CHAPTER 2

Propositional Logic

2.1 Introduction

Propositional Logic is concerned with propositions and their interrelationships. The notion of a proposition here cannot be defined precisely. Roughly speaking, a *proposition* is a possible condition of the world that is either true or false, e.g. the possibility that it is raining, the possibility that it is cloudy, and so forth. The condition need not be true in order for it to be a proposition. In fact, we might want to say that it is false or that it is true if some other proposition is true.

In this chapter, we first look at the syntactic rules that define the language of Propositional Logic. We then introduce the notion of a truth assignment and use it to define the meaning of Propositional Logic sentences. After that, we present a mechanical method for evaluating sentences for a given truth assignment, and we present a mechanical method for finding truth assignments that satisfy sentences. We conclude with some examples of Propositional Logic in formalizing Natural Language and Digital Circuits.

2.2 Syntax

In Propositional Logic, there are two types of sentences -- simple sentences and compound sentences. Simple sentences express simple facts about the world. Compound sentences express logical relationships between the simpler sentences of which they are composed.

Simple sentences in Propositional Logic are often called *proposition constants* or, sometimes, *logical constants*. In what follows, we write proposition constants as strings of letters, digits, and underscores ("_"), where the first character is a lower case letter. For example, *raining* is a proposition constant, as are *rAiNiNg*, *r32aining*, and *raining_or_snowing*. *Raining* is not a proposition constant because it begins with an upper case character. 324567 fails because it begins with a number. *raining-or-snowing* fails because it contains hyphens (instead of underscores).

Compound sentences are formed from simpler sentences and express relationships among the constituent sentences. There are five types of compound sentences, viz. negations, conjunctions, disjunctions, implications, and biconditionals.

A *negation* consists of the negation operator ¬ and an arbitrary sentence, called the *target*. For example, given the sentence p, we can form the negation of p as shown below.

$$(\neg p)$$

A *conjunction* is a sequence of sentences separated by occurrences of the ∧ operator and enclosed in parentheses, as shown below. The constituent sentences are called *conjuncts*. For example, we can form the conjunction of p and q as follows.

$$(p \wedge q)$$

A *disjunction* is a sequence of sentences separated by occurrences of the ∨ operator and enclosed in parentheses. The constituent sentences are called *disjuncts*. For example, we can form the

disjunction of p and q as follows.

$$(p \vee q)$$

An *implication* consists of a pair of sentences separated by the $\Rightarrow$ operator and enclosed in parentheses. The sentence to the left of the operator is called the *antecedent*, and the sentence to the right is called the *consequent*. The implication of p and q is shown below.

$$(p \Rightarrow q)$$

A *biconditional* is a combination of an implication and a reverse implication. For example, we can express the biconditional of p and q as shown below.

$$(p \Leftrightarrow q)$$

Note that the constituent sentences within any compound sentence can be either simple sentences or compound sentences or a mixture of the two. For example, the following is a legal compound sentence.

$$((p \vee q) \Rightarrow r)$$

One disadvantage of our notation, as written, is that the parentheses tend to build up and need to be matched correctly. It would be nice if we could dispense with parentheses, e.g. simplifying the preceding sentence to the one shown below.

$$p \vee q \Rightarrow r$$

Unfortunately, we cannot do without parentheses entirely, since then we would be unable to render certain sentences unambiguously. For example, the sentence shown above could have resulted from dropping parentheses from either of the following sentences.

$$((p \vee q) \Rightarrow r)$$

$$(p \vee (q \Rightarrow r))$$

The solution to this problem is the use of *operator precedence*. The following table gives a hierarchy of precedences for our operators. The $\neg$ operator has higher precedence than $\wedge$; $\wedge$ has higher precedence than $\vee$; and $\vee$ has higher precedence than $\Rightarrow$ and $\Leftrightarrow$.

$$\begin{array}{c} \neg \\ \wedge \\ \vee \\ \Rightarrow \; \Leftrightarrow \end{array}$$

In sentences without parentheses, it is often the case that an expression is flanked by operators, one on either side. In interpreting such sentences, the question is whether the expression associates with the operator on its left or the one on its right. We can use precedence to make this determination. In particular, we agree that an operand in such a situation always associates with the operator of higher precedence. When an operand is surrounded by operators of equal precedence, the operand associates to the right. The following examples show how these rules work in various cases. The expressions on the right are the fully parenthesized versions of the expressions on the left.

