1

Propositional Logic

> *A deduction is speech in which, certain things having been supposed, something different from the things supposed results of necessity because of their being so.*
>
> — Aristotle
> *Prior Analytics*, 4th century BC

A calculus is a set of symbols and a system of rules for manipulating the symbols. In an interesting calculus, the symbols and rules have meaning in some domain that matters. For example, the differential calculus defines rules for manipulating the integral symbol over a polynomial to compute the area under the curve that the polynomial defines. Area has meaning outside of the calculus; the calculus provides the tool for computing such quantities. The domain of the differential calculus, loosely speaking, consists of real numbers and functions over those numbers.

Computer scientists are interested in a different domain and thus require a different calculus. The behavior of programs, or computation, is a computer scientist's chief concern. What is an appropriate domain for studying computation? The basic entity of the domain is *state*: roughly, the assignment of values (for example, Booleans, integers, or addresses) to variables. Pairs of states comprise *transitions*. A *computation* is a sequence of states, each adjacent pair of which is a transition. A program defines the form of its states, the set of transitions between states, and the set of computations that it can produce. A program's set of computations characterizes the program itself as precisely as its source code. Chapter 5 studies these ideas in depth.

With a domain in mind, a computer scientist can now ask questions. Does this program that accepts an array of integers produce a sorted array? In other words, does each of the program's computations have a state in which a sorted array is returned? Does this program ever access unallocated memory? Does this function always halt? To answer such questions, we need a calculus to reason about computations.

This chapter and the next introduce the calculus that will be the basis for studying computation in this book. In this chapter, we cover **propositional logic** (**PL**); in the next chapter, we build on the presentation to define **first-order logic** (**FOL**). PL and FOL are also known as **propositional calculus** and **predicate calculus**, respectively, because they are calculi for reasoning about propositions ("the sky is blue", "this comment references itself") and predicates ("x is blue", "y references z"), respectively. Propositions are either true or false, while predicates evaluate to true or false depending on the values given to their parameters (x, y, and z).

Just as differential calculus has a set of symbols, a set of rules, and a mapping to reality that provides its meaning, propositional logic has its own symbols, rules of inference, and meaning. Sections 1.1 and 1.2 introduce the *syntax* and *semantics* (meaning) of PL formulae. Then Section 1.3 discusses two concepts that are fundamental throughout this book, *satisfiability* (Is this formula ever true?) and *validity* (Is this formula always true?), and the rules for computing whether a PL formula is satisfiable or valid. Rules for manipulating PL formulae, some of which preserve satisfiability and validity, are discussed in Section 1.5 and applied in Section 1.6.

1.1 Syntax

In this section, we introduce the syntax of PL. The **syntax** of a logical language consists of a set of symbols and rules for combining them to form "sentences" (in this case, **formulae**) of the language.

The basic elements of PL are the **truth symbols** $\top$ ("true") and $\bot$ ("false") and the **propositional variables**, usually denoted by P, Q, R, P_1, P_2, ... A countably infinite set of propositional variable symbols exists. **Logical connectives**, also called **Boolean connectives**, provide the expressive power of PL. A **formula** is simply $\top$, $\bot$, or a propositional variable P; or the application of one of the following connectives to formulae F, F_1, or F_2:

- $\neg F$: negation, pronounced "not";
- $F_1 \wedge F_2$: conjunction, pronounced "and";
- $F_1 \vee F_2$: disjunction, pronounced "or";
- $F_1 \rightarrow F_2$: implication, pronounced "implies";
- $F_1 \leftrightarrow F_2$: iff, pronounced "if and only if".

Each connective has an **arity** (the number of arguments that it takes): negation is **unary** (it takes one argument), while the other connectives are **binary** (they take two arguments). The left and right arguments of $\rightarrow$ are called the **antecedent** and **consequent**, respectively.

Some common terminology is useful. An **atom** is a truth symbol $\top$, $\bot$ or propositional variable P, Q, ... A **literal** is an atom α or its negation $\neg\alpha$. A **formula** is a literal or the application of a logical connective to a formula or formulae.

Formula G is a **subformula** of formula F if it occurs syntactically within G. More precisely,

- the only subformula of P is P;
- the subformulae of $\neg F$ are $\neg F$ and the subformulae of F;
- and the subformulae of $F_1 \wedge F_2$, $F_1 \vee F_2$, $F_1 \to F_2$, $F_1 \leftrightarrow F_2$ are the formula itself and the subformulae of F_1 and F_2.

Notice that every formula is a subformula of itself. The **strict subformulae** of a formula are all its subformulae except itself.

Example 1.1. Consider the formula

$$F : \ (P \wedge Q) \ \to \ (P \vee \neg Q) \ .$$

It contains two propositional variables, P and Q. Each instance of P and Q is an atom and a literal. $\neg Q$ is a literal, but not an atom. F has six distinct subformulae:

$$F \ , \quad P \vee \neg Q \ , \quad \neg Q \ , \quad P \wedge Q \ , \quad P \ , \quad Q \ .$$

Its strict subformulae are all of its subformulae except F itself. ■

Parentheses are cumbersome. We define the relative precedence of the logical connectives from highest to lowest as follows: $\neg$, $\wedge$, $\vee$, $\to$, $\leftrightarrow$. Additionally, let $\to$ and $\leftrightarrow$ associate to the right, so that $P \to Q \to R$ is the same formula as $P \to (Q \to R)$.

Example 1.2. Abbreviate F of Example 1.1 as

$$F' : \ P \wedge Q \ \to \ P \vee \neg Q \ .$$

Also,

$$P_1 \wedge \neg P_2 \wedge \top \vee \neg P_1 \wedge P_2$$

stands for

$$(P_1 \wedge ((\neg P_2) \wedge \top)) \vee ((\neg P_1) \wedge P_2) \ .$$

Finally,

$$P_1 \ \to \ P_2 \ \to \ P_3$$

abbreviates

$$P_1 \ \to \ (P_2 \ \to \ P_3) \ .$$

■

1.2 Semantics

So far, we have considered the syntax of PL. The **semantics** of a logic provides its meaning. What exactly is meaning? In PL, meaning is given by the **truth values** true and false, where true $\neq$ false. Our objective is to define how to give meaning to formulae.

The first step in defining the semantics of PL is to provide a mechanism for evaluating the propositional variables. An **interpretation** I assigns to every propositional variable exactly one truth value. For example,

$$I: \ \{P \mapsto \mathsf{true}, \ Q \mapsto \mathsf{false}, \ \ldots\}$$

is an interpretation assigning true to P and false to Q, where $\ldots$ elides the (countably infinitely many) assignments that are not relevant to us. That is, I assigns to every propositional variable available to us (and there are countably infinitely many) a value. We usually do not write the elision. Clearly, many interpretations exist.

Now given a PL formula F and an interpretation I, the truth value of F can be computed. The simplest manner of computing the truth value of F is via a **truth table**. Let us first examine truth tables that indicate how to evaluate each logical connective in terms of its arguments. First, a propositional variable gets its truth value immediately from I. Now consider the possible evaluations of F: it is either true or false. How is $\neg F$ evaluated? The following table summarizes the possibilities, where 0 corresponds to the value false, and 1 corresponds to true:

F	$\neg F$
0	1
1	0

The other connective can be defined similarly given values of F_1 and F_2:

F_1	F_2	$F_1 \wedge F_2$	$F_1 \vee F_2$	$F_1 \rightarrow F_2$	$F_1 \leftrightarrow F_2$
0	0	0	0	1	1
0	1	0	1	1	0
1	0	0	1	0	0
1	1	1	1	1	1

In particular, $F_1 \rightarrow F_2$ is false iff F_1 is true and F_2 is false. (Throughout the book, we use the word "iff" to abbreviate the phrase "if and only if"; one can also read it as "precisely when".)

Example 1.3. Consider the formula

$$F: \ P \wedge Q \ \rightarrow \ P \vee \neg Q$$

and the interpretation

$I : \{P \mapsto \mathsf{true},\ Q \mapsto \mathsf{false}\}$.

To evaluate the truth value of F under I, construct the following table:

P	Q	$\neg Q$	$P \wedge Q$	$P \vee \neg Q$	F
1	0	1	0	1	1

The top row is given by the subformulae of F. I provides values for the first two columns; then the semantics of PL provide the values for the remainder of the table. Hence, F evaluates to true under I. ■

This tabular notation is convenient, but it is unsuitable for the predicate logic of Chapter 2. Instead, we introduce an **inductive definition** of PL's semantics that will extend to Chapter 2. An inductive definition defines the meaning of basic elements first, which in the case of PL are atoms. Then it assumes that the meaning of a set of elements is fixed and defines a more complex element in terms of these elements. For example, in PL, $F_1 \wedge F_2$ is a more complex formula than either of the formulae F_1 or F_2.

Recall that we want to compute whether F has value true under interpretation I. We write $I \models F$ if F evaluates to true under I and $I \not\models F$ if F evaluates to false. To start our inductive definition, define the meaning of truth symbols:

$$I \models \top$$
$$I \not\models \bot$$

Under any interpretation I, $\top$ has value true, and $\bot$ has value false. Next, define the truth value of propositional variables:

$$I \models P \qquad \text{iff } I[P] = \mathsf{true}$$

P has value true iff the interpretation I assigns P to have value true.

