarrow p) \rightarrow \neg\forall r(r \rightarrow p) \vee p$. The Prenexing Algorithm applied to this formula yields

$$
\begin{array}{l}
\exists q(q \rightarrow p) \rightarrow \neg\forall r(r \rightarrow p) \vee p \Rightarrow \\
\forall q((q \rightarrow p) \rightarrow \neg\forall r(r \rightarrow p) \vee p) \Rightarrow \\
\forall q((q \rightarrow p) \rightarrow \exists r\neg(r \rightarrow p) \vee p) \Rightarrow \\
\forall q((q \rightarrow p) \rightarrow \exists r(\neg(r \rightarrow p) \vee p)) \Rightarrow \\
\forall q\exists r((q \rightarrow p) \rightarrow \neg(r \rightarrow p) \vee p).
\end{array}
$$

Prenexing rules:

$$\exists\forall pF_1 \between \ldots \between F_n \;\Rightarrow\; \exists\forall p(F_1 \between \ldots \between F_n)$$

$$\forall pF_1 \to F_2 \;\Rightarrow\; \exists p(F_1 \to F_2) \qquad \exists pF_1 \to F_2 \;\Rightarrow\; \forall p(F_1 \to F_2)$$
$$F_1 \to \forall pF_2 \;\Rightarrow\; \forall p(F_1 \to F_2) \qquad F_1 \to \exists pF_2 \;\Rightarrow\; \exists p(F_1 \to F_2)$$

$$\neg\forall pF \;\Rightarrow\; \exists p\neg F \qquad \neg\exists pF \;\Rightarrow\; \forall p\neg F$$

Figure 11.1: Prenexing rules

The quantifiers pushed out at each transformation step are shaded in this picture. Note that a different sequence of transformation can result in a different (but equivalent) prenex form. For example,

$$\begin{array}{l}\exists q(q \to p) \to \neg\forall r(r \to p) \vee p \Rightarrow \\ \exists q(q \to p) \to \exists r\neg(r \to p) \vee p \Rightarrow \\ \exists q(q \to p) \to \exists r(\neg(r \to p) \vee p) \Rightarrow \\ \exists r(\exists q(q \to p) \to \neg(r \to p) \vee p) \Rightarrow \\ \exists r\forall q((q \to p) \to \neg(r \to p) \vee p).\end{array}$$

Therefore, the set of rewrite rules for prenexing is not confluent. However, the resulting prenex formula is equivalent to the original one, see Lemma 11.19. ❑

Note that it is essential that the formula used in the prenexing algorithm be rectified. For example, take the formula $F \stackrel{\text{def}}{=} \exists p(p) \wedge p$. This formula is not rectified. The first prenexing rule applied to this formula yields $\exists p(p \wedge p)$. This formula is not equivalent to F.

LEMMA 11.19 *All rules of Figure 11.1 are equivalence-preserving when applied to rectified formulas.*

PROOF. **Later**. ❑

Note also that each of the prenexing rules rewrites a rectified formula into a rectified one. Note also that there is no prenexing rules for equivalence. One cannot define simple prenexing rules for equivalence. Indeed, the formula $\forall pF_1 \leftrightarrow F_2$ is, in general, neither equivalent to $\forall p(F_1 \leftrightarrow F_2)$, nor to $\exists p(F_1 \leftrightarrow F_2)$. Equivalences must be eliminated from the formula before prenexing.

THEOREM 11.20 *Let F be a rectified formula without occurrences of equivalences. Then the Prenexing Algorithm terminates on the input F and returns a prenex form of F.*

PROOF. Termination is obvious. The equivalence of F and a normal form obtained by the algorithm follows immediately from Lemma 11.19. It remains to note that any normal form is a prenex formula. ❑

11.2.2 Conjunctive Normal Forms

In this section we introduce a notion of conjunctive normal form of quantified boolean formulas. Conjunctive normal form is used in many systems implementing satisfiability-checking for such formulas.

DEFINITION 11.21 (CNF) A quantified boolean formula F is in *conjunctive normal form*, or simply *CNF*, if it is either $\bot$, or $\top$, or has the form $\exists\forall_1 p_1 \ldots \exists\forall_n p_n F_1$, where $n \geq 0$ and F_1 is a propositional formula in conjunctive normal form. A formula G is called a *conjunctive normal form of a formula* F if G is equivalent to F and G is in conjunctive normal form. ❑

In other words, a formula is in CNF if it is prenex and its propositional part is in CNF.