$\neg p \wedge q$	$((\neg p) \wedge q)$
$p \wedge \neg q$	$(p \wedge (\neg q))$
$p \wedge q \vee r$	$((p \wedge q) \vee r)$
$p \vee q \wedge r$	$(p \vee (q \wedge r))$
$p \Rightarrow q \Rightarrow r$	$(p \Rightarrow (q \Rightarrow r))$
$p \Rightarrow q \Leftrightarrow r$	$(p \Rightarrow (q \Leftrightarrow r))$

Note that just because precedence allows us to delete parentheses in some cases does not mean that we can dispense with parentheses entirely. Consider the example shown earlier. Precedence eliminates the ambiguity by dictating that the sentence without parentheses is an implication with a disjunction as antecedent. However, this makes for a problem for those cases when we want to express a disjunction with an implication as a disjunct. In such cases, we must retain at least one pair of parentheses.

We end the section with two simple definitions that are useful in discussing Propositional Logic. A *propositional vocabulary* is a set of proposition constants. A *propositional language* is the set of all propositional sentences that can be formed from a propositional vocabulary.

2.3 Semantics

The treatment of semantics in Logic is similar to its treatment in Algebra. Algebra is unconcerned with the real-world significance of variables. What is interesting are the relationships among the values of the variables expressed in the equations we write. Algebraic methods are designed to respect these relationships, independent of what the variables represent.

In a similar way, Logic is unconcerned with the real world significance of proposition constants. What is interesting is the relationship among the truth values of simple sentences and the truth values of compound sentences within which the simple sentences are contained. As with Algebra, logical reasoning methods are independent of the significance of proposition constants; all that matter is the form of sentences.

Although the values assigned to proposition constants are not crucial in the sense just described, in talking about Logic, it is sometimes useful to make truth assignments explicit and to consider various assignments or all assignments and so forth. Such an assignment is called a truth assignment.

Formally, a *truth assignment* for a propositional vocabulary is a function assigning a truth value to each of the proposition constants of the vocabulary. In what follows, we use the digit 1 as a synonym for *true* and 0 as a synonym for *false*; and we refer to the value of a constant or expression under a truth assignment i by superscripting the constant or expression with i as the superscript.

The assignment shown below is an example for the case of a propositional vocabulary with just three proposition constants, viz. p, q, and r.

$$p^i = 1$$
$$q^i = 0$$
$$r^i = 1$$

The following assignment is another truth assignment for the same vocabulary.

$$p^i = 0$$
$$q^i = 0$$

$$r^i = 1$$

Note that the formulas above are not themselves sentences in Propositional Logic. Propositional Logic does not allow superscripts and does not use the = symbol. Rather, these are informal, metalevel statements *about* particular truth assignments. Although talking about Propositional Logic using a notation similar to that Propositional Logic can sometimes be confusing, it allows us to convey meta-information precisely and efficiently. To minimize problems, in this book we use such meta-notation infrequently and only when there is little chance of confusion.

Looking at the preceding truth assignments, it is important to bear in mind that, as far as logic is concerned, any truth assignment is as good as any other. Logic itself does not fix the truth assignment of individual proposition constants.

On the other hand, *given* a truth assignment for the proposition constants of a language, logic *does* fix the truth assignment for all compound sentences in that language. In fact, it is possible to determine the truth value of a compound sentence by repeatedly applying the following rules.

If the truth value of a sentence is *true*, the truth value of its negation is *false*. If the truth value of a sentence is *false*, the truth value of its negation is *true*.

ϕ	$\neg\phi$
1	0
0	1

The truth value of a conjunction is *true* if and only if the truth values of its conjuncts are both *true*; otherwise, the truth value is *false*.

