

Chapter 3

Propositional Logic

True/false questions present a statement, and prompt the student to choose whether the statement is truthful. Students typically have a great deal of experience with this type of question. . . . True/false questions are among the easiest to write, and can be scored electronically.

University of Wisconsin, Teaching Academy

There are no real boolean values in Perl. Still every value in Perl is either true or false.

Contents

3.1	Syntax	30
3.2	Semantics	31
3.3	Satisfiability, Validity, and Equivalence	33
3.4	Subformula	35
3.5	Evaluating a Formula in an Interpretation	36
3.6	Evaluating Formulas Using Simplification Rules	37
3.7	Propositional Formulas and Boolean Circuits	39
	Exercises	39

This chapter deals with propositional logic. We describe its syntax and semantics, and introduce the key notions of *satisfiability*, *validity*, and *equivalence*.

A more detailed exposition of propositional logic may be found in standard textbooks on mathematical logic, e.g. Kleene [1952], Smullyan [1968], Lee and Chang [1973], Gallier [1986], and Fitting [1996].

As any other logic, propositional logic can express properties. They are expressed using a language; expressions in this language are called (propositional) *formulas*. These formulas formalize a mathematical analogue of the notion of proposition (assertion, statement,

Connective	Name	Priority
\top	<i>verum</i>	
\perp	<i>falsum</i>	
\neg	<i>negation</i>	1
\wedge	<i>conjunction</i>	2
\vee	<i>disjunction</i>	2
\rightarrow	<i>implication</i>	3
\leftrightarrow	<i>equivalence</i>	4

Figure 3.1: Connectives

sentence). Intuitively, a proposition is anything that can be either true or false. When we manipulate with formulas of propositional logic, we abstract away from the concrete meaning, or *content* of propositions. That is, we consider them only from an abstract viewpoint, so that the meaning of complex propositions is defined in terms of the meanings of simpler ones.

3.1 Syntax

Propositional formulas are written in a propositional language. To fix a propositional language, we fix a countable set P , whose elements will be called *boolean variables* and denoted by p, q, r .

DEFINITION 3.1 (Formula) *Propositional formulas* are defined inductively as follows:

- (1) Every boolean variable is a formula, also called *atomic formula*, or simply *atom*.
- (2) \top and \perp are formulas.
- (3) If A_1, \dots, A_n are formulas, where $n \geq 2$, then $(A_1 \wedge \dots \wedge A_n)$ and $(A_1 \vee \dots \vee A_n)$ are formulas.
- (4) If A is a formula, then $\neg A$ is a formula.
- (5) If A and B are formulas, then $(A \rightarrow B)$ and $(A \leftrightarrow B)$ are formulas.

The symbols \top , \perp , \wedge , \vee , \neg , \rightarrow , and \leftrightarrow used to build formulas are called *connectives*. \square

Note that we gave an inductive definition of propositional formulas in the sense of Section 2.7. The connectives are simply the constructors of this inductive definition. The connectives \wedge and \vee are *varyadic*, that is, they do not have a fixed number of arguments.

The names of all connectives are summarized in Figure 3.1. We will sometimes call formulas by the names of their main (outermost) connective. For example, we will say that

a formula $A_1 \vee \dots \vee A_n$ is a *disjunction* (of formulas A_1, \dots, A_n). To make formulas shorter, we will omit parentheses in them according to the standard conventions given in Figure 3.1 on the preceding page. Connectives with higher priorities bind stronger: for example, $A_1 \rightarrow A_2 \vee A_3 \leftrightarrow \neg A_4$ means $(A_1 \rightarrow (A_2 \vee A_3)) \leftrightarrow \neg A_4$. We will always use parentheses to distinguish connectives of the same priority. For example, we will never write $p \vee q \wedge r$, since this formula can denote both $(p \vee q) \wedge r$ and $p \vee (q \wedge r)$. Likewise, we will never write $p \rightarrow q \rightarrow r$ or $p \leftrightarrow q \leftrightarrow r$.