Since an interpretation assigns a truth value to every propositional variable, I assigns false to P when I does not assign true to P. Thus, we can instead define the truth values of propositional variables as follows:

$$I \not\models P \qquad \text{iff } I[P] = \mathsf{false}$$

Since $\mathsf{true} \neq \mathsf{false}$, both definitions yield the same (unique) truth values.

Having completed the base cases of our inductive definition, we turn to the inductive step. Assume that formulae F, F_1, and F_2 have truth values. From these formulae, evaluate the semantics of more complex formulae:

$$\begin{array}{lll}
I \models \neg F & & \text{iff } I \not\models F \\
I \models F_1 \wedge F_2 & & \text{iff } I \models F_1 \text{ and } I \models F_2 \\
I \models F_1 \vee F_2 & & \text{iff } I \models F_1 \text{ or } I \models F_2 \\
I \models F_1 \rightarrow F_2 & & \text{iff, if } I \models F_1 \text{ then } I \models F_2 \\
I \models F_1 \leftrightarrow F_2 & & \text{iff } I \models F_1 \text{ and } I \models F_2, \text{ or } I \not\models F_1 \text{ and } I \not\models F_2
\end{array}$$

In studying these definitions, it is useful to recall the earlier definitions given by the truth tables, which are free of English ambiguities.

For implication, consider also the equivalent formulation

$$I \not\models F_1 \rightarrow F_2 \qquad \text{iff } I \models F_1 \text{ and } I \not\models F_2$$

The formula $F_1 \rightarrow F_2$ has truth value true under I when either F_1 is false or F_2 is true. It is false only when F_1 is true and F_2 is false. Our inductive definition of the semantics of PL is complete.

Example 1.4. Consider the formula

$$F: \; P \wedge Q \rightarrow P \vee \neg Q$$

and the interpretation

$$I: \; \{P \mapsto \mathsf{true}, \; Q \mapsto \mathsf{false}\} \; .$$

Compute the truth value of F as follows:

1.	$I \models P$	since $I[P] = \mathsf{true}$
2.	$I \not\models Q$	since $I[Q] = \mathsf{false}$
3.	$I \models \neg Q$	by 2 and semantics of $\neg$
4.	$I \not\models P \wedge Q$	by 2 and semantics of $\wedge$
5.	$I \models P \vee \neg Q$	by 1 and semantics of $\vee$
6.	$I \models F$	by 4 and semantics of $\rightarrow$

We considered the distinct subformulae of F according to the **subformula ordering**: F_1 precedes F_2 if F_1 is a subformula of F_2. In that order, we computed the truth value of F from its simplest subformulae to its most complex subformula (F itself).

The final line of the calculation deserves some explanation. According to the semantics for implication,

$$I \models F_1 \rightarrow F_2 \qquad \text{iff, if } I \models F_1 \text{ then } I \models F_2$$

the implication $F_1 \rightarrow F_2$ has value true when $I \not\models F_1$. Thus, line 5 is unnecessary for establishing the truth value of F. ■

1.3 Satisfiability and Validity

We now consider a fundamental characterization of PL formulae.

A formula F is **satisfiable** iff there exists an interpretation I such that $I \models F$. A formula F is **valid** iff for all interpretations I, $I \models F$. Determining satisfiability and validity of formulae are important tasks in logic.

Satisfiability and validity are dual concepts, and switching from one to the other is easy. F is valid iff $\neg F$ is unsatisfiable. For suppose that F is valid;

then for any interpretation I, $I \models F$. By the semantics of negation, $I \not\models \neg F$, so $\neg F$ is unsatisfiable. Conversely, suppose that $\neg F$ is unsatisfiable. For any interpretation I, $I \not\models \neg F$, so that $I \models F$ by the semantics of negation. Thus, F is valid.

Because of this duality between satisfiability and validity, we are free to focus on either one or the other in the text, depending on which is more convenient for the discussion. The reader should realize that statements about one are also statements about the other.

In this section, we present several methods of determining validity and satisfiability of PL formulae.

1.3.1 Truth Tables

Our first approach to checking the validity of a PL formula is the **truth-table method**. We exhibit this method by example.

Example 1.5. Consider the formula

$$F: \ P \wedge Q \ \rightarrow \ P \vee \neg Q \ .$$

Is it valid? Construct a table in which the first row is a list of the subformulae of F ordered according to the subformula ordering. Fill columns of propositional variables with all possible combinations of truth values. Then apply the semantics of PL to fill the rest of the table:

P	Q	$P \wedge Q$	$\neg Q$	$P \vee \neg Q$	F
0	0	0	1	1	1
0	1	0	0	0	1
1	0	0	1	1	1
1	1	1	0	1	1

The final column, which represents the truth value of F under the possible interpretations, is filled entirely with true. F is valid. ■

Example 1.6. Consider the formula

$$F: \ P \vee Q \ \rightarrow \ P \wedge Q \ .$$

Construct the truth table:

P	Q	$P \vee Q$	$P \wedge Q$	F
0	0	0	0	1
0	1	1	0	0
1	0	1	0	0
1	1	1	1	1

Because the second and third rows show that F can be false, F is invalid. ■

1.3.2 Semantic Arguments

Our next approach to validity checking is the **semantic argument method**. While more complicated than the truth-table method, we introduce it and emphasize it throughout the remainder of the chapter because it is our only method of evaluating the satisfiability and validity of formulae in Chapter 2.

A proof based on the semantic method begins by assuming that the given formula F is invalid: hence, there is a **falsifying interpretation** I such that $I \not\models F$. The proof proceeds by applying the semantic definitions of the logical connectives in the form of **proof rules**. A proof rule has one or more **premises** (assumed facts) and one or more **deductions** (deduced facts). An application of a proof rule requires matching the premises to facts already existing in the semantic argument and then forming the deductions. The proof rules are the following:

- According to the semantics of negation, from $I \models \neg F$, deduce $I \not\models F$; and from $I \not\models \neg F$, deduce $I \models F$:

$$\frac{I \models \neg F}{I \not\models F} \qquad \frac{I \not\models \neg F}{I \models F}$$

- According to the semantics of conjunction, from $I \models F \wedge G$, deduce both $I \models F$ and $I \models G$; and from $I \not\models F \wedge G$, deduce $I \not\models F$ or $I \not\models G$. The latter deduction results in a fork in the proof; each case must be considered separately.

$$\frac{I \models F \wedge G}{\begin{array}{l} I \models F \\ I \models G \end{array}} \qquad \frac{I \not\models F \wedge G}{I \not\models F \mid I \not\models G}$$

- According to the semantics of disjunction, from $I \models F \vee G$, deduce $I \models F$ or $I \models G$; and from $I \not\models F \vee G$, deduce both $I \not\models F$ and $I \not\models G$. The former deduction requires a case analysis in the proof.

$$\frac{I \models F \vee G}{I \models F \mid I \models G} \qquad \frac{I \not\models F \vee G}{\begin{array}{l} I \not\models F \\ I \not\models G \end{array}}$$

- According to the semantics of implication, from $I \models F \rightarrow G$, deduce $I \not\models F$ or $I \models G$; and from $I \not\models F \rightarrow G$, deduce both $I \models F$ and $I \not\models G$. The former deduction requires a case analysis in the proof.

$$\frac{I \models F \rightarrow G}{I \not\models F \mid I \models G} \qquad \frac{I \not\models F \rightarrow G}{\begin{array}{l} I \models F \\ I \not\models G \end{array}}$$

- According to the semantics of iff, from $I \models F \leftrightarrow G$, deduce $I \models F \wedge G$ or $I \not\models F \vee G$; and from $I \not\models F \leftrightarrow G$, deduce $I \models F \wedge \neg G$ or $I \models \neg F \wedge G$. Both deductions require considering multiple cases.

$$\frac{I \models F \leftrightarrow G}{I \models F \wedge G \quad | \quad I \not\models F \vee G} \qquad \frac{I \not\models F \leftrightarrow G}{I \models F \wedge \neg G \quad | \quad I \models \neg F \wedge G}$$

- Finally, a contradiction occurs when following the above proof rules results in the claim that an interpretation I both satisfies a formula F and does not satisfy F.

$$\frac{I \models F \qquad I \not\models F}{I \models \bot}$$

Before explaining proofs in more detail, let us see several examples.

Example 1.7. To prove that the formula

$$F: \; P \wedge Q \;\rightarrow\; P \vee \neg Q$$

is valid, assume that it is invalid and derive a contradiction. Thus, assume that there is a falsifying interpretation I of F (such that $I \not\models F$). Then,

1.	$I \not\models P \wedge Q \rightarrow P \vee \neg Q$	assumption
2.	$I \models P \wedge Q$	by 1 and semantics of $\rightarrow$
3.	$I \not\models P \vee \neg Q$	by 1 and semantics of $\rightarrow$
4.	$I \models P$	by 2 and semantics of $\wedge$
5.	$I \models Q$	by 2 and semantics of $\wedge$
6.	$I \not\models P$	by 3 and semantics of $\vee$
7.	$I \not\models \neg Q$	by 3 and semantics of $\vee$
8.	$I \models Q$	by 7 and semantics of $\neg$

Lines 4 and 6 contradict each other, so that our assumption must be wrong: F is actually valid.