ϕ	ψ	$\phi \wedge \psi$
1	1	1
1	0	0
0	1	0
0	0	0

The truth value of a disjunction is *true* if and only if the truth value of at least one its disjuncts is *true*; otherwise, the truth value is *false*. Note that this is the *inclusive or* interpretation of the $\vee$ operator and is differentiated from the *exclusive or* interpretation in which a disjunction is true if and only if an odd number of its disjuncts are true.

ϕ	ψ	$\phi \vee \psi$
1	1	1
1	0	1
0	1	1
0	0	0

The truth value of an implication is *false* if and only if its antecedent is *true* and its consequent is *false*; otherwise, the truth value is *true*. This is called *material implication*.

ϕ	ψ	$\phi \Rightarrow \psi$
1	1	1
1	0	0
0	1	1
0	0	1

A biconditional is *true* if and only if the truth values of its constituents agree, i.e. they are either both *true* or both *false*.

ϕ	ψ	$\phi \Leftrightarrow \psi$
1	1	1
1	0	0
0	1	0
0	0	1

Using these definitions, it is easy to determine the truth values of compound sentences with proposition constants as constituents. As we shall see in the next section, we can determine the truth values of compound sentences with other compound sentences as parts by first evaluating the constituent sentences and then applying these definitions to the results.

We finish up this section with a few definitions for future use. We say that a truth assignment *satisfies* a sentence if and only if the sentence is *true* under that truth assignment. We say that a truth assignment *falsifies* a sentence if and only if the sentence is *false* under that truth assignment. A truth assignment satisfies a *set* of sentences if and only if it satisfies *every* sentence in the set. A truth assignment falsifies a *set* of sentences if and only if it falsifies *at least* one sentence in the set.

2.4 Evaluation

Evaluation is the process of determining the truth values of compound sentences given a truth assignment for the truth values of proposition constants.

As it turns out, there is a simple technique for evaluating complex sentences. We substitute true and false values for the proposition constants in our sentence, forming an expression with 1s and 0s and logical operators. We use our operator semantics to evaluate subexpressions with these truth values as arguments. We then repeat, working from the inside out, until we have a truth value for the sentence as a whole.

As an example, consider the truth assignment *i* shown below.

$$p^i = 1$$
$$q^i = 0$$
$$r^i = 1$$

Using our evaluation method, we can see that *i* satisfies $(p \vee q) \wedge (\neg q \vee r)$.

$$(p \vee q) \wedge (\neg q \vee r)$$
$$(1 \vee 0) \wedge (\neg 0 \vee 1)$$
$$1 \wedge (\neg 0 \vee 1)$$
$$1 \wedge (1 \vee 1)$$
$$1 \wedge 1$$
$$1$$

Now consider truth assignment *j* defined as follows.

$$p^j = 0$$
$$q^j = 1$$
$$r^j = 0$$

In this case, *j* does not satisfy $(p \vee q) \wedge (\neg q \vee r)$.

$$(p \vee q) \wedge (\neg q \vee r)$$
$$(0 \vee 1) \wedge (\neg 1 \vee 0)$$
$$1 \wedge (\neg 1 \vee 0)$$
$$1 \wedge (0 \vee 0)$$
$$1 \wedge 0$$
$$0$$

Using this technique, we can evaluate the truth of arbitrary sentences in our language. The cost is proportional to the size of the sentence. Of course, in some cases, it is possible to economize and do even better. For example, when evaluating a conjunction, if we discover that the first conjunct is false, then there is no need to evaluate the second conjunct since the sentence as a whole must be false.

2.5 Satisfaction

Satisfaction is the opposite of evaluation. We begin with one or more compound sentences and try to figure out which truth assignments satisfy those sentences. One nice feature of Propositional Logic is that there are effective procedures for finding truth assignments that satisfy Propositional Logic sentences. In this section, we look at a method based on truth tables.

A *truth table* for a propositional language is a table showing all of the possible truth assignments for the proposition constants in the language. The columns of the table correspond to the proposition constants of the language, and the rows correspond to different truth assignments for those constants.