One can obtain a simple CNF transformation algorithm by combining Standard CNF Transformation Algorithm 5.4 on page 62 and the Prenexing Algorithm. However, one has to be careful, since the Prenexing Algorithm is subject to some restrictions: firstly, it does not work with formulas containing equivalences, secondly, it can be incorrect on formulas which are not rectified. The first problem is easy to overcome, since the rewrite rules used for the CNF transformation eliminate equivalences anyhow. To cope with the second problem, one has to rename bound variables in newly obtained formulas as soon as these formulas are not rectified, see Example 11.23 below.

ALGORITHM 11.22 (CNF Transformation) An algorithm transforming formulas to their CNFs is given by the union of the rewrite rule systems of Figure 5.2 on page 63 for obtaining CNF of a propositional formula and of Figure 11.1 for obtaining prenex form of rectified formulas. If at any stage we obtain a formula that is not rectified, we also apply renaming of bounded variables to obtain a rectified formula. Given an input formula G, we apply the rewrite rules to it as follows. If G is rectified, then we simply apply any applicable rewrite rule. Otherwise, we rename bound variables in G to obtain a rectified formula. ❑

EXAMPLE 11.23 Consider the following formula:

$$\forall p(\forall q(q \vee p) \leftrightarrow p).$$

Let us transform this formula into CNF. We cannot apply any prenexing rule to this formula, because the quantifier $\forall q$ is in the scope of equivalence $\leftrightarrow$. So we use the rewrite rule for removing equivalence obtaining

$$\forall p((\forall q(q \vee p) \rightarrow p) \wedge (p \rightarrow \forall q(q \vee p))).$$

This formula is not rectified despite that the original formula was rectified, so we have to rename bound variables in it to obtain a rectified one:

$$\forall p((\forall q(q \vee p) \rightarrow p) \wedge (p \rightarrow \forall r(r \vee p))).$$

Now we can apply prenexing rules to obtain a prenex formula:

$$\begin{array}{ll}
\forall p((\forall q(q \vee p) \to p) \wedge (p \to \forall r(r \vee p))) & \Rightarrow \\
\forall p((\forall q(q \vee p) \to p) \wedge \forall r(p \to (r \vee p))) & \Rightarrow \\
\forall p \forall r((\forall q(q \vee p) \to p) \wedge (p \to (r \vee p))) & \Rightarrow \\
\forall p \forall r(\exists q((q \vee p) \to p) \wedge (p \to (r \vee p))) & \Rightarrow \\
\forall p \forall r \exists q(((q \vee p) \to p) \wedge (p \to (r \vee p))). &
\end{array}$$

To obtain a formula in CNF, it remains to apply the rewrite rules used to obtain CNF of a propositional formula:

$$\begin{array}{ll}
\forall p \forall r \exists q(((q \vee p) \to p) \wedge (p \to (r \vee p))) & \Rightarrow \\
\forall p \forall r \exists q((\neg(q \vee p) \vee p) \wedge (p \to (r \vee p)))) & \Rightarrow \\
\forall p \forall r \exists q(((\neg q \wedge \neg p) \vee p) \wedge (p \to (r \vee p))) & \Rightarrow \\
\forall p \forall r \exists q(((\neg q \wedge \neg p) \vee p) \wedge (\neg p \vee r \vee p)) & \Rightarrow \\
\forall p \forall r \exists q((\neg q \vee p) \wedge (\neg p \vee p) \wedge (\neg p \vee r \vee p)). &
\end{array}$$

The resulting formula is in CNF. ❑

THEOREM 11.24 (CNF) *Let G be a quantified boolean formula. Then the algorithm terminates on G and returns a CNF of G.*

PROOF. **Later** ❑

11.3 Checking Satisfiability of Quantified Boolean Formulas by Splitting

In this and the next sections we will modify the splitting algorithm and the DLL algorithm to deal with quantified boolean formulas. The new algorithms are similar to their propositional analogues, except for two points. Firstly, the universally quantified variables are treated differently from the existentially quantified ones. Secondly, we cannot any more split on variables in an arbitrary order.

Before formulating a satisfiability-checking algorithm, we will make a few simplifying assumptions. The splitting algorithm for propositional formulas is based on Lemma 4.4: $I \models \neg p$, then F is equivalent to $F_p^\perp$ in I, and similarly for $I \models p$. We need a similar lemma for quantified boolean formulas, but we also have to treat quantified variables. By Lemma 11.7 satisfiability-checking for arbitrary quantified boolean formulas can be reduced to satisfiability-checking for closed quantified boolean formulas. By Lemma 11.6, for closed formulas the notions of truth in an interpretation and satisfiability coincide, so there is no need to deal with interpretations at all.