We can end the proof as soon as we have a contradiction. For example,

1.	$I \not\models P \wedge Q \rightarrow P \vee \neg Q$	assumption
2.	$I \models P \wedge Q$	by 1 and semantics of $\rightarrow$
3.	$I \not\models P \vee \neg Q$	by 1 and semantics of $\rightarrow$
4.	$I \models P$	by 2 and semantics of $\wedge$
5.	$I \not\models P$	by 3 and semantics of $\vee$

This argument is sufficient because a contradiction already exists. In other words, the discovered contradiction closes the one branch of the proof. We sometimes note the contradiction explicitly in the proof:

6.	$I \models \bot$	4 and 5 are contradictory

■

Example 1.8. To prove that the formula

$$F: \ (P \to Q) \wedge (Q \to R) \ \to \ (P \to R)$$

is valid, assume otherwise and derive a contradiction:

1.	$I \not\models F$	assumption
2.	$I \models (P \to Q) \ \wedge \ (Q \to R)$	by 1 and semantics of $\to$
3.	$I \not\models P \to R$	by 1 and semantics of $\to$
4.	$I \models P$	by 3 and semantics of $\to$
5.	$I \not\models R$	by 3 and semantics of $\to$
6.	$I \models P \to Q$	by 2 and semantics of $\wedge$
7.	$I \models Q \to R$	by 2 and semantics of $\wedge$

There are two cases to consider from 6. In the first case,

8*a*.	$I \not\models P$	by 6 and semantics of $\to$
9*a*.	$I \models \bot$	4 and 8*a* are contradictory

In the second case,

8*b*.	$I \models Q$	by 6 and semantics of $\to$

Now there are two more cases from 7. In the first case,

9*ba*.	$I \not\models Q$	by 7 and semantics of $\to$
10*ba*.	$I \models \bot$	8*b* and 9*ba* are contradictory

In the second case,

9*bb*.	$I \models R$	by 7 and semantics of $\to$
10*bb*.	$I \models \bot$	5 and 9*bb* are contradictory

All three branches of the proof are closed: F is valid. ■

We introduce vocabulary for discussing semantic proofs. The reader need not memorize these terms now; just refer to them as they are used. A **line** $L : I \models F$ or $L : I \not\models F$ is a single statement in the proof, sometimes labeled as in the examples. A line L is a **direct descendant** of a **parent** M if L is directly below M in the proof. L is a **descendant** of M if M is L itself, if L is a direct descendant of M, or if the parent of L is a descendant of M (in other words, *descendant* is the reflexive and transitive closure of *direct descendant*). M is an **ancestor** of L if L is a descendant of M. Several proof rules — the second conjunction rule, the first disjunction rule, the first implication rule, and both rules for iff — produce a fork in the argument, as the last example shows. A proof thus evolves as a tree rather than linearly. A **branch** of the tree is a sequence of lines descending from the root. A branch is **closed** if it contains a contradiction, either explicitly as $I \models \bot$ or implicitly as $I \models G$

and $I \not\models G$ for some formula G. Otherwise, the branch is **open**. A semantic argument is **finished** when no more proof rules are applicable. It is a proof of the validity of F if every branch is closed; otherwise, each open branch describes a falsifying interpretation of F.

While the given proof rules are (theoretically) sufficient, **derived** proof rules can make proofs more concise.

Example 1.9. The derived rule of **modus ponens** simplifies the proof of Example 1.8. The rule is the following:

$$\frac{I \models F \qquad I \models F \to G}{I \models G}$$

In words, from $I \models F$ and $I \models F \to G$, deduce $I \models G$.

Using this rule, let us simplify the proof of the validity of

$$F: \ (P \to Q) \wedge (Q \to R) \ \to \ (P \to R) \ .$$

We assume that it is invalid and try to derive a contradiction.

1.	$I \not\models F$	assumption
2.	$I \models (P \to Q) \ \wedge \ (Q \to R)$	by 1 and semantics of $\to$
3.	$I \not\models P \to R$	by 1 and semantics of $\to$
4.	$I \models P$	by 3 and semantics of $\to$
5.	$I \not\models R$	by 3 and semantics of $\to$
6.	$I \models P \to Q$	by 2 and semantics of $\wedge$
7.	$I \models Q \to R$	by 2 and semantics of $\wedge$
8.	$I \models Q$	by 4, 6, and *modus ponens*
9.	$I \models R$	by 8, 7, and *modus ponens*
10.	$I \models \bot$	5 and 9 are contradictory

This proof has only one branch. ■

The truth-table and semantic methods can be used to check satisfiability. For example, the truth table of Example 1.6 can be extended to show that

$$\neg F: \ \neg(P \vee Q \ \to \ P \wedge Q)$$

is satisfiable:

P	Q	$P \vee Q$	$P \wedge Q$	F	$\neg F$
0	0	0	0	1	0
0	1	1	0	0	1
1	0	1	0	0	1
1	1	1	1	1	0

The second and third rows represent satisfying interpretations of $\neg F$. Additionally, the semantic argument in the following example shows that

$$G: \neg(P \vee Q \rightarrow P \wedge Q)$$

is satisfied by the discovered interpretation I, and thus that G is satisfiable.

Example 1.10. To prove that the formula

$$F: P \vee Q \rightarrow P \wedge Q$$

is valid, assume that F is invalid; then there is an interpretation I such that $I \models \neg F$:

1.	$I \not\models P \vee Q \rightarrow P \wedge Q$	assumption
2.	$I \models P \vee Q$	by 1 and semantics of $\rightarrow$
3.	$I \not\models P \wedge Q$	by 1 and semantics of $\rightarrow$

We have two choices to make. By 2 and the semantics of disjunction, either P or Q must be true. By 3 and the semantics of conjunction, either P or Q must be false. So there are two options: either P is true and Q is false, or P is false and Q is true. We choose P to be true and Q to be false. Then,

4a.	$I \models P$	by 2 and semantics of $\vee$
5a.	$I \not\models Q$	by 3 and semantics of $\wedge$

The only subformulae of P and Q are themselves, so the table is complete. Yet we did not derive a contradiction. In fact, we found the interpretation

$$I: \{P \mapsto \mathsf{true},\ Q \mapsto \mathsf{false}\}$$

for which $I \models \neg F$. Therefore, F is actually invalid. The interpretation $I : \{P \mapsto \mathsf{true},\ Q \mapsto \mathsf{false}\}$ is a falsifying interpretation.

If our choice had resulted in a contradiction, then we would have had to try the other choice for P and Q, in which P is false and Q is true. In general, we stop either when we have found an interpretation or when we have closed every branch. ■

1.4 Equivalence and Implication

Just as satisfiability and validity are important properties of PL formulae, **equivalence** and **implication** are important properties of pairs of formulae. Two formulae F_1 and F_2 are **equivalent** if they evaluate to the same truth value under all interpretations I. That is, for all interpretations I, $I \models F_1$ iff $I \models F_2$. Another way to state the equivalence of F_1 and F_2 is to assert the validity of the formula $F_1 \leftrightarrow F_2$. We write $F_1 \Leftrightarrow F_2$ when F_1 and F_2 are equivalent. $F_1 \Leftrightarrow F_2$ is *not* a formula; it simply abbreviates the statement "F_1 and F_2 are equivalent."

We use the last characterization to prove that two formulae are equivalent.

Example 1.11. To prove that

$$P \;\Leftrightarrow\; \neg\neg P \;,$$

we prove that

$$P \;\leftrightarrow\; \neg\neg P$$

is valid via a truth table:

P	$\neg P$	$\neg\neg P$	$P \;\leftrightarrow\; \neg\neg P$
0	1	0	1
1	0	1	1

■

Example 1.12. To prove

$$P \rightarrow Q \;\Leftrightarrow\; \neg P \vee Q \;,$$

we prove that

$$F: \; P \rightarrow Q \;\leftrightarrow\; \neg P \vee Q$$

is valid via a truth table:

P	Q	$P \rightarrow Q$	$\neg P$	$\neg P \vee Q$	F
0	0	1	1	1	1
0	1	1	1	1	1
1	0	0	0	0	1
1	1	1	0	1	1

■

Formula F_1 **implies** formula F_2 if $I \models F_2$ for every interpretation I such that $I \models F_1$. Another way to state that F_1 implies F_2 is to assert the validity of the formula $F_1 \rightarrow F_2$. We write $F_1 \Rightarrow F_2$ when F_1 implies F_2. Do not confuse the *implication* $F_1 \Rightarrow F_2$, which asserts the validity of $F_1 \rightarrow F_2$, with the *PL formula* $F_1 \rightarrow F_2$, which is constructed using the logical operator $\rightarrow$. $F_1 \Rightarrow F_2$ is *not* a formula.

As with equivalences, we use the validity characterization to prove implications.

Example 1.13. To prove that

$$R \wedge (\neg R \vee P) \;\Rightarrow\; P \;,$$

we prove that

$$F:\ R \wedge (\neg R \vee P)\ \rightarrow\ P$$

is valid via a semantic argument. Suppose F is not valid; then there exists an interpretation I such that $I\ \not\models\ F$:

1.	$I\ \not\models\ F$	assumption
2.	$I\ \models\ R \wedge (\neg R \vee P)$	by 1 and semantics of $\rightarrow$
3.	$I\ \not\models\ P$	by 1 and semantics of $\rightarrow$
4.	$I\ \models\ R$	by 2 and semantics of $\wedge$
5.	$I\ \models\ \neg R \vee P$	by 2 and semantics of $\wedge$

There are two cases to consider. In the first case,

6*a*.	$I\ \models\ \neg R$	by 5 and semantics of $\vee$
7*a*.	$I\ \models\ \bot$	4 and 6*a* are contradictory

In the second case,

6*b*.	$I\ \models\ P$	by 5 and semantics of $\vee$
7*b*.	$I\ \models\ \bot$	3 and 6*b* are contradictory

Thus, our assumption that $I\ \not\models\ F$ is wrong, and F is valid. ■

1.5 Substitution

Substitution is a syntactic operation on formulae with significant semantic consequences. It allows us to prove the validity of entire sets of formulae via **formula templates**. It is also an essential tool for manipulating formulae throughout the text.