3.2 Semantics

So far we have only defined the syntax of propositional logic, without giving any meaning to formulas. In real life, the meaning of propositions expressed in natural language depends on the current situation, or the state of the world. For example, the proposition

this book is written in a foreign language (3.1)

may be true under some circumstances and false under other. The situation is similar for other simple (atomic) propositions, for example

this book has a red cover. (3.2)

For more complex propositions, their meaning may also depend on the current state of the world, but in a slightly different way. For example the proposition “this book is written in a foreign language and has a red cover” is true for some books and false for others, but it is true for every book for which propositions (3.1) and (3.2) are both true.

This can be summarized as follows.

- (1) The meaning of simple, atomic propositions depends on their interpretation in the current world.
- (2) The meaning of more complex propositions depends on the meaning of their components.

The semantics of propositional logic is based on these two assumptions. Formally, the semantics is defined through the notion of *interpretation*. An interpretation assigns values to boolean variables.¹ These values are used to define the values of arbitrarily complex formulas, based on the values of these components.

DEFINITION 3.2 (Boolean Value, Interpretation, Truth) A *boolean value*, also called a *truth value*, is either 1 or 0. A *interpretation* for a set of boolean variables P is a mapping from

¹Remember that boolean variables in propositional logic are intended to be formal analogues of simple, atomic propositions.

\wedge	1	0	\vee	1	0	\neg		\rightarrow	1	0	\leftrightarrow	1	0
1	1	0	1	1	1	1	0	1	1	0	1	1	0
0	0	0	0	1	0	0	1	0	1	1	0	0	1

Figure 3.2: Operation tables for connectives

P to the set of boolean values $\{1, 0\}$. We can say that boolean variables are *interpreted* as boolean values. Interpretations will also be called *truth assignments*.

We will now extend interpretations to arbitrary propositional formulas. The definition is by induction.

- (1) $I(\top) = 1$ and $I(\perp) = 0$.
- (2) $I(A_1 \wedge \dots \wedge A_n) = 1$ if and only if $I(A_i) = 1$ for all i .
- (3) $I(A_1 \vee \dots \vee A_n) = 1$ if and only if $I(A_i) = 1$ for some i .
- (4) $I(\neg A) = 1$ if and only if $I(A) = 0$.
- (5) $I(A \rightarrow B) = 1$ if and only if $I(A) = 0$ or $I(B) = 1$.
- (6) $I(A \leftrightarrow B) = 1$ if and only if $I(A) = I(B)$.

If, for a formula A and interpretation I , we have $I(A) = 1$ (respectively, $I(A) = 0$), we say that A is *true* (respectively, *false*) in I , denoted by $I \models A$ (respectively, $I \not\models A$). \square

Essentially, an interpretation I evaluates each formula A to a truth value, so we can speak about the *value* of A in I , that is $I(A)$. Note that the truth value of a compound formula is *uniquely determined* by the truth values of its components. Therefore, logical connectives can be alternatively considered as functions on truth values. The notion of truth can then be illustrated by so-called *operation tables* for all connectives which define the truth value of a compound formula using the truth values of its components. The operation tables for all connectives are given in Figure 3.2. For convenience, we only consider conjunctions and disjunctions of two formulas.

Natural language sentences, even the simplest ones, may be ambiguous, so it may be difficult to say whether they are true or false. For example sentence (3.2) is not necessarily true or false. What if the cover is mainly red, but also has other colors? What if the color is between red and orange, but very close to red? There is a whole range of logics studied in philosophical logic in which there are more than two truth values,² or even an infinite number of truth values, but we will not consider these logics in this book.

It will sometimes be convenient for us to use *empty conjunctions* and *disjunctions*. The empty conjunction is true in every interpretation, while the empty disjunction is always false.

²For example “true”, “false”, and “unknown”.

3.3 Satisfiability, Validity, and Equivalence

In this section we introduce the key notion of satisfiability of formulas and sets of formulas and some related notions.