The following lemma is a syntactic analogue of Lemma 4.4 which serves as a basis for the splitting algorithm.

Simplification rules for $\top$:

$$\neg\top \;\Rightarrow\; \bot$$
$$\top \wedge F_1 \wedge \ldots \wedge F_n \;\Rightarrow\; F_1 \wedge \ldots \wedge F_n$$
$$\top \vee F_1 \vee \ldots \vee F_n \;\Rightarrow\; \top$$
$$F \to \top \;\Rightarrow\; \top \qquad \top \to F \;\Rightarrow\; F$$
$$F \leftrightarrow \top \;\Rightarrow\; F \qquad \top \leftrightarrow F \;\Rightarrow\; F$$
$$\forall p\top \;\Rightarrow\; \top$$
$$\exists p\top \;\Rightarrow\; \top$$

Simplification rules for $\bot$:

$$\neg\bot \;\Rightarrow\; \top$$
$$\bot \wedge F_1 \wedge \ldots \wedge F_n \;\Rightarrow\; \bot$$
$$\bot \vee F_1 \vee \ldots \vee F_n \;\Rightarrow\; F_1 \vee \ldots \vee F_n$$
$$F \to \bot \;\Rightarrow\; \neg F \qquad \bot \to F \;\Rightarrow\; \top$$
$$F \leftrightarrow \bot \;\Rightarrow\; \neg F \qquad \bot \leftrightarrow F \;\Rightarrow\; \neg F$$
$$\forall p\bot \;\Rightarrow\; \bot$$
$$\exists p\bot \;\Rightarrow\; \bot$$

Figure 11.2: Simplification rules for quantified boolean formulas

LEMMA 11.25 *(i) A closed formula $\forall pF$ is true if and only if the formulas $F_p^\bot$ and $F_p^\top$ are true. (ii) A closed formula $\exists pF$ is true if and only if at least one of the formulas $F_p^\bot$ and $F_p^\top$ is true.*

PROOF. **Later** ❑

The splitting algorithm for propositional formulas is based on two ideas: splitting into $F_p^\bot$ and $F_p^\top$ and simplification of formulas containing $\bot$ and $\top$. When we extend it to quantified boolean formulas, we will have to extend both the treatment of splitting and the simplification rules. For the latter, we add simplification rules for quantifiers to the rules of Figure 4.5 on page 46. The extended set of rules is given in Figure 11.2. It is not hard to prove that these rewrite rules are equivalence-preserving, i.e., the right-hand side of every rule is equivalent to its left-hand size.

Let us introduce a useful notation. Let $F = \exists\forall_1 p_1 \ldots \exists\forall_n p_n G$ be a prenex formula such that G is a propositional formula. We call the *outermost prefix* of F the longest subsequence $\exists\forall_1 p_1 \ldots \exists\forall_n p_k$ of $\exists\forall_1 p_1 \ldots \exists\forall_n p_n$ such that $\exists\forall_1 = \ldots = \exists\forall_k$. For example, the outermost prefix of any formula of the form $\forall p_1 \exists p_2 \exists p_3 G'$ is $\forall p_1$. The outermost prefix of $\exists p \exists q (p \wedge q)$ is $\exists p \exists q$.

ALGORITHM 11.26 (Splitting) The splitting algorithm for quantified boolean formulas is shown in Figure 11.3. The algorithm is parametrized by a function $select\_signed\_variable(G)$, where G is a closed prenex formula. This function returns a pair (p, b) such that p is a variable occurring in the outermost prefix of G and b is a boolean value. The function $simplify(F)$ simplifies the formula F using the rules of Figure 11.2. The input to the algorithm is a closed rectified prenex formula. ❑

Let us make a few comments about this algorithm. The variable p selected by the function $select\_signed\_variable$ is used for splitting. When the quantifier $\exists\forall$ is $\forall$, we use the equivalence $\forall pG' \equiv (G')_p^{G_1} \wedge (G')_p^{G_2}$. We always have $\{G_1, G_2\} = \{\bot, \top\}$, so the function $select\_signed\_variable$ not only selects the variable p, but also decides which one