A **substitution** σ is a mapping from formulae to formulae:

$$\sigma:\ \{F_1 \mapsto G_1,\ \ldots,\ F_n \mapsto G_n\}\ .$$

The **domain** of σ, $\mathsf{domain}(\sigma)$, is

$$\mathsf{domain}(\sigma):\ \{F_1, \ldots, F_n\}\ ,$$

while the **range** $\mathsf{range}(\sigma)$ is

$$\mathsf{range}(\sigma):\ \{G_1, \ldots, G_n\}\ .$$

The application of a substitution σ to a formula F, $F\sigma$, replaces each occurrence of a formula F_i in the domain of σ with its corresponding formula G_i in the range of σ. Replacements occur all at once. We remove any ambiguity by establishing that when both subformulae F_j and F_k are in the domain of σ, and F_k is a strict subformula of F_j, then the larger subformula F_j is replaced by the corresponding formula G_j. An example clarifies this statement.

Example 1.14. Consider formula

$$F:\ P \wedge Q\ \rightarrow\ P \vee \neg Q$$

and substitution

$$\sigma:\ \{P\ \mapsto\ R,\quad P \wedge Q\ \mapsto\ P \rightarrow Q\}\ .$$

Then

$$F\sigma:\ (P \rightarrow Q)\ \rightarrow\ R \vee \neg Q\ ,$$

where the antecedent $P \wedge Q$ of F is replaced by $P \rightarrow Q$, and the P of the consequent is replaced by R. Moreover,

$$F\sigma\ \neq\ R \wedge Q\ \rightarrow\ R \vee \neg Q$$

by our convention. ■

A **variable substitution** is a substitution in which the domain consists only of propositional variables.

One notation is useful when working with substitutions. When we write $F[F_1, \ldots, F_n]$, we mean that formula F can have formulae F_i, $i = 1, \ldots, n$, as subformulae. If σ is $\{F_1 \mapsto G_1, \ldots, F_n \mapsto G_n\}$, then

$$F[F_1, \ldots, F_n]\sigma:\ F[G_1, \ldots, G_n]\ .$$

In the formula of Example 1.14, writing

$$F[P,\ P \wedge Q]\sigma:\ F[R,\ P \rightarrow Q]$$

emphasizes that subformulae P and $P \wedge Q$ of F are replaced by formulae R and $P \rightarrow Q$, respectively.

Two interesting semantic consequences can be derived from substitution. Proposition 1.15 states that substituting subformulae F_i of F with corresponding equivalent subformulae G_i results in an equivalent formula F'.

Proposition 1.15 (Substitution of Equivalent Formulae). *Consider substitution*

$$\sigma:\ \{F_1 \mapsto G_1, \ldots, F_n \mapsto G_n\}$$

such that for each i, $F_i \Leftrightarrow G_i$. Then $F \Leftrightarrow F\sigma$.

Example 1.16. Consider applying substitution

$$\sigma:\ \{P \rightarrow Q\ \mapsto\ \neg P \vee Q\}$$

to

$$F : (P \to Q) \to R .$$

Since $P \to Q \Leftrightarrow \neg P \vee Q$, the formula

$$F\sigma : (\neg P \vee Q) \to R$$

is equivalent to F. ■

Proposition 1.17 asserts that proving the validity of a PL formula F actually proves the validity of an infinite set of formulae: those formulae that can be derived from F via variable substitutions.

Proposition 1.17 (Valid Template). *If F is valid and $G = F\sigma$ for some variable substitution σ, then G is valid.*

Example 1.18. In Example 1.12, we proved that $P \to Q$ is equivalent to $\neg P \vee Q$:

$$F : (P \to Q) \leftrightarrow (\neg P \vee Q)$$

is valid. The validity of F implies that every formula of the form $F_1 \to F_2$ is equivalent to $\neg F_1 \vee F_2$, for arbitrary subformulae F_1 and F_2. ■

Finally, it is often useful to compute the **composition** of substitutions. Given substitutions σ_1 and σ_2, the idea is to compute substitution σ such that $F\sigma_1\sigma_2 = F\sigma$ for any F. Compute $\sigma_1\sigma_2$ as follows:

1. apply σ_2 to each formula of the range of σ_1, and add the results to σ;
2. if F_i of $F_i \mapsto G_i$ appears in the domain of σ_2 but *not* in the domain of σ_1, then add $F_i \mapsto G_i$ to σ.

Example 1.19. Compute the composition of substitutions

$$\sigma_1\sigma_2 : \{P \mapsto R,\ P \wedge Q \mapsto P \to Q\}\{P \mapsto S,\ S \mapsto Q\}$$

as follows:

$$\begin{aligned}&\{P \mapsto R\sigma_2,\ P \wedge Q \mapsto (P \to Q)\sigma_2,\ S \mapsto Q\}\\ &\quad = \{P \mapsto R,\ P \wedge Q \mapsto S \to Q,\ S \mapsto Q\}\end{aligned}$$

■

1.6 Normal Forms

A **normal form** of formulae is a syntactic restriction such that for every formula of the logic, there is an equivalent formula in the normal form. Three normal forms are particularly important for PL.

Negation normal form (**NNF**) requires that $\neg$, $\wedge$, and $\vee$ be the only connectives and that negations appear only in literals. Transforming a formula F to equivalent formula F' in NNF can be computed recursively using the following list of template equivalences:

$$\begin{aligned}
\neg\neg F_1 &\Leftrightarrow F_1 \\
\neg\top &\Leftrightarrow \bot \\
\neg\bot &\Leftrightarrow \top \\
\neg(F_1 \wedge F_2) &\Leftrightarrow \neg F_1 \vee \neg F_2 \\
\neg(F_1 \vee F_2) &\Leftrightarrow \neg F_1 \wedge \neg F_2 \\
F_1 \rightarrow F_2 &\Leftrightarrow \neg F_1 \vee F_2 \\
F_1 \leftrightarrow F_2 &\Leftrightarrow (F_1 \rightarrow F_2) \wedge (F_2 \rightarrow F_1)
\end{aligned}$$

When implementing the transformation, the equivalences should be applied left-to-right. The equivalences

$$\neg(F_1 \wedge F_2) \Leftrightarrow \neg F_1 \vee \neg F_2 \qquad \neg(F_1 \vee F_2) \Leftrightarrow \neg F_1 \wedge \neg F_2$$

are known as **De Morgan's Law**.

Propositions 1.15 and 1.17 justify that the result of applying the template equivalences to a formula produces an equivalent formula. The transitivity of equivalence justifies that this equivalence holds over any number of transformations: if $F \Leftrightarrow G$ and $G \Leftrightarrow H$, then $F \Leftrightarrow H$.

Example 1.20. To convert the formula

$$F: \neg(P \rightarrow \neg(P \wedge Q))$$

into NNF, apply the template equivalence

$$F_1 \rightarrow F_2 \Leftrightarrow \neg F_1 \vee F_2 \tag{1.1}$$

to produce

$$F': \neg(\neg P \vee \neg(P \wedge Q)) .$$

Let us understand this "application" of the template equivalence in detail. First, apply variable substitution

$$\sigma_1: \{F_1 \mapsto P,\ F_2 \mapsto \neg(P \wedge Q)\}$$

to the valid template formula of equivalence (1.1):

$$(F_1 \rightarrow F_2 \leftrightarrow \neg F_1 \vee F_2)\sigma_1: P \rightarrow \neg(P \wedge Q) \leftrightarrow \neg P \vee \neg(P \wedge Q) .$$

Proposition 1.17 implies that the result is valid. Then construct substitution

$$\sigma_2: \{P \rightarrow \neg(P \wedge Q) \mapsto \neg P \vee \neg(P \wedge Q)\} ,$$

and apply Proposition 1.15 to $F\sigma_2$ to yield that

$$F' : \ \neg(\neg P \vee \neg(P \wedge Q))$$

is equivalent to F. Subsequently, we shall not provide these details.

Continuing with the conversion to NNF, apply De Morgan's law

$$\neg(F_1 \vee F_2) \ \Leftrightarrow \ \neg F_1 \wedge \neg F_2$$

to produce

$$F'' : \ \neg\neg P \wedge \neg\neg(P \wedge Q) \ .$$

Apply

$$\neg\neg F_1 \ \Leftrightarrow \ F_1$$

twice to produce

$$F''' : \ P \wedge P \wedge Q \ ,$$

which is in NNF and equivalent to F. ■

A formula is in **disjunctive normal form** (**DNF**) if it is a disjunction of conjunctions of literals:

$$\bigvee_i \bigwedge_j \ell_{i,j} \quad \text{for literals } \ell_{i,j} \ .$$

To convert a formula F into an equivalent formula in DNF, transform F into NNF and then use the following table of template equivalences:

$$\begin{array}{l} (F_1 \vee F_2) \wedge F_3 \ \Leftrightarrow \ (F_1 \wedge F_3) \vee (F_2 \wedge F_3) \\ F_1 \wedge (F_2 \vee F_3) \ \Leftrightarrow \ (F_1 \wedge F_2) \vee (F_1 \wedge F_3) \end{array}$$

Again, when implementing the transformation, the equivalences should be applied left-to-right. The equivalences simply say that conjunction distributes over disjunction.