DEFINITION 3.3 (Model, Satisfiability, Validity, Equivalence) If a formula A is true in an interpretation I we say that I *satisfies* A . We also say that I is a *model* of A . A formula A is *satisfiable* (respectively, *valid*) if it is true in some (respectively, in every) interpretation. Valid formulas are also called *tautologies*. Two formulas A and B are called *equivalent*, denoted $A \equiv B$ if every model of A is a model of B , and vice versa, every model of B is a model of A . \square

We generalize the notions of satisfiability and model to sets of formulas as follows. We say that an interpretation I *satisfies a set of formulas* S , denoted by $I \models S$, if it satisfies every formula in S . If $I \models S$, we also say that I is a *model of* S . A set of formulas is called *satisfiable* if there exists an interpretation which satisfies every formula in this set.

Consider examples. Let p be a boolean variable. Then the formula $p \wedge \neg p$ is unsatisfiable. Each of the formulas p and $\neg p$ is satisfiable but not valid. The formula $p \vee \neg p$ is valid.

For propositional formulas, the notions of satisfiability, validity, and truth in an interpretation coincide. This will not be the case for more general classes of formulas introduced later in this book. The notions of satisfiability and validity are central in logic. *Automated reasoning* is a branch of computer science that deals with mechanized methods of establishing satisfiability or unsatisfiability of sets of formulas and related questions.

It is not hard to argue that two formulas A and B are equivalent if in every interpretation their values coincide, that is, for every interpretation I we have $I(A) = I(B)$.

EXAMPLE 3.4 For all formulas A and B , the following equivalences hold.

$$A \rightarrow \perp \equiv \neg A; \quad (3.3)$$

$$\top \rightarrow A \equiv A; \quad (3.4)$$

$$A \rightarrow B \equiv \neg(A \wedge \neg B); \quad (3.5)$$

$$A \wedge B \equiv \neg(\neg A \vee \neg B); \quad (3.6)$$

$$A \vee B \equiv \neg A \rightarrow B. \quad (3.7)$$

Consider, for example, the first pair of formulas, $A \rightarrow \perp$ and $\neg A$, and let us show that they are equivalent. Take any interpretation I . If $I \models A$, then $I(A \rightarrow \perp) = 0 = I(\neg A)$. If $I \not\models A$, then $I(A \rightarrow \perp) = 1 = I(\neg A)$. In both cases we have $I(A \rightarrow \perp) = I(\neg A)$, so $A \rightarrow \perp \equiv \neg A$. We will show later how to verify such equivalences in general. \square

Let us note the following trivial, but useful, fact showing that the problems of checking equivalence, validity, and satisfiability can be regarded as instances of each other, in a sense.

LEMMA 3.5 (i) A formula A is valid if and only if $\neg A$ is unsatisfiable. (ii) A formula A is satisfiable if and only if $\neg A$ is not valid. (iii) A formula A is valid if and only if A is equivalent to \top . (iv) Formulas A and B are equivalent if and only if the formula $A \leftrightarrow B$ is valid.

PROOF. We will only prove property (iv). To prove the “only if” direction, suppose that A and B are equivalent. Take any interpretation I . If $I \models A$, then by the equivalence $I \models B$, and hence $I \models A \leftrightarrow B$. If $I \not\models A$, then again by the equivalence $I \not\models B$, and hence $I \models A \leftrightarrow B$. In both cases we have $I \models A \leftrightarrow B$, so $A \leftrightarrow B$ is valid. To prove the “if” direction suppose that $A \leftrightarrow B$ is valid. Take any interpretation I . We have $I \models A \leftrightarrow B$. By inspection of the operation table for \leftrightarrow (see Figure 3.2 on page 32) one can see that $I \models A$ if and only if $I \models B$, so A and B are equivalent. \square

Although complexity-related questions are beyond the scope of this book, we would like to note that this lemma also shows that the problems of checking validity and equivalence are polynomial-time equivalent. In fact, validity and equivalence checking are coNP-complete problems, while satisfiability checking is NP-complete. This implies a subtle difference between checking satisfiability and checking unsatisfiability of a formula. To establish satisfiability of a formula A , it is enough to find *some* interpretation that satisfies A . To establish unsatisfiability, one has to check that A is false in *all* interpretations. Strangely enough, this observation may be used to build algorithms which can sometimes establish satisfiability of a formula but can never establish unsatisfiability.