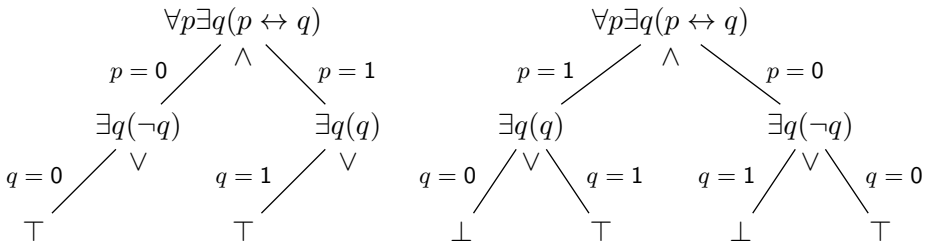
procedure $splitting(G)$
input: closed rectified prenex formula G
output: 0 or 1
parameters: function $select\_signed\_variable$
begin
  $G := simplify(G)$
  if $G = \bot$ **then return** 0
  if $G = \top$ **then return** 1
  Let G have the form $\exists\forall p_1 \ldots \exists\forall p_k G_1$
  $(p, b) := select\_signed\_variable(G)$
  Let G' be obtained from G by deleting $\exists\forall p$ from its outermost prefix
  if $b = 0$
    then $(G_1, G_2) := (\bot, \top)$
    else $(G_1, G_2) := (\top, \bot)$
  case $(splitting((G')_p^{G_1}),\ \exists\forall)$ **of**
    $(0, \forall) \Rightarrow$ **return** 0
    $(0, \exists) \Rightarrow$ **return** $splitting((G')_p^{G_2})$
    $(1, \forall) \Rightarrow$ **return** $splitting((G')_p^{G_2})$
    $(1, \exists) \Rightarrow$ **return** 1
end

Figure 11.3: The splitting algorithm for quantified boolean formulas

of the formulas $\bot, \top$ should be tried first. If $(G')_p^{G_1}$ evaluates to 0, then $\forall p G'$ evaluates to 0 too, so there is no need to evaluate $(G')_p^{G_2}$. The case when $\exists\forall$ is $\exists$ is similar.

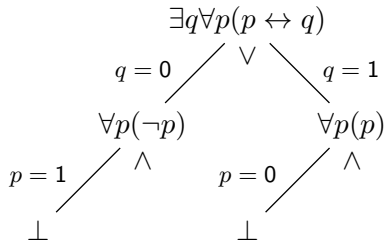
As in the case of the splitting algorithm for propositional formulas, the behaviour of this algorithm is best illustrated by a tree whose nodes are labelled by formulas G such that $splitting(G)$ is called. This tree is called the *splitting tree*. The arcs in the splitting tree are labelled by signed atoms $p = 0$ or $p = 1$ selected by the function $select\_signed\_variable$. The boolean value 0 (respectively 1) is used when we try $(G')_p^{\bot}$ (respectively, $(G')_p^{\top}$) first. Unlike the propositional case, there are two kinds of nodes: AND-nodes (additionally labelled by $\wedge$) and OR-nodes (additionally labelled by $\vee$). They correspond respectively to the cases when $\exists\forall = \forall$ and $\exists\forall = \exists$ in the algorithm. When we have an AND-node (respectively, OR-node) in the tree, the result is the conjunction (respectively, the disjunction) of the results collected on the branches. Consider two examples.

EXAMPLE 11.27 Consider the (true) formula $\forall p \exists q (p \leftrightarrow q)$. Two different splitting trees for this formula are given below.



As one can see from this example, the order of selecting branches can influence the size of the splitting tree (but not the result!). ❑

EXAMPLE 11.28 Consider the (false) formula $\exists q \forall p(p \leftrightarrow q)$. A splitting tree for this formula is given below.



It follows from these examples that, in general, one cannot select arbitrary variables to split upon, since the result can then become incorrect. The split variables must be selected from the outermost quantifier prefix $\exists\forall p_1 \ldots \exists\forall p_k$. ❑

It follows from Monotonic Replacement Theorem 11.8 that for variables having only occurrences of the same polarity, it is not necessary to consider two truth values, which gives us an analogue of the pure atom rule. However, this analogue is not so straightforward as for propositional formulas. Consider, for example, the formula $F = \exists p(p)$. This formula is true. The variable p has only a positive occurrence in it, so to get the value 1, we have to replace this occurrence by $\top$. Consider now the formula $G = \forall p(p)$. This formula is false, but to get the value 0 we have to replace p by $\bot$ rather than by $\top$. As in the case of propositional formulas, it is sufficient to consider a single value for p, but now this value depends not only on the polarity, but also on the quantifier binding p.

The pure atom rule for quantified boolean formulas is based on the following lemma which generalizes Pure Atom Lemma 4.15 on page 54.