Example 1.21. To convert

$$F : \ (Q_1 \vee \neg\neg Q_2) \ \wedge \ (\neg R_1 \to R_2)$$

into DNF, first transform it into NNF

$$F' : \ (Q_1 \vee Q_2) \ \wedge \ (R_1 \vee R_2) \ ,$$

and then apply distributivity to obtain

$$F'' : \ (Q_1 \wedge (R_1 \vee R_2)) \ \vee \ (Q_2 \wedge (R_1 \vee R_2)) \ ,$$

and then distributivity twice again to produce

$$F''' : \ (Q_1 \wedge R_1) \ \vee \ (Q_1 \wedge R_2) \ \vee \ (Q_2 \wedge R_1) \ \vee \ (Q_2 \wedge R_2) \ .$$

F''' is in DNF and is equivalent to F. ■

The dual of DNF is **conjunctive normal form** (**CNF**). A formula in CNF is a conjunction of disjunctions of literals:

$$\bigwedge_i \bigvee_j \ell_{i,j} \quad \text{for literals } \ell_{i,j} \ .$$

Each inner block of disjunctions is called a **clause**. To convert a formula F into an equivalent formula in CNF, transform F into NNF and then use the following table of template equivalences:

$$\begin{array}{l} (F_1 \wedge F_2) \vee F_3 \ \Leftrightarrow \ (F_1 \vee F_3) \wedge (F_2 \vee F_3) \\ F_1 \vee (F_2 \wedge F_3) \ \Leftrightarrow \ (F_1 \vee F_2) \wedge (F_1 \vee F_3) \end{array}$$

Example 1.22. To convert

$$F: \ (Q_1 \wedge \neg\neg Q_2) \ \vee \ (\neg R_1 \rightarrow R_2)$$

into CNF, first transform F into NNF:

$$F': \ (Q_1 \wedge Q_2) \ \vee \ (R_1 \vee R_2) \ .$$

Then apply distributivity to obtain

$$F'': \ (Q_1 \vee R_1 \vee R_2) \ \wedge \ (Q_2 \vee R_1 \vee R_2) \ ,$$

which is in CNF and equivalent to F. ■

1.7 Decision Procedures for Satisfiability

Section 1.3 introduced the truth-table and semantic argument methods for determining the satisfiability of PL formulae. In this section, we study algorithms for *deciding* satisfiability (see Section 2.6 for a formal discussion of decidability). A **decision procedure** for satisfiability of PL formulae reports, after some finite amount of computation, whether a given PL formula F is satisfiable.

1.7.1 Simple Decision Procedures

The truth-table method immediately suggests a decision procedure: construct the full table, which has 2^n rows when F has n variables, and report whether the final column, representing F, has value 1 in any row.

The semantic argument method also suggests a decision procedure. The basic idea is to make sure that a proof rule is only applied to each line in the argument at most once. Because each deduction is simpler in construction than its premise, the constructed proof is of finite size (see Chapter 4 for

a formal approach to proving this point). When the semantic argument is finished, report whether any branch is still open.

This simple description leaves out many details. Most importantly, when many lines exist to which one can apply proof rules, which line should be considered next? Different implementations of this decision, called **proof tactics**, result in different proof shapes and sizes. For example, one basic tactic is to apply proof rules with only one deduction before proof rules with multiple deductions to delay forks in the proof as long as possible.

Subsequent sections consider more sophisticated procedures that are the basis for modern satisfiability solvers.

1.7.2 Reconsidering the Truth-Table Method

In the naive decision procedure based on the truth-table method, the entire table is constructed. Actually, only one row need be considered at a time, making for a space efficient procedure. This idea is implemented in the following recursive algorithm for deciding the satisfiability of a PL formula F:

```
let rec SAT F =
  if F = ⊤ then true
  else if F = ⊥ then false
  else
    let P = CHOOSE vars(F) in
    (SAT F{P ↦ ⊤}) ∨ (SAT F{P ↦ ⊥})
```

The notation "`let rec` SAT F =" declares SAT as a recursive function that takes one argument, a formula F. The notation "`let` P = CHOOSE vars(F) `in`" means that P's value in the subsequent text is the variable returned by the CHOOSE function. When applying the substitutions $F\{P \mapsto \top\}$ or $F\{P \mapsto \bot\}$, the template equivalences of Exercise 1.2 should be applied to simplify the result. Then the comparisons $F = \top$ and $F = \bot$ can be implemented as purely syntactic operations.

At each recursive step, if F is not yet $\top$ or $\bot$, a variable is chosen on which to branch. Each possibility for P is attempted if necessary. This algorithm returns `true` immediately upon finding a satisfying interpretation. Otherwise, if F is unsatisfiable, it eventually returns $\bot$. SAT may save branching on certain variables by simplifying intermediate formulae.

Example 1.23. Consider the formula

$$F : \ (P \to Q) \wedge P \wedge \neg Q \ .$$

To compute SAT F, choose a variable, say P, and recurse on the first case,

$$F\{P \mapsto \top\} : \ (\top \ \to \ Q) \wedge \top \wedge \neg Q \ ,$$

which simplifies to

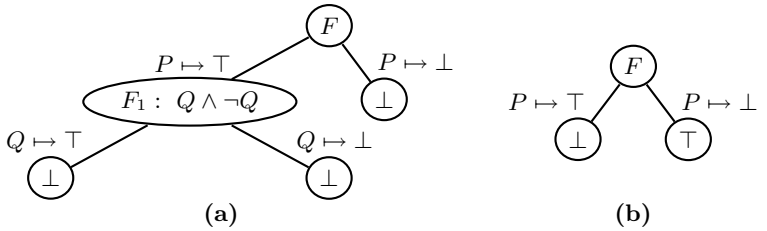

Fig. 1.1. Visualizing runs of SAT

$$F_1 : \ Q \wedge \neg Q \ .$$

Now try each of

$$F_1\{Q \mapsto \top\} \quad \text{and} \quad F_1\{Q \mapsto \bot\} \ .$$

Both simplify to $\bot$, so this branch ends without finding a satisfying interpretation.

Now try the other branch for P in F:

$$F\{P \mapsto \bot\} : \ (\bot \ \rightarrow \ Q) \wedge \bot \wedge \neg Q \ ,$$

which simplifies to $\bot$. Thus, this branch also ends without finding a satisfying interpretation. Thus, F is unsatisfiable.

The run of SAT on F is visualized in Figure 1.1**(a)**. ■

Example 1.24. Consider the formula

$$F : \ (P \rightarrow Q) \wedge \neg P \ .$$

To compute SAT F, choose a variable, say P, and recurse on the first case,

$$F\{P \mapsto \top\} : \ (\top \ \rightarrow \ Q) \wedge \neg \top \ ,$$

which simplifies to $\bot$. Therefore, try

$$F\{P \mapsto \bot\} : \ (\bot \ \rightarrow \ Q) \wedge \neg \bot$$

instead, which simplifies to $\top$. Arbitrarily assigning a value to Q produces the following satisfying interpretation:

$$I : \ \{P \mapsto \mathsf{false}, \ Q \mapsto \mathsf{true}\} \ .$$

The run of SAT on F is visualized in Figure 1.1**(b)**. ■

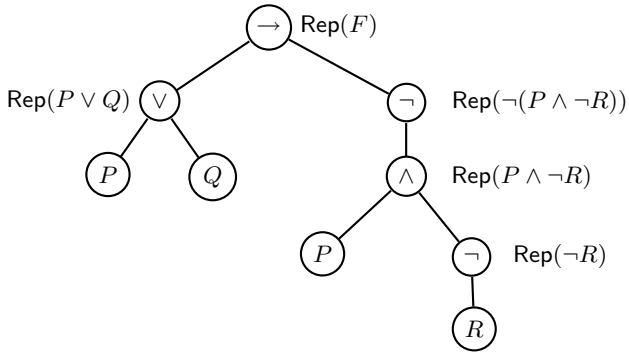

Fig. 1.2. Parse tree of $F: \ P \vee Q \ \rightarrow \ \neg(P \wedge \neg R)$ with representatives for subformulae

1.7.3 Conversion to an Equisatisfiable Formula in CNF

The next two decision procedures operate on PL formulae in CNF. The transformation suggested in Section 1.6 produces an equivalent formula that can be exponentially larger than the original formula: consider converting a formula in DNF into CNF. However, to decide the satisfiability of F, we need only examine a formula F' such that F and F' are **equisatisfiable**. F and F' are equisatisfiable when F is satisfiable iff F' is satisfiable.

We define a method for converting PL formula F to equisatisfiable PL formula F' in CNF that is at most a constant factor larger than F. The main idea is to introduce new propositional variables to represent the subformulae of F. The constructed formula F' includes extra clauses that assert that these new variables are equivalent to the subformulae that they represent.