The following figure shows a truth table for a propositional language with just three proposition constants (p, q, and r). Each column corresponds to one proposition constant, and each row corresponds to a single truth assignment. The truth assignments i and j defined in the preceding section correspond to the third and sixth rows of this table, respectively.

p	***q***	***r***
1	1	1
1	1	0
1	0	1
1	0	0
0	1	1
0	1	0
0	0	1
0	0	0

Note that, for a propositional language with n proposition constants, there are n columns in the truth table and 2^n rows.

In solving satisfaction problems, we start with a truth table for the proposition constants of our language. We then process our sentences in turn, for each sentence placing an x next to rows in the truth table corresponding to truth assignments that do not satisfy the sentence. The truth assignments remaining at the end of this process are all possible truth assignments of the input sentences.

As an example, consider the sentence $p \vee q \Rightarrow q \wedge r$. We can find all truth assignments that satisfy this sentence by constructing a truth table for p, q, and r. See below. We place an x next to each row that does not satisfy the sentence (rows 2, 3, 4, 6). Finally, we take the remaining rows (1, 5, 7, 8) as answers.

	p	*q*	*r*	
	1	1	1	
x	1	1	0	x
x	1	0	1	x
x	1	0	0	x
	0	1	1	
x	0	1	0	x
	0	0	1	
	0	0	0	

The disadvantage of the truth table method is computational complexity. As mentioned above, the size of a truth table for a language grows exponentially with the number of proposition constants in the language. When the number of constants is small, the method works well. When the number is large, the method becomes impractical. Even for moderate sized problems, it can be tedious. Even for an application like Sorority World, where there are only 16 proposition constants, there are 65,536 truth assignments.

Over the years, researchers have proposed ways to improve the performance of truth table checking. However, the best approach to dealing with large vocabularies is to use symbolic manipulation (i.e. logical reasoning and proofs) in place of truth table checking. We discuss these methods in Chapters 4 and 5.

2.6 Example - Natural Language

As an exercise in working with Propositional Logic, let's look at the encoding of various English sentences as formal sentences in Propositional Logic. As we shall see, the structure of English sentences, along with various key words, such as *if* and *no*, determine how such sentences should be translated.

The following examples concern three properties of people, and we assign a different proposition constant to each of these properties. We use the constant *c* to mean that a person is cool. We use the constant *f* to mean that a person is funny. And we use the constant *p* to mean that a person is popular.

As our first example, consider the English sentence *If a person is cool or funny, then he is popular.* Translating this sentence into the language of Propositional Logic is straightforward. The use of the words *if* and *then* suggests an implication. The condition (*cool or funny*) is clearly a disjunction, and the conclusion (*popular*) is just a simple fact. Using the vocabulary from the last paragraph, this leads to the Propositional Logic sentence shown below.

$$c \vee f \Rightarrow p$$

Next, we have the sentence *A person is popular only if he is either cool or funny*. This is similar to the previous sentence, but the presence of the phrase *only if* suggests that the conditionality goes the other way. It is equivalent to the sentence *If a person is popular, then he is either cool or funny*. And this sentence can be translated directly into Propositional Logic as shown below.

$$p \Rightarrow c \vee f$$

A person is popular if and only if he is either cool or funny. The use of the phrase *if and only if* suggests a biconditional, as in the translation shown below. Note that this is the equivalent to the conjunction of the two implications shown above. The biconditional captures this conjunction in a more compact form.

$$p \Leftrightarrow c \vee f$$

Finally, we have a negative sentence. *There is no one who is both cool and funny*. The word *no* here suggests a negation. To make it easier to translate into Propositional Logic, we can first rephrase this as *It is not the case that there is a person who is both cool and funny*. This leads directly to the following encoding.

$$\neg(c \wedge f)$$

Note that, just because we can translate sentences into the language of Propositional Logic does not mean that they are true. The good news is that we can use our evaluation procedure to determine which sentences are true and which are false?

Suppose we were to imagine a person who is cool and funny and popular, i.e. the proposition constants c and f and p are all true. Which of our sentences are true and which are false.