LEMMA 11.29 (Pure Atom) *Let p be a boolean variable and F be a formula.*

(*1*) *If all occurrences of p in F are positive, then $\exists pF$ is equivalent to $F_p^\top$ and $\forall pF$ is equivalent to $F_p^\bot$.*

(2) *If all occurrences of p in F are negative, then $\exists pF$ is equivalent to $F_p^{\bot}$ and $\forall pF$ is equivalent to $F_p^{\top}$.*

PROOF. **Later ...**. ❑

Consider, for example, the formula $\exists p \forall q(p \to q)$. The variable p has only negative occurrences in $\forall q(p \to q)$, so by the Pure Atom Lemma this formula is equivalent to $\forall q(\bot \to q)$. By simplifying this formula, we obtain $\top$ without even considering any truth value for q.

11.4 The DLL Algorithm for Quantified Boolean Formulas

As in the case of propositional formulas, the clausal analogue of the splitting algorithm is obtained by considering formulas in CNF and adding unit propagation. We will also refer to it as the DLL algorithm. However, unit propagation for DLL formulas is different from that for propositional formulas in the case of universally quantified variables. To illustrate the difference, consider the simple formula $\forall p(p)$. If we apply unit propagation to the set of clauses $\{p\}$, we obtain the empty set of clauses, which is satisfiable while the formula $\forall pp$ is unsatisfiable. Therefore, unit propagation does not work correctly for this example and should be modified. The difference with unit propagation in the case of propositional satisfiability comes from the fact that p is universally quantified, so any formula of the form $\forall p(p \wedge F)$ is unsatisfiable. In contrast, any formula $\exists p(p \wedge F)$ is equivalent to $F_p^{\top}$, so for existentially quantified variables unit propagation can be defined as in the case of propositional satisfiability.

DEFINITION 11.30 (Unit Propagation) Let S be a set of clauses and Q be a sequence of quantifiers $\exists\forall_1 p_1 \ldots\ldots \exists\forall_n p_n$. We say that a set of clauses S' is obtained from S by *unit propagation with respect to* Q if S' is obtained from S by repeatedly performing the following transformation: if S contains a unit clause consisting of one literal L of the form p or $\neg p$, then

(1) if Q contains $\forall p$ then replace S by the set $\{\Box\}$;

(2) otherwise do the following:

   (a) remove from S every clause of the form $L \vee C'$;
   (b) replace in S every clause of the form $\tilde{L} \vee C'$ by the clause C'. ❑

ALGORITHM 11.31 (DLL for quantified boolean formulas) The algorithm is shown in Figure 11.4. The function *select_literal* selects a literal p or $\neg p$ such that p occurs in the outermost quantifier prefix of $\exists\forall p_1 \ldots \exists\forall p_n$. To evaluate a closed rectified formula G of the form $\exists\forall_1 p_1 \ldots \exists\forall_n p_n G$, where G is a propositional formula $C_1 \wedge \ldots \wedge C_m$ in CNF, we run the algorithm on the sequence of quantifiers ($\exists\forall_1 p_1 \ldots \exists\forall_n p_n$ and set of clauses $\{C_1, \ldots, C_m\}$). ❑

procedure $DLL(Q, S)$
input: sequence of quantifiers $Q = \exists\forall_1 p_1 \ldots \exists\forall_n p_n$, set of clauses S
output: 0 or 1
parameters: function $select\_literal$
begin
  S := $propagate(Q, S)$
  if S is empty **then return** 1
  if S contains $\Box$ **then return** 0
  L := $select\_literal(\exists\forall_1 p_1 \ldots \exists\forall_n p_n, S)$
  Let p be the variable of L
  Let Q_1 be obtained by deleting $\exists\forall_1 p$ from Q
  case $(DLL(Q_1, S \cup \{L\}),\ \exists\forall_1)$ **of**
    $(0, \forall) \Rightarrow$ **return** 0
    $(0, \exists) \Rightarrow$ **return** $DLL(Q_1, S \cup \{\tilde{L}\})$
    $(1, \forall) \Rightarrow$ **return** $DLL(Q_1, S \cup \{\tilde{L}\})$
    $(1, \exists) \Rightarrow$ **return** 1
end

Figure 11.4: The DLL algorithm for quantified boolean formulas

EXAMPLE 11.32 Consider the following formula in CNF:

$$\exists p \forall q \exists r((p \vee q \vee \neg r) \wedge (p \vee \neg q \vee r) \wedge (\neg p \vee q \vee r) \wedge (\neg p \vee q \vee \neg r)).$$

The DLL algorithm on this formula is illustrated in Figure 11.5. The splitting steps are denoted by solid lines, while the unit propagation steps by dashed lines. The formula is true. ❑

One can add the pure literal rule to the DLL algorithm for quantified boolean formulas using Lemma 11.29. In addition to the pure literal rule, the algorithm also admits another rule which can be applied to eliminate some literals in clauses.