Figure 1.2 visualizes the idea of the procedure. Each node of the "parse tree" of F represents a subformula G of F. With each node G is associated a representative propositional variable $\mathsf{Rep}(G)$. In the constructed formula F', each representative $\mathsf{Rep}(G)$ is asserted to be equivalent to the subformula G that it represents in such a way that the conjunction of all such assertions is in CNF. Finally, the representative $\mathsf{Rep}(F)$ of F is asserted to be true.

To obtain a small formula in CNF, each assertion of equivalence between $\mathsf{Rep}(G)$ and G refers at most to the children of G in the parse tree. How is this possible when a subformula may be arbitrarily large? The main trick is to refer to the representatives of G's children rather than the children themselves.

Let the "representative" function $\mathsf{Rep} : \mathsf{PL} \rightarrow \mathcal{V} \cup \{\top, \bot\}$ map PL formulae to propositional variables $\mathcal{V}$, $\top$, or $\bot$. In the general case, it is intended to map a formula F to its representative propositional variable P_F such that the truth value of P_F is the same as that of F. In other words, P_F provides a compact way of referring to F.

Let the "encoding" function $\mathsf{En} : \mathsf{PL} \rightarrow \mathsf{PL}$ map PL formulae to PL formulae. En is intended to map a PL formula F to a PL formula F' in CNF that asserts that F's representative, P_F, is equivalent to F: "$\mathsf{Rep}(F) \leftrightarrow F$".

As the base cases for defining Rep and En, define their behavior on $\top$, $\bot$, and propositional variables P:

$$\begin{array}{ll} \mathsf{Rep}(\top) = \top & \mathsf{En}(\top) = \top \\ \mathsf{Rep}(\bot) = \bot & \mathsf{En}(\bot) = \top \\ \mathsf{Rep}(P) = P & \mathsf{En}(P) = \top \end{array}$$

The representative of $\top$ is $\top$ itself, and the representative of $\bot$ is $\bot$ itself. Thus, $\mathsf{Rep}(\top) \leftrightarrow \top$ and $\mathsf{Rep}(\bot) \leftrightarrow \bot$ are both trivially valid, so $\mathsf{En}(\top)$ and $\mathsf{En}(\bot)$ are both $\top$. Finally, the representative of a propositional variable P is P itself; and again, $\mathsf{Rep}(P) \leftrightarrow P$ is trivially valid so that $\mathsf{En}(P)$ is $\top$.

For the inductive case, F is a formula other than an atom, so define its representative as a unique propositional variable P_F:

$$\mathsf{Rep}(F) = P_F \ .$$

En then asserts the equivalence of F and P_F as a CNF formula. On conjunction, define

$$\begin{array}{l} \mathsf{En}(F_1 \wedge F_2) = \\ \quad \texttt{let}\ P = \mathsf{Rep}(F_1 \wedge F_2)\ \texttt{in} \\ \quad (\neg P \vee \mathsf{Rep}(F_1)) \ \wedge \ (\neg P \vee \mathsf{Rep}(F_2)) \ \wedge \ (\neg \mathsf{Rep}(F_1) \vee \neg \mathsf{Rep}(F_2) \vee P) \end{array}$$

The returned formula

$$(\neg P \vee \mathsf{Rep}(F_1)) \ \wedge \ (\neg P \vee \mathsf{Rep}(F_2)) \ \wedge \ (\neg \mathsf{Rep}(F_1) \vee \neg \mathsf{Rep}(F_2) \vee P)$$

is in CNF and is equivalent to

$$\mathsf{Rep}(F_1 \wedge F_2) \ \leftrightarrow \ \mathsf{Rep}(F_1) \wedge \mathsf{Rep}(F_2) \ .$$

In detail, the first two clauses

$$(\neg P \vee \mathsf{Rep}(F_1)) \ \wedge \ (\neg P \vee \mathsf{Rep}(F_2))$$

together assert

$$P \ \rightarrow \ \mathsf{Rep}(F_1) \wedge \mathsf{Rep}(F_2)$$

(since, for example, $\neg P \vee \mathsf{Rep}(F_1)$ is equivalent to $P \rightarrow \mathsf{Rep}(F_1)$), while the final clause asserts

$$\mathsf{Rep}(F_1) \wedge \mathsf{Rep}(F_2) \ \rightarrow \ P \ .$$

Notice the application of Rep to F_1 and F_2. As mentioned above, it is the trick to producing a small CNF formula.

On negation, $\mathsf{En}(\neg F)$ returns a formula equivalent to $\mathsf{Rep}(\neg F) \ \leftrightarrow \ \neg \mathsf{Rep}(F)$:

$$\begin{array}{l} \mathsf{En}(\neg F) = \\ \quad \texttt{let}\ P = \mathsf{Rep}(\neg F)\ \texttt{in} \\ \quad (\neg P \vee \neg \mathsf{Rep}(F)) \ \wedge \ (P \vee \mathsf{Rep}(F)) \end{array}$$

En is defined for $\vee$, $\rightarrow$, and $\leftrightarrow$ as well:

$$\begin{aligned}
&\mathsf{En}(F_1 \vee F_2) = \\
&\quad \texttt{let } P = \mathsf{Rep}(F_1 \vee F_2) \texttt{ in} \\
&\quad (\neg P \vee \mathsf{Rep}(F_1) \vee \mathsf{Rep}(F_2)) \ \wedge \ (\neg \mathsf{Rep}(F_1) \vee P) \ \wedge \ (\neg \mathsf{Rep}(F_2) \vee P)
\end{aligned}$$

$$\begin{aligned}
&\mathsf{En}(F_1 \rightarrow F_2) = \\
&\quad \texttt{let } P = \mathsf{Rep}(F_1 \rightarrow F_2) \texttt{ in} \\
&\quad (\neg P \vee \neg \mathsf{Rep}(F_1) \vee \mathsf{Rep}(F_2)) \ \wedge \ (\mathsf{Rep}(F_1) \vee P) \ \wedge \ (\neg \mathsf{Rep}(F_2) \vee P)
\end{aligned}$$

$$\begin{aligned}
&\mathsf{En}(F_1 \leftrightarrow F_2) = \\
&\quad \texttt{let } P = \mathsf{Rep}(F_1 \leftrightarrow F_2) \texttt{ in} \\
&\quad (\neg P \vee \neg \mathsf{Rep}(F_1) \vee \mathsf{Rep}(F_2)) \ \wedge \ (\neg P \vee \mathsf{Rep}(F_1) \vee \neg \mathsf{Rep}(F_2)) \\
&\qquad \wedge \ (P \vee \neg \mathsf{Rep}(F_1) \vee \neg \mathsf{Rep}(F_2)) \ \wedge \ (P \vee \mathsf{Rep}(F_1) \vee \mathsf{Rep}(F_2))
\end{aligned}$$

Having defined En, let us construct the full CNF formula that is equisatisfiable to F. If S_F is the set of all subformulae of F (including F itself), then

$$F' : \ \mathsf{Rep}(F) \ \wedge \bigwedge_{G \in S_F} \mathsf{En}(G)$$

is in CNF and is equisatisfiable to F. The second main conjunct asserts the equivalences between all subformulae of F and their corresponding representatives. $\mathsf{Rep}(F)$ asserts that F's representative, and thus F itself (according to the second conjunct), is true.

If F has size n, where each instance of a logical connective or a propositional variable contributes one unit of size, then F' has size at most $30n + 2$. The size of F' is thus linear in the size of F. The number of symbols in the formula returned by $\mathsf{En}(F_1 \leftrightarrow F_2)$, which incurs the largest expansion, is 29. Up to one additional conjunction is also required per symbol of F. Finally, two extra symbols are required for asserting that $\mathsf{Rep}(F)$ is true.

Example 1.25. Consider formula

$$F : \ (Q_1 \wedge Q_2) \ \vee \ (R_1 \wedge R_2) \ ,$$

which is in DNF. To convert it to CNF, we collect its subformulae

$$S_F : \ \{Q_1, \ Q_2, \ Q_1 \wedge Q_2, \ R_1, \ R_2, \ R_1 \wedge R_2, \ F\}$$

and compute

$$\begin{aligned}
\mathsf{En}(Q_1) &= \top \\
\mathsf{En}(Q_2) &= \top \\
\mathsf{En}(Q_1 \wedge Q_2) &= (\neg P_{(Q_1 \wedge Q_2)} \vee Q_1) \ \wedge \ (\neg P_{(Q_1 \wedge Q_2)} \vee Q_2) \\
&\qquad \wedge \ (\neg Q_1 \vee \neg Q_2 \vee P_{(Q_1 \wedge Q_2)})
\end{aligned}$$

$$
\begin{aligned}
\mathsf{En}(R_1) &= \top \\
\mathsf{En}(R_2) &= \top \\
\mathsf{En}(R_1 \wedge R_2) &= (\neg P_{(R_1 \wedge R_2)} \vee R_1) \ \wedge \ (\neg P_{(R_1 \wedge R_2)} \vee R_2) \\
&\qquad \wedge \ (\neg R_1 \vee \neg R_2 \vee P_{(R_1 \wedge R_2)})
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{En}(F) = \ & (\neg P_{(F)} \vee P_{(Q_1 \wedge Q_2)} \vee P_{(R_1 \wedge R_2)}) \\
& \wedge \ (\neg P_{(Q_1 \wedge Q_2)} \vee P_{(F)}) \\
& \wedge \ (\neg P_{(R_1 \wedge R_2)} \vee P_{(F)})
\end{aligned}
$$

Then

$$F' : \ P_{(F)} \ \wedge \bigwedge_{G \in S_F} \mathsf{En}(G)$$

is equisatisfiable to F and is in CNF. ■

1.7.4 The Resolution Procedure

The next decision procedure that we consider is based on **resolution** and applies only to PL formulae in CNF. Therefore, the procedure of Section 1.7.3 must first be applied to the given PL formula if it is not already in CNF.