Using the evaluation procedure described earlier, we can see that, for this person, the first sentence is true.

$$\begin{array}{c} c \vee f \Rightarrow p \\ (1 \vee 1) \Rightarrow 1 \\ 1 \Rightarrow 1 \\ 1 \end{array}$$

The second sentence is also true.

$$\begin{array}{c} p \Rightarrow c \vee f \\ 1 \Rightarrow (1 \vee 1) \\ 1 \Rightarrow 1 \\ 1 \end{array}$$

Since the third sentence is really just the conjunction of the first two sentences, it is also true, which we can confirm directly as shown below.

$$\begin{array}{c} p \Leftrightarrow c \vee f \\ 1 \Leftrightarrow (1 \vee 1) \\ 1 \Leftrightarrow 1 \\ 1 \end{array}$$

Unfortunately, the fourth sentence is not true, since the person in this case is both cool and funny.

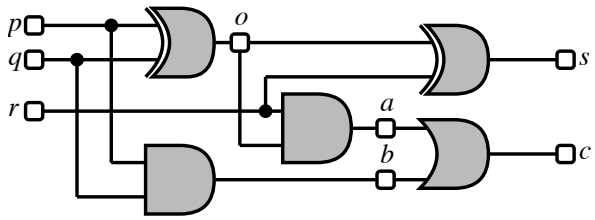
$$\begin{array}{c} \neg(c \wedge f) \\ \neg(1 \wedge 1) \\ \neg 1 \\ 0 \end{array}$$

In this particular case, three of the sentences are true, while one is false. The upshot of this is that there is no such person (assuming that the theory expressed in our sentences is correct). The good news is that there are cases where all four sentences are true, e.g. a person who is cool and popular but not funny or the case of a person who is funny and popular but not cool. Question to consider: What about a person is neither cool nor funny nor popular? Is this possible according to our theory? Which of the sentences would be true and which would be false?

2.7 Example - Digital Circuits

Now let's consider the use of Propositional Logic in modeling a portion of the physical world, in this case, a digital circuit like the ones used in building computers.

The diagram below is a pictorial representation of such a circuit. There are three input *nodes*, some internal nodes, and two output nodes. There are five *gates* connecting these nodes to each other - two *xor* gates (the gates on the top), two *and* gates (the gates on the lower left), and one or gate (the gate on the lower right).



Click on p, q, r to toggle their values.

At a given point in time, a node in a circuit can be either *on* or *off*. The input nodes are set from outside the circuit. A gate sets its output either *on* or *off* based on the type of gate and the values of its input nodes. The output of an *and* gate is *on* if and only if both of its inputs are *on*. The value of an *or* node is *on* if and only if at least one of its inputs is *on*. The output of an *xor* gate is *on* if and only if its inputs disagree with each other.

Given the Boolean nature of signals on nodes and the deterministic character of gates, it is quite natural to model digital circuits in Propositional Logic. We can represent each node of a circuit as a proposition constant, with the idea that the node is *on* if and only if the constant is true. Using the language of Propositional Logic, we can capture the behavior of gates by writing sentences relating the values of the inputs nodes and out nodes of the gates.

The sentences shown below capture the five gates in the circuit shown above. Node o must be *on* if and only if nodes p and q disagree.

$$(p \wedge \neg q) \vee (\neg p \wedge q) \Leftrightarrow o$$
$$r \wedge o \Leftrightarrow a$$
$$p \wedge q \Leftrightarrow b$$
$$(o \wedge \neg r) \vee (\neg o \wedge r) \Leftrightarrow s$$
$$a \vee b \Leftrightarrow c$$

Once we have done this, we can use our formalization to analyze the circuit - to determine if it meets it specification, to test whether a particular instance is operating correctly, and to diagnose the problem in cases here it is not.