For example, consider the following algorithm. Given a propositional formula A with variables p_1, \dots, p_n , do the following. Select *randomly* boolean values b_1, \dots, b_n for p_1, \dots, p_n , for example, by tossing a coin. This gives us a “random” interpretation $I = \{p_1 \mapsto b_1, \dots, p_n \mapsto b_n\}$. If $I \models A$, then a model of A is found, so terminate and return “satisfiable”. Repeat this procedure some number of times and no model was found, return “I don’t know”. Even if we repeat this procedure a large number of times, there is no guarantee that A is unsatisfiable. We can claim that A is unsatisfiable only with some probability.

Consider now satisfiability checking of *sets* of formulas. For *finite* sets one can reduce satisfiability checking to satisfiability checking for formulas using the following lemma.

LEMMA 3.6 Let $S = \{A_1, \dots, A_n\}$ be a set of formulas. Then S is satisfiable if and only if the formula $A_1 \wedge \dots \wedge A_n$ is satisfiable. \square

The relations between satisfiability and validity for *infinite* sets of formulas are not that straightforward and will be investigated later. Though in practice we do not deal with infinite sets of formulas, they will still be useful when we discuss satisfiability and validity for *first-order* formulas. In many cases the semantics of finite sets of first-order formulas can be conveniently characterized by the semantics of infinite sets of propositional formulas.

Obviously, if an interpretation I satisfies an infinite set of formulas S , it also satisfies every subset of S . It is interesting that satisfiability of infinite sets of formulas can be characterized in terms of satisfiability of finite sets as follows: a set S of formulas is satisfiable

if and only if every finite subset of S is satisfiable too. This property is called *compactness* and proved in Theorem 7.28. Note that compactness has an interesting corollary: if an infinite set S of formulas is unsatisfiable, then it contains a *finite* unsatisfiable subset.

3.4 Subformula

The value of a compound formula is uniquely determined, using the operation tables, by the value of its components, or its *immediate subformulas*.

DEFINITION 3.7 (Subformula) The notion of *subformula* and *immediate subformula* of a formula is defined inductively as follows.

- (1) The formulas A_1, \dots, A_n are the immediate subformulas of the formulas $A_1 \wedge \dots \wedge A_n$ and $A_1 \vee \dots \vee A_n$.
- (2) The formulas A is the immediate subformula of the formula $\neg A$.
- (3) The formulas A_1, A_2 are the immediate subformulas of the formulas $A_1 \rightarrow A_2$ and $A_1 \leftrightarrow A_2$.
- (4) Every formula A is a subformula of itself.
- (5) If A_1 is a subformula of A_2 and A_2 is an immediate subformula of A_3 , then A_1 is a subformula of A_3 .

A subformula A of B is called *proper* if $A \neq B$. □

For example, the formula $p_1 \rightarrow p_2 \wedge p_3 \wedge \neg p_1$ has the following subformulas: $p_1, p_2, p_3, \neg p_1, p_2 \wedge p_3 \wedge \neg p_1$, and the formula $p_1 \rightarrow p_2 \wedge p_3 \wedge \neg p_1$ itself. Its immediate subformulas are p_1 and $p_2 \wedge p_3 \wedge \neg p_1$.

It is not hard to argue that the subformula relation is a reflexive and transitive closure of the notion of immediate subformula. The notions of subformula and immediate subformula can be obtained by specializing the general notions of expression and subexpression, see Section 2.7.1, to formulas.

Let us now prove several properties of truth, satisfiability, and validity. These properties will be used, explicitly or implicitly, in justifying satisfiability-checking algorithms. One of the most important properties for us is that the semantics of formulas is preserved under equivalence.

Since the truth value of a formula in an interpretation is uniquely determined by the truth values of its immediate subformulas, we have the following result.