Resolution follows from the following observation of any PL formula F in CNF: to satisfy clauses $C_1[P]$ and $C_2[\neg P]$ that share variable P but disagree on its value, either the rest of C_1 or the rest of C_2 must be satisfied. Why? If P is true, then a literal other than $\neg P$ in C_2 must be satisfied; while if P is false, then a literal other than P in C_1 must be satisfied. Therefore, the clause $C_1[\bot] \vee C_2[\bot]$, simplified according to the template equivalences of Exercise 1.2, can be added as a conjunction to F to produce an equivalent formula still in CNF.

Clausal resolution is stated as the following proof rule:

$$\frac{C_1[P] \qquad C_2[\neg P]}{C_1[\bot] \vee C_2[\bot]}$$

From the two clauses of the premise, deduce the new clause, called the **resolvent**.

If ever $\bot$ is deduced via resolution, F must be unsatisfiable since $F \wedge \bot$ is unsatisfiable. Otherwise, if every possible resolution produces a clause that is already known, then F must be satisfiable.

Example 1.26. The CNF of $(P \rightarrow Q) \wedge P \wedge \neg Q$ is the following:

$$F : \ (\neg P \vee Q) \ \wedge \ P \ \wedge \ \neg Q \ .$$

From resolution

$$\frac{(\neg P \vee Q) \qquad P}{Q} \ ,$$

construct

$$F_1 : \ (\neg P \vee Q) \ \wedge \ P \ \wedge \ \neg Q \ \wedge \ Q \ .$$

From resolution

$$\frac{\neg Q \qquad Q}{\bot} \ ,$$

deduce that F, and thus the original formula, is unsatisfiable. ■

Example 1.27. Consider the formula

$$F : \ (\neg P \vee Q) \ \wedge \ \neg Q \ .$$

The one possible resolution

$$\frac{(\neg P \vee Q) \qquad \neg Q}{\neg P}$$

yields

$$F_1 : \ (\neg P \vee Q) \ \wedge \ \neg Q \ \wedge \ \neg P \ .$$

Since no further resolutions are possible, F is satisfiable. Indeed,

$$I : \ \{P \mapsto \mathsf{false}, \ Q \mapsto \mathsf{false}\}$$

is a satisfying interpretation. A CNF formula that does not contain the clause $\bot$ and to which no more resolutions can be applied represents all possible satisfying interpretations. ■

1.7.5 DPLL

Modern satisfiability procedures for propositional logic are based on the Davis-Putnam-Logemann-Loveland algorithm (**DPLL**), which combines the space-efficient procedure of Section 1.7.2 with a restricted form of resolution. We review in this section the basic algorithm. Much research in the past decade has advanced the state-of-the-art considerably.

Like the resolution procedure, DPLL operates on PL formulae in CNF. But again, as the procedure decides satisfiability, we can apply the conversion procedure of Section 1.7.3 to produce a small equisatisfiable CNF formula.

As in the procedure SAT, DPLL attempts to construct an interpretation of F; failing to do so, it reports that the given formula is unsatisfiable. Rather than relying solely on enumerating possibilities, however, DPLL applies a restricted form of resolution to gain some deductive power. The process of applying this restricted resolution as much as possible is called **Boolean constraint propagation** (**BCP**).

BCP is based on **unit resolution**. Unit resolution operates on two clauses. One clause, called the **unit clause**, consists of a single literal ℓ ($\ell = P$ or $\ell = \neg P$ for some propositional variable P). The second clause contains the negation of ℓ: $C[\neg\ell]$. Then unit resolution is the deduction

$$\frac{\ell \qquad C[\neg\ell]}{C[\bot]} \ .$$

Unlike with full resolution, the literals of the resolvent are a subset of the literals of the second clause. Hence, the resolvent replaces the second clause.

Example 1.28. In the formula

$$F: \ (P) \ \wedge \ (\neg P \vee Q) \ \wedge \ (R \vee \neg Q \vee S) \ ,$$

(P) is a unit clause. Therefore, applying unit resolution

$$\frac{P \qquad (\neg P \vee Q)}{Q}$$

produces

$$F': \ (Q) \ \wedge \ (R \vee \neg Q \vee S) \ .$$

Applying unit resolution again

$$\frac{Q \qquad R \vee \neg Q \vee S}{R \vee S}$$

produces

$$F'': \ (R \vee S) \ ,$$

ending this round of BCP. ■

The implementation of DPLL is structurally similar to SAT, except that it begins by applying BCP:

```
let rec DPLL F =
  let F' = BCP F in
  if F' = ⊤ then true
  else if F' = ⊥ then false
  else
    let P = CHOOSE vars(F') in
    (DPLL F'{P ↦ ⊤}) ∨ (DPLL F'{P ↦ ⊥})
```

As in SAT, intermediate formulae are simplified according to the template equivalences of Exercise 1.2.

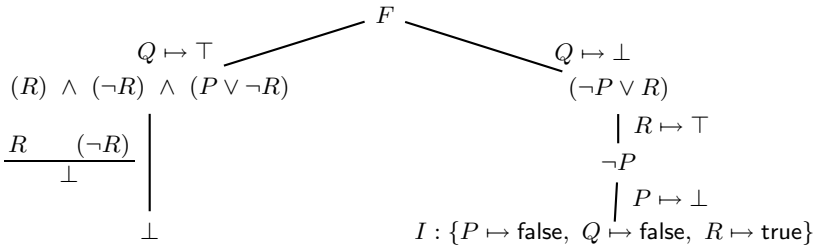

Fig. 1.3. Visualization of Example 1.30

One easy optimization is the following: if variable P appears only positively or only negatively in F, it should not be chosen by CHOOSE $\mathsf{vars}(F')$. P appears only positively when every P-literal is just P; P appears only negatively when every P-literal is $\neg P$. In both cases, F is equisatisfiable to the formula F' constructed by removing all clauses containing an instance of P. Therefore, these clauses do not contribute to BCP. When only such variables remain, the formula must be satisfiable: a full interpretation can be constructed by setting each variable's value based on whether it appears only positively (true) or only negatively (false).

The values to which propositional variables are set on the path to a solution can be recorded so that DPLL can return a satisfying interpretation if one exists, rather than just `true`.

Example 1.29. Consider the formula

$$F : \ (P) \ \wedge \ (\neg P \vee Q) \ \wedge \ (R \vee \neg Q \vee S) \ .$$

On the first level of recursion, DPLL recognizes the unit clause (P) and applies the BCP steps from Example 1.28, resulting in the formula

$$F'' : \ R \vee S \ .$$

The unit resolutions correspond to the partial interpretation

$$\{P \mapsto \mathsf{true}, \ Q \mapsto \mathsf{true}\} \ .$$

Only positively occurring variables remain, so F is satisfiable. In particular,

$$\{P \mapsto \mathsf{true}, \ Q \mapsto \mathsf{true}, \ R \mapsto \mathsf{true}, \ S \mapsto \mathsf{true}\}$$

is a satisfying interpretation of F.

Branching was not required in this example. ■

Example 1.30. Consider the formula

$$F : \ (\neg P \vee Q \vee R) \ \wedge \ (\neg Q \vee R) \ \wedge \ (\neg Q \vee \neg R) \ \wedge \ (P \vee \neg Q \vee \neg R) \ .$$

On the first level of recursion, DPLL must branch. Branching on Q or R will result in unit clauses; choose Q.

Then

$$F\{Q \mapsto \top\} : \ (R) \ \wedge \ (\neg R) \ \wedge \ (P \vee \neg R) \ .$$

The unit resolution

$$\frac{R \qquad (\neg R)}{\bot}$$

finishes this branch.

On the other branch,

$$F\{Q \mapsto \bot\} : \ (\neg P \vee R) \ .$$

P appears only negatively, and R appears only positively, so the formula is satisfiable. In particular, F is satisfied by interpretation

$$I : \ \{P \mapsto \mathsf{false}, \ Q \mapsto \mathsf{false}, \ R \mapsto \mathsf{true}\} \ .$$

This run of DPLL is visualized in Figure 1.3. ■

1.8 Summary

This chapter introduces propositional logic (PL). It covers:

- Its *syntax*. How one constructs a PL formula. Propositional variables, atoms, literals, logical connectives.
- Its *semantics*. What a PL formula means. Truth values true and false. Interpretations. Truth-table definition, inductive definition.
- *Satisfiability* and *validity*. Whether a PL formula evaluates to true under any or all interpretations. Duality of satisfiability and validity, truth-table method, semantic argument method.
- *Equivalence* and *implication*. Whether two formulae always evaluate to the same truth value under every interpretation. Whether under any interpretation, if one formula evaluates to true, the other also evaluates to true. Reduction to validity.
- *Substitution*, which is a tool for manipulating formulae and making general claims. Substitution of equivalent formulae. Valid templates.
- *Normal forms*. A normal form is a set of syntactically restricted formulae such that every PL formula is equivalent to some member of the set.
- *Decision procedures for satisfiability*. Truth-table method, SAT, resolution procedure, DPLL. Transformation to equisatisfiable CNF formula.