Recap

The syntax of Propositional Logic begins with a set of *proposition constants*. Compound sentences are formed by combining simpler sentences with logical operators. In the version of Propositional Logic used here, there are five types of compound sentences - negations, conjunctions, disjunctions, implications, and biconditionals. A *truth assignment* for Propositional Logic is a mapping that assigns a truth value to each of the proposition constants in the language. A truth assignment *satisfies* a sentence if and only if the sentences is *true* under that truth assignment according to rules defining the logical operators of the language. *Evaluation* is the process of determining the truth values of a complex sentence, given a truth assignment for the truth values of proposition constants in that sentence. *Satisfaction* is the process of determining whether or not a sentence has a truth assignment that satisfies it.

Problems

Problem 2.1: Say whether each of the following expressions is a syntactically legal sentence of Propositional Logic.

(*a*) $p \wedge \neg p$
(*b*) $\neg p \vee \neg p$
(*c*) $\neg(q \vee r)\ \neg q \Rightarrow \neg\neg p$
(*d*) $(p \wedge q) \vee (p \neg\wedge q)$
(*e*) $p \vee \neg q \wedge \neg p \vee \neg q \Rightarrow p \vee q$

Problem 2.2: Consider a truth assignment in which p is true, q is false, r is true. Use this truth assignment to evaluate the following sentences.

(*a*) $p \Rightarrow q \wedge r$
(*b*) $p \Rightarrow q \vee r$
(*c*) $p \wedge q \Rightarrow r$
(*d*) $p \wedge q \Rightarrow \neg r$
(*e*) $p \wedge q \Leftrightarrow q \wedge r$

Problem 2.3: A small company makes widgets in a variety of constituent materials (aluminum, copper, iron), colors (red, green, blue, grey), and finishes (matte, textured, coated). Although there are more than one thousand possible combinations of widget features, the company markets only a subset of the possible combinations. The following sentences are constraints that characterize the possibilities. Suppose that a customer places an order for a copper widget that is both green and blue with a matte coating. Your job is to determine which constraints are satisfied and which are violated.

(*a*) $aluminum \vee copper \vee iron$
(*b*) $aluminum \Rightarrow grey$
(*c*) $copper \wedge \neg coated \Rightarrow red$
(*d*) $coated \wedge \neg copper \Rightarrow green$
(*e*) $green \vee blue \Leftrightarrow \neg textured \wedge \neg iron$

Problem 2.4: Consider the sentences shown below. There are three proposition constants here, meaning that there are eight possible truth assignments. How many of these assignments satisfy all of these sentences?

$$p \vee q \vee r$$
$$p \Rightarrow q \wedge r$$
$$q \Rightarrow \neg r$$

Problem 2.5: A small company makes widgets in a variety of constituent materials (aluminum, copper, iron), colors (red, green, blue, grey), and finishes (matte, textured, coated). Although there are more than one thousand possible combinations of widget features, the company markets only a subset of the possible combinations. The sentences below are some constraints that characterize the possibilities. Your job here is to select materials, colors, and finishes in such a way that *all* of the product constraints are satisfied. Note that there are multiple ways this can be done.

$$aluminum \vee copper \vee iron$$
$$red \vee green \vee blue \vee grey$$
$$aluminum \Rightarrow grey$$
$$copper \wedge \neg coated \Rightarrow red$$
$$iron \Rightarrow coated$$

Problem 2.6: Consider a propositional language with three proposition constants - *mushroom*, *purple*, and *poisonous* - each indicating the property suggested by its spelling. Using these proposition constants, encode the following English sentences as Propositional Logic sentences.

(*a*) *All purple mushrooms are poisonous*.
(*b*) *A mushroom is poisonous only if it is purple*.
(*c*) *A mushroom is not poisonous unless it is purple*.
(*d*) *No purple mushroom is poisonous*.

Problem 2.7: Consider the digital circuit described in section 2.7. Suppose we set nodes p, q, and r to be *on*, and we observe that all of the other nodes are *on*. Running our evaluation procedure, we would see that the first sentence in our description of the circuit is not true. Hence the circuit is malfunctioning. Is there any combination of inputs p, q, and r that would result in all other nodes being *on* in a correctly functioning circuit? Hint: To answer this, you need consider a truth table with just eight rows (the possible values for nodes p, q, and r) since all other nodes are observed to be *on*.