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 for the language of Propositional Logic. We then look at the semantics of