PL is an important logic with applications in software and hardware design and analysis, knowledge representation, combinatorial optimization, and complexity theory, to name a few. Although relatively simple, the Boolean structure that is central to PL is often a main source of complexity in applications of the algorithmic reasoning that is the focus of Part II. Exercise 8.1 explores this point in more depth.

Besides being an important logic in its own right, PL serves to introduce the main concepts that are important throughout the book, in particular syntax, semantics, and satisfiability and validity. Chapter 2 presents first-order logic by building on the concepts of this chapter.

Bibliographic Remarks

For a complete and concise presentation of propositional logic, see Smullyan's text *First-Order Logic* [87]. The semantic argument method is similar to Smullyan's tableau method.

The DPLL algorithm is based on work by Davis and Putnam, presented in [26], and by Davis, Logemann, and Loveland, presented in [25].

Exercises

1.1 (PL validity & satisfiability). For each of the following PL formulae, identify whether it is valid or not. If it is valid, prove it with a truth table or semantic argument; otherwise, identify a falsifying interpretation. Recall our conventions for operator precedence and associativity from Section 1.1.

(*a*) $P \wedge Q \rightarrow P \rightarrow Q$
(*b*) $(P \rightarrow Q) \vee P \wedge \neg Q$
(*c*) $(P \rightarrow Q \rightarrow R) \rightarrow P \rightarrow R$
(*d*) $(P \rightarrow Q \vee R) \rightarrow P \rightarrow R$
(*e*) $\neg(P \wedge Q) \rightarrow R \rightarrow \neg R \rightarrow Q$
(*f*) $P \wedge Q \vee \neg P \vee (\neg Q \rightarrow \neg P)$
(*g*) $(P \rightarrow Q \rightarrow R) \rightarrow \neg R \rightarrow \neg Q \rightarrow \neg P$
(*h*) $(\neg R \rightarrow \neg Q \rightarrow \neg P) \rightarrow P \rightarrow Q \rightarrow R$

1.2 (Template equivalences). Use the truth table or semantic argument method to prove the following template equivalences.

(*a*) $\top \Leftrightarrow \neg\bot$
(*b*) $\bot \Leftrightarrow \neg\top$
(*c*) $\neg\neg F \Leftrightarrow F$
(*d*) $F \wedge \top \Leftrightarrow F$
(*e*) $F \wedge \bot \Leftrightarrow \bot$
(*f*) $F \wedge F \Leftrightarrow F$

$(g)\ F \vee \top \ \Leftrightarrow\ \top$
$(h)\ F \vee \bot \ \Leftrightarrow\ F$
$(i)\ F \vee F \ \Leftrightarrow\ F$
$(j)\ F \rightarrow \top \ \Leftrightarrow\ \top$
$(k)\ F \rightarrow \bot \ \Leftrightarrow\ \neg F$
$(l)\ \top \rightarrow F \ \Leftrightarrow\ F$
$(m)\ \bot \rightarrow F \ \Leftrightarrow\ \top$
$(n)\ \top \ \leftrightarrow\ F \ \Leftrightarrow\ F$
$(o)\ \bot \ \leftrightarrow\ F \ \Leftrightarrow\ \neg F$
$(p)\ \neg(F_1 \wedge F_2) \ \Leftrightarrow\ \neg F_1 \vee \neg F_2$
$(q)\ \neg(F_1 \vee F_2) \ \Leftrightarrow\ \neg F_1 \wedge \neg F_2$
$(r)\ F_1 \rightarrow F_2 \ \Leftrightarrow\ \neg F_1 \vee F_2$
$(s)\ F_1 \rightarrow F_2 \ \Leftrightarrow\ \neg F_2 \rightarrow \neg F_1$
$(t)\ \neg(F_1 \rightarrow F_2) \ \Leftrightarrow\ F_1 \wedge \neg F_2$
$(u)\ (F_1 \vee F_2) \ \wedge\ F_3 \ \Leftrightarrow\ (F_1 \wedge F_3) \ \vee\ (F_2 \wedge F_3)$
$(v)\ (F_1 \wedge F_2) \ \vee\ F_3 \ \Leftrightarrow\ (F_1 \vee F_3) \ \wedge\ (F_2 \vee F_3)$
$(w)\ (F_1 \rightarrow F_3) \ \wedge\ (F_2 \rightarrow F_3) \ \Leftrightarrow\ F_1 \vee F_2 \ \rightarrow\ F_3$
$(x)\ (F_1 \rightarrow F_2) \ \wedge\ (F_1 \rightarrow F_3) \ \Leftrightarrow\ F_1 \ \rightarrow\ F_2 \wedge F_3$
$(y)\ F_1 \ \rightarrow\ F_2 \ \rightarrow\ F_3 \ \Leftrightarrow\ F_1 \wedge F_2 \ \rightarrow\ F_3$
$(z)\ (F_1 \leftrightarrow F_2) \ \wedge\ (F_2 \leftrightarrow F_3) \ \Rightarrow\ (F_1 \leftrightarrow F_3)$

1.3 (Redundant logical connectives). Given $\top$, $\wedge$, and $\neg$, prove that $\bot$, $\vee$, $\rightarrow$, and $\leftrightarrow$ are redundant logical connectives. That is, show that each of $\bot$, $F_1 \vee F_2$, $F_1 \rightarrow F_2$, and $F_1 \leftrightarrow F_2$ is equivalent to a formula that uses only F_1, F_2, $\top$, $\vee$, and $\neg$.

1.4 (The nand connective). Let the logical connective $\overline{\wedge}$ (pronounced "nand") be defined according to the following truth table:

F_1	F_2	$F_1 \overline{\wedge} F_2$
0	0	1
0	1	1
1	0	1
1	1	0

Show that all standard logical connectives can be defined in terms of $\overline{\wedge}$.

1.5 (Normal forms). Convert the following PL formulae to NNF, DNF, and CNF via the transformations of Section 1.6.

$(a)\ \neg(P \ \rightarrow\ Q)$
$(b)\ \neg(\neg(P \wedge Q) \ \rightarrow\ \neg R)$
$(c)\ (Q \wedge R \ \rightarrow\ (P \vee \neg Q)) \ \wedge\ (P \vee R)$
$(d)\ \neg(Q \rightarrow R) \ \wedge\ P \ \wedge\ (Q \ \vee\ \neg(P \wedge R))$

1.6 (Graph coloring). A solution to a **graph coloring** problem is an assignment of colors to vertices such that no two adjacent vertices have the same color. Formally, a finite graph $G = \langle V, E\rangle$ consists of vertices $V = \{v_1, \ldots, v_n\}$ and edges $E = \{\langle v_{i_1}, w_{i_1}\rangle, \ldots, \langle v_{i_k}, w_{i_k}\rangle\}$. The finite set of colors is given by $C = \{c_1, \ldots, c_m\}$. A problem instance is given by a graph and a set of colors: the problem is to assign each vertex $v \in V$ a $\mathsf{color}(v) \in C$ such that for every edge $\langle v, w\rangle \in E$, $\mathsf{color}(v) \neq \mathsf{color}(w)$. Clearly, not all instances have solutions.

Show how to encode an instance of a graph coloring problem into a PL formula F. F should be satisfiable iff a graph coloring exists.

(*a*) Describe a set of constraints in PL asserting that every vertex is colored. Since the sets of vertices, edges, and colors are all finite, use notation such as "$\mathsf{color}(v) = c$" to indicate that vertex v has color c. Realize that such an assertion is encodeable as a single propositional variable P_v^c.
(*b*) Describe a set of constraints in PL asserting that every vertex has at most one color.
(*c*) Describe a set of constraints in PL asserting that no two connected vertices have the same color.
(*d*) Identify a significant optimization in this encoding. *Hint*: Can any constraints be dropped? Why?
(*e*) If the constraints are not already in CNF, specify them in CNF now. For N vertices, K edges, and M colors, how many variables does the optimized encoding require? How many clauses?

1.7 (CNF). Example 1.25 constructs a CNF formula that is equisatisfiable to a given small formula in DNF.

(*a*) If distribution of disjunction over conjunction (described in Section 1.6) were used, how many clauses would the resulting formula have?
(*b*) Consider the formulae

$$F_n : \bigvee_{i=1}^{n} (Q_i \wedge R_i)$$

for positive integers n. As a function of n, how many clauses are in
(*i*) the formula F' constructed based on distribution of disjunction over conjunction?
(*ii*) the formula

$$F' : \mathsf{Rep}(F_n) \wedge \bigwedge_{G \in S_{F_n}} \mathsf{En}(G) \ ?$$

(*iii*) For which n is the distribution approach better?

1.8 (DPLL). Describe the execution of DPLL on the following formulae.

(*a*) $(P \vee \neg Q \vee \neg R) \wedge (Q \vee \neg P \vee R) \wedge (R \vee \neg Q)$
(*b*) $(P \vee Q \vee R) \wedge (\neg P \vee \neg Q \vee \neg R) \wedge (\neg P \vee Q \vee R) \wedge (\neg Q \vee R) \wedge (Q \vee \neg R)$

http://www.springer.com/978-3-540-74112-1

The Calculus of Computation
Decision Procedures with Applications to Verification
Bradley, A.R.; Manna, Z.
2007, XVI, 366 p. 60 illus., Hardcover
ISBN: 978-3-540-74112-1