LEMMA 3.8 (Equivalent Replacement) *Let I be an interpretation, a formula A_1 be a subformula of a formula B_1 and $I \models A_1 \leftrightarrow A_2$. Let the formula B_2 be obtained from B_1 by replacement of one or more occurrences of A_1 by A_2 . Then $I \models B_1 \leftrightarrow B_2$.* □

Note that using the notation introduced in Section 2.7.3 we can reformulate this lemma as follows: if $I \models A_1 \leftrightarrow A_2$, then $I \models B[A_1] \leftrightarrow B[A_2]$. This lemma immediately implies the following theorem.

THEOREM 3.9 (Equivalent Replacement) *Let A_1 be a subformula of a formula B_1 and $A_1 \equiv A_2$. Let the formula B_2 be obtained from B_1 by replacement of one or more occurrences of A_1 by A_2 . Then $B_1 \equiv B_2$.* \square

3.5 Evaluating a Formula in an Interpretation

In this section we consider how to evaluate a formula in an interpretation. This can be formalized as the following decision problem.

DEFINITION 3.10 (Formula Evaluation) *Formula Evaluation* is the following decision problem. An instance is a pair (A, I) , where A is formula and I is an interpretation. The answer is “yes” if $I \models A$. \square

This decision problem has a close relation to the so-called *circuit value* problem discussed below.

We can evaluate formulas in interpretations by a straightforward use of the definition of the value of a formula. To this end, we can first evaluate its immediate subformulas, and then evaluate the formula itself using the operation tables of Figure 3.2 on page 32. Let us try to evaluate some formulas.

EXAMPLE 3.11 Consider the formula $A = (p \rightarrow q) \wedge (p \wedge q \rightarrow r) \rightarrow (p \rightarrow r)$ and consider the interpretation $I = \{p \mapsto 1, q \mapsto 0, r \mapsto 1\}$. Put all the subformulas of this formula in a table in such a way that every formula is above all of its proper subformulas, see Figure 3.3.³ Then evaluate all the subformulas starting from the smallest ones (in the bottom of the table).

For example, the formula $(p \rightarrow q) \wedge (p \wedge q \rightarrow r)$ on line 3 is the conjunction of the formulas $p \rightarrow q$ and $p \wedge q \rightarrow r$ situated on lines 5 and 4, respectively. Since the formula on line 5 has the value 0 and the formula on line 4 has the value 1, their conjunction evaluates to 0. \square

Observe that it is not necessary to evaluate all the subformulas of a formula A in order to obtain the value of A . Consider, for example, the formula of Example 3.11. Since we know that r is true in I , we can immediately obtain that $p \rightarrow r$ is true in I , too (see the operation table for the implication). Likewise, $p \rightarrow r$ implies that the whole formula $(p \rightarrow q) \wedge (p \wedge q \rightarrow r) \rightarrow (p \rightarrow r)$ is true. Therefore, instead of evaluating all the subformulas of A it was enough to evaluate only three subformulas, including A itself. Moreover, it even was not necessary to know the values of p and q : the formula is true in every interpretation in which r is true.

³For a better illustration, we repeat subformulas p , q and r having multiple occurrences in the formula.

	subformula				value
1	$(p \rightarrow q) \wedge (p \wedge q \rightarrow r) \rightarrow (p \rightarrow r)$				1
2				$p \rightarrow r$	1
3	$(p \rightarrow q) \wedge (p \wedge q \rightarrow r)$				0
4				$p \wedge q \rightarrow r$	1
5	$p \rightarrow q$				0
6				$p \wedge q$	0
7	p	p		p	1
8		q		q	0
9			r	r	1

Figure 3.3: Evaluation of formula using truth tables

3.6 Evaluating Formulas Using Simplification Rules

In this section we will show that evaluation of a formula can be done in a purely syntactic way. We will introduce a rewrite rule system which can be used to evaluate formulas in interpretations. To this end, we make the following observation. Suppose that we evaluate A in I and a subformula B of A is true in I . In this case B and \top have the same truth value in I , so can replace B by \top without affecting the truth value of A . Likewise, we can replace any subformula having the value 0 by \perp .