Consider a quantifier prefix Q. Introduce an order $>_Q$ on variables as follows: $p > q$ if p occurs before q in Q. For example, when $Q = \forall p \exists q \forall r$, we have $p >_Q q >_Q r$. This order is naturally extended to a partial order on literals: for two literals L_1, L_2 with variables p_1, p_2 respectively, we have $L_1 > L_2$ if $p_1 > p_2$. For example, for the prefix Q as above, we have $p >_Q \neg q$, $\neg p >_Q \neg q$, and $\neg p >_Q q$ because $p >_Q q$. We call a literal L *universal* (respectively, existential) in Q, if Q contains $\forall p$ (respectively, $\exists p$) and p occurs in L. For example for the prefix Q as above, the universal literals in this prefix are $p, \neg p, r, \neg r$, while the existential ones are $q, \neg q$.

LEMMA 11.33 (Universal Literal Deletion) *Let $G = Q(C_1 \wedge \ldots \wedge C_n)$ be a closed rectified CNF formula, where Q is its quantifier prefix. Suppose that some C_i is not a tautology,*

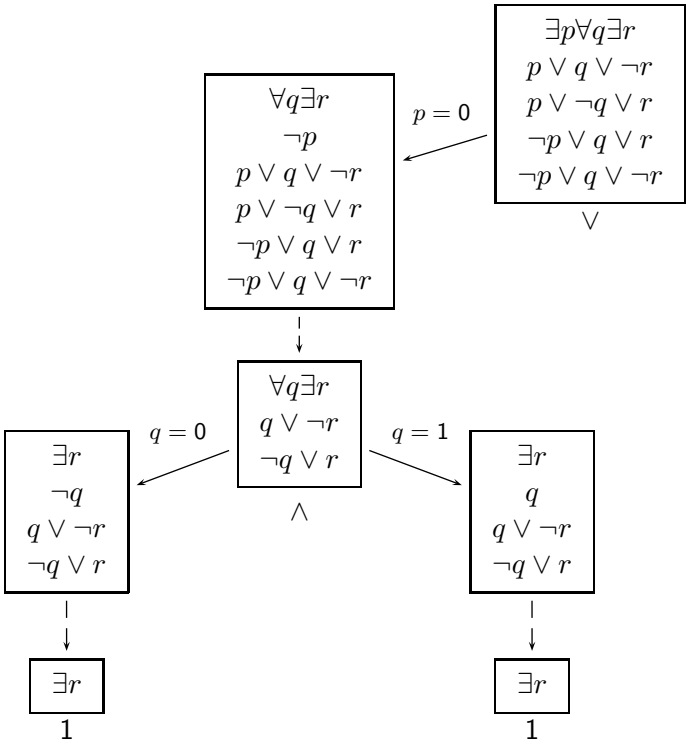


Figure 11.5: DLL algorithm on the formula of Example 11.32

contains a literal L universal in Q, and this literal is smaller w.r.t. $>_Q$ than any existential literal in C_i. Then removal of L from C_i does not change the truth value of G.

PROOF. **Later . . .** ❑

Consider an example. Suppose that $Q = \forall p \exists q \forall r \exists s$. Then any literal containing r can be removed from every non-tautological clause which does not contain s.

This lemma also implies that, if there exists a clause C_i containing only literals universal in Q, then the formula G is unsatisfiable. Indeed, all literals of this clause can be removed by Lemma 11.33, so we obtain the empty clause.

11.5 OBDD-Based Algorithms for Quantified Boolean Formulas

The expressive power of quantified boolean formulas is the same as the expressive power of propositional formulas. This means that for every quantified boolean formula F one can effectively find a propositional formula G equivalent to F. In general G cannot be found

in polynomial time, unless NP=PSPACE, but the observation about the equivalence of the expressive powers of propositional and quantified boolean formulas shows that quantified boolean formulas can be represented by OBDDs.

One can built OBDDs from quantified boolean formulas using OBDD Construction Algorithm 10.14 on page 144 modified for quantified boolean formula. The only change to the algorithm is that closed formulas F are evaluated to 0 or 1 using any satisfiability-checking algorithm for quantified boolean formula. There exists, however, a different algorithm for building an OBDD based on *quantifier elimination*. We will describe such an algorithm below, since it will also be useful for model checking procedures defined in the subsequent chapters.

Quantifier elimination takes a quantified boolean formula and returns an equivalent propositional formula. We are interested in performing quantifier elimination directly on OBDDs. That is, we will define an algorithm which, given an OBDD d for a formula F, computes an OBDD for a formula $\exists p_1 \ldots \exists p_n F$ or $\forall p_1 \ldots \forall p_n F$, where $\{p_1, \ldots, p_n\}$ is a subset of variables of F.