Using this observation, we can evaluate propositional formulas in an interpretation I as follows:

- (1) For every atom p true in I replace p by \top ; for every atom q false in I replace q by \perp .
- (2) Use the rewrite rules corresponding to the operation tables of Figure 3.2 on page 32 to simplify the resulting formula. For example, the rewrite rules for \rightarrow have the following form⁴

$$\begin{array}{l} \top \rightarrow \top \Rightarrow \top; \quad \top \rightarrow \perp \Rightarrow \perp; \\ \perp \rightarrow \top \Rightarrow \top; \quad \perp \rightarrow \perp \Rightarrow \perp. \end{array} \quad (3.8)$$

Note that every rewrite rule replaces a subformula by an equivalent one, so the value of the formula will not change.

- (3) If the formula finally rewrites into \top , then it is true; if it rewrites into \perp , then it is false.

We can use a stronger form of rewrite rules for formula evaluation. The idea can be illustrated by rewrite rules (3.8) for the implication. Note that $A \rightarrow \top$ is equivalent to

⁴We do not give here the complete list of all the rewrite rules, since later we will introduce their simplified version.

$\begin{aligned} \top \wedge \dots \wedge \top &\Rightarrow \top \\ \perp \wedge A_1 \wedge \dots \wedge A_n &\Rightarrow \perp \end{aligned}$	$\begin{aligned} \top \vee A_1 \vee \dots \vee A_n &\Rightarrow \top \\ \perp \vee \dots \vee \perp &\Rightarrow \perp \end{aligned}$
$\begin{aligned} \neg \top &\Rightarrow \perp \\ \neg \perp &\Rightarrow \top \end{aligned}$	
$\begin{aligned} A \rightarrow \top &\Rightarrow \top \\ \perp \rightarrow A &\Rightarrow \top \\ \top \rightarrow \perp &\Rightarrow \perp \end{aligned}$	$\begin{aligned} \top \leftrightarrow \top &\Rightarrow \top \\ \top \leftrightarrow \perp &\Rightarrow \perp \\ \perp \leftrightarrow \top &\Rightarrow \perp \\ \perp \leftrightarrow \perp &\Rightarrow \top \end{aligned}$

Figure 3.4: Rewrite rules for formula evaluation

\top independently of the A . Likewise, $\perp \rightarrow A$ is equivalent to \top independently of A . Therefore, we can replace the four rewrite rules of (3.8) by the following three rewrite rules:

$$A \rightarrow \top \Rightarrow \top; \quad \perp \rightarrow A \Rightarrow \top; \quad \top \rightarrow \perp \Rightarrow \perp.$$

These rewrite rules use the variable A ranging over arbitrary propositional formulas.

The rewrite rules for evaluating a formula are given in Figure 3.4. They are defined modulo permutation of arguments of \wedge and \vee . That is, we do not distinguish $A_1 \wedge \dots \wedge A_n$ from any formula $A_{i_1} \wedge \dots \wedge A_{i_n}$, where the sequence i_1, \dots, i_n is a permutation of the sequence $1, \dots, n$. For example, we treat $A_1 \vee A_2 \vee A_3$ and $A_3 \vee A_1 \vee A_2$ as the same formula.

The algorithm for evaluating formulas is given in Figure 3.5 on the next page.

EXAMPLE 3.12 (See Example 3.11 on page 36.) Consider again the formula $A = (p \rightarrow q) \wedge (p \wedge q \rightarrow r) \rightarrow (p \rightarrow r)$ and the interpretation $I = \{p \mapsto 1, q \mapsto 0, r \mapsto 1\}$. Let us evaluate A in I using the Formula Evaluation Algorithm. According to the algorithm, we first replace the true atoms by \top , and the false ones by \perp . We obtain the formula

$$(\top \rightarrow \perp) \wedge (\top \wedge \perp \rightarrow \top) \rightarrow (\top \rightarrow \top).$$