As any other OBDD algorithm, the quantifier elimination algorithm analyzes the OBDD d for F top-down, but builds the resulting OBDD bottom-up. As any other OBDD algorithm, it works on a global dag D containing at least the input OBDD for F, but maybe some other OBDDs. For simplicity, we only present an algorithm which eliminates existential quantifiers, the reader can easily build a similar algorithm for eliminating universal quantifiers. As one can guess, the algorithm for eliminating existential quantifiers is very similar to the algorithm for building the disjunction of OBDDs, see Figure 10.12 on page 151.

ALGORITHM 11.34 (Quantifier Elimination in OBDDs) An algorithm for quantifier elimination in OBDDs is given in Figure 11.6. In this algorithm D is a global dag which satisfies all conditions 1–6 of the definition of BDDs and which contains OBDDs rooted at $n_1, \ldots, n_m$ as subdags. The function $max\_variable(n_1, \ldots, n_m)$ returns the maximal variable occurring in the OBDDs rooted at $n_1, \ldots, n_m$. ❑

THEOREM 11.35 *Let $n_1, \ldots, n_m$ be nodes representing formulas $F_1, \ldots, F_m$, respectively. Then $ex\_elim(\{p_1, \ldots, p_k\}, \{n_1, \ldots, n_m\})$ returns a node n representing a formula equivalent to $\exists p_1 \ldots \exists p_k (F_1 \vee \ldots \vee F_m)$.* ❑

Exercises

EXERCISE 11.1 Show how to derive Equivalent Replacement Theorem from Monotonic Replacement Theorem for formulas without $\leftrightarrow$. ❑

EXERCISE 11.2 Draw parse trees for the following formulas:

(1) $\forall p(p \rightarrow \exists q(q \wedge \neg p) \vee q) \leftrightarrow p \wedge \forall q(r \rightarrow p)$.

```
procedure ex_elim({p_1,...,p_k},{n_1,...,n_m})
parameters: a dag D containing n_1,...,n_m and 0, 1 as nodes
input: nodes n_1,...,n_m representing F_1,...,F_m in D
output: a node n representing ∃p_1 ... ∃p_k(F_1 ∨ ... ∨ F_m) in (modified) D
begin
  if m = 0 then return 0
  if m = 1 and k = 0 then return n_1
  if some n_i is 0 then
    return ex_elim({p_1,...,p_k},{n_1,...,n_{i-1},n_{i+1},...,n_m})
  if some n_i is 1 then return 1
  p := max_variable(n_1,...,n_m)
  forall i = 1...m
    if n_i is labelled by p
      then (l_i,r_i) := (neg(n_i),pos(n_i))
      else (l_i,r_i) := (n_i,n_i)
  if p ∈ {p_1,...,p_k}
    then return ex_elim({p_1,...,p_k}-{p},{l_1,...,l_m,r_1,...,r_m})
    else
      k_1 := ex_elim({p_1,...,p_k},{l_1,...,l_m})
      k_2 := ex_elim({p_1,...,p_k},{r_1,...,r_m})
      if k_1 = k_2 then return k_1
      return integrate(k_1,p,k_2,D)
end
```

Figure 11.6: An algorithm for eliminating existential quantifiers in OBDDs

(2) $\forall p(p \rightarrow \exists q(q \rightarrow \forall r(p \vee q \vee r)))$.

(3) $(\forall p(\exists r\ r \rightarrow p) \rightarrow q) \wedge (\forall q(\exists r\ r \rightarrow q) \rightarrow p)$. ❑

EXERCISE 11.3 For each of the formulas of Exercise 11.2, underline free occurrences of variables in it. Which of these formulas are closed? ❑

EXERCISE 11.4 Which of the formulas of Exercise 11.2, are rectified? Rectify the non-rectified formulas by renaming bound variables in them. ❑

EXERCISE 11.5 Transform the following formulas into prenex form:

(1) $\forall p\ \neg p \vee \forall p\ p \rightarrow \neg p$;

(2) $\forall p\ p \rightarrow \exists p\ p$. ❑

EXERCISE 11.6 Transform the following formulas into CNF:

(1) $\forall p(p \leftrightarrow (\forall q \exists r(q \rightarrow p \wedge r)))$;

(2) $\forall q(\forall p(p \rightarrow q) \rightarrow \forall p(\neg q \rightarrow p))$;

(3) $\forall q(\forall p(\neg q \rightarrow p) \rightarrow \forall p(p \rightarrow q))$. ❑