Then we apply the rewrite rules to obtain its normal form. One possible sequence of rewritings is as follows:

$$\begin{aligned} (\top \rightarrow \perp) \wedge (\top \wedge \perp \rightarrow \top) \rightarrow (\top \rightarrow \top) &\Rightarrow \\ (\top \rightarrow \perp) \wedge (\top \wedge \perp \rightarrow \top) \rightarrow \top &\Rightarrow \\ \top. & \end{aligned}$$

```

procedure evaluate(G, I)
input: formula G, interpretation I
output: a boolean value
begin
  forall atoms p occurring in G
    if  $I(p) = 1$ 
      then replace all occurrences of p in G by  $\top$ 
    else replace all occurrences of p in G by  $\perp$ 
  rewrite G into a normal form using the rewrite rules of Figure 3.4
  if  $G = \top$  then return 1 else return 0
end

```

Figure 3.5: Formula Evaluation Algorithm

Another possible sequence of rewritings is as follows:

$$\begin{aligned}
 (\top \rightarrow \perp) \wedge (\top \wedge \perp \rightarrow \top) &\rightarrow (\top \rightarrow \top) \Rightarrow \\
 \perp \wedge (\top \wedge \perp \rightarrow \top) &\rightarrow (\top \rightarrow \top) \Rightarrow \\
 \perp \rightarrow (\top \rightarrow \top) &\Rightarrow \\
 \top. &
 \end{aligned}$$

Note that both sequences yield the same answer (see Exercise 3.7). □

3.7 Propositional Formulas and Boolean Circuits

There exists a close resemblance between propositional formulas and boolean circuits.

Correspondence between circuits and formulas. Boolean functions. Can all boolean functions be represented by circuits/formulas? Traditional questions related to boolean circuits: given two boolean circuits, are they equivalent? Given a circuit, does there exist an equivalent circuit of a smaller size?

Exercises

Formalize some sentences.

EXERCISE 3.1 The following formula has its parentheses removed according to the priorities given in Figure 3.1 on page 30. Restore the parentheses.

$$\neg p_1 \rightarrow \neg \neg p_2 \leftrightarrow p_3 \wedge p_4.$$

□

EXERCISE 3.2 Find all positions in the formula of Exercise 3.1. □

EXERCISE 3.3 Which formulas have no immediate subformulas? □

EXERCISE 3.4 For each of the following formulas, write down all of their subformulas.

$$\begin{aligned} \neg p_1 \rightarrow \neg\neg p_2 \leftrightarrow p_3 \wedge p_4; \\ p \wedge r \wedge q \rightarrow p \vee q \vee r. \end{aligned} \quad \square$$

EXERCISE 3.5 Evaluate each of the formulas $p \leftrightarrow (q \leftrightarrow r)$ and $(p \leftrightarrow \neg\neg q) \leftrightarrow r \vee p \vee \neg q$ in the interpretation $\{p \mapsto 0, q \mapsto 1, r \mapsto 0\}$ using (i) the table-based method of Example 3.11; (ii) the rewriting-based algorithm of Figure 3.5. □

EXERCISE 3.6 Represent as a formula the boolean function f of three arguments p_1, p_2, p_3 with the following operation table:

p_1	p_2	p_3	$f(p_1, p_2, p_3)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

Can you find a formula with this property that contains only three occurrences of connectives? □

EXERCISE 3.7 Prove that the Formula Evaluation Algorithm of Figure 3.5 on the preceding page returns the same answer independently of the order of rewriting. In other words, prove that the rewrite rule system of Figure 3.4 is confluent. □

EXERCISE 3.8 A propositional formula $A(p_1, \dots, p_n)$ of atoms p_1, \dots, p_n is called a *parity check formula* if its models are exactly those that satisfy an even number of atoms among p_1, \dots, p_n . Find parity check formulas which contain exactly one occurrence of each atom. □

EXERCISE 3.9 Show that the formulas $p \rightarrow (q \rightarrow r)$ and $(p \rightarrow q) \rightarrow r$ are *not* equivalent by finding an interpretation in which they have different truth values. □