EXERCISE 11.7 Evaluate the following formulas using the Splitting Algorithm:

(1) $\exists r \forall q \exists p(p \leftrightarrow ((p \rightarrow r) \leftrightarrow q))$;

(2) $\forall r \forall q \exists p(p \leftrightarrow ((p \rightarrow r) \leftrightarrow q))$;

(3) $\forall q(\forall p(p \rightarrow q) \rightarrow \forall p(\neg q \rightarrow p))$;

(4) $\forall q(\forall p(\neg q \rightarrow p) \rightarrow \forall p(p \rightarrow q))$. ❑

EXERCISE 11.8 For each formula of Exercise 11.7, make a table consisting of its subformulas, their positions and polarities. ❑

EXERCISE 11.9 For each of the following formulas, show how one can simplify the formula using Pure Atom Lemma 11.29.

(1) $\forall p(p \rightarrow \exists q \exists r((q \wedge \neg r) \vee \neg p))$;

(2) $\forall q(\forall p(\neg q \rightarrow p) \rightarrow \forall p(p \rightarrow q))$. ❑

EXERCISE 11.10 Design an algorithm for eliminating universal quantifiers in OBDDs, similar to Algorithm 11.34. ❑

EXERCISE 11.11 Evaluate the formulas of Example 11.7 using the following algorithm. First, build an OBDD which represents the propositional part of the formula. Then apply quantifier elimination to the OBDD until all quantifiers are eliminated. ❑

EXERCISE 11.12 Consider the following formula in CNF:

$$\forall s \exists p \forall q \exists r((\neg p \vee q \vee r \vee s) \wedge (\neg p \vee q \vee \neg r \vee s) \wedge (p \vee q) \wedge (p \vee q \vee \neg r \vee \neg s)).$$

Evaluate this formula using only the pure literal rule and unit propagation. ❑

EXERCISE 11.13 For each of the following formulas in CNF, evaluate the formula using only the pure literal rule, universal literal deletion, and unit propagation.

(1) $\forall p \exists q \forall s \exists r((p \vee q \vee s) \wedge (p \vee \neg q \vee \neg r) \wedge (p \vee \neg q \vee r \vee \neg s))$;

(2) $\exists p \forall q \exists r \forall s((p \vee q \vee s) \wedge (\neg p \vee \neg q \vee r) \wedge (\neg q \vee \neg r \vee s))$. ❑

EXERCISE 11.14 Consider the following formula in CNF:

$$\exists p \exists q \forall r \exists s((p \vee \neg r) \wedge (\neg q \vee r) \wedge (\neg p \vee q \vee s) \wedge (\neg p \vee q \vee r \vee \neg s)).$$

Evaluate this formula using only universal literal deletion and unit propagation. ❑

EXERCISE 11.15 Evaluate each of the following formulas using DLL:

(1) $\forall s \exists p \forall q \exists r((\neg p \vee q \vee r \vee s) \wedge (\neg p \vee q \vee \neg r \vee s) \wedge (p \vee q) \wedge (p \vee q \vee \neg r \vee \neg s))$;

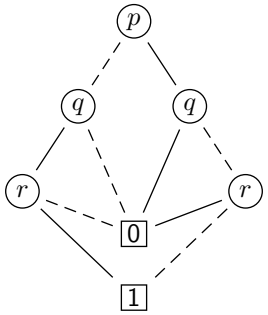
(2) $\forall p \exists q \forall s \exists r((p \vee q \vee s) \wedge (p \vee \neg q \vee \neg r \vee \neg s) \wedge (p \vee \neg q \vee r \vee \neg s))$;

(3) $\exists p \exists q \forall r \exists s((p \vee \neg r) \wedge (\neg q \vee r) \wedge (\neg p \vee q \vee s) \wedge (\neg p \vee q \vee r \vee \neg s))$. ❑

EXERCISE 11.16 The structure-preserving clausal form transformation for propositional formulas introduced new boolean variables as names for subformulas. If we apply it to a formula F, the obtained clausal form G is, in general, not equivalent to F. Suppose that $p_1, \ldots, p_n$ are all new variables introduced during the transformation of F into its clausal form G. Show that F is equivalent to the quantified boolean formula $\exists p_1 \ldots \exists p_n G$. ❑

EXERCISE 11.17 Extend the splitting algorithm to work with non-prenex formulas. ❑

EXERCISE 11.18 A formula F has the following OBDD.



Apply the quantifier elimination algorithm to this OBDD to obtain

(1) the OBDD for the formula $\exists p \exists r F$;

(2) the OBDD for the formula $\exists q F$.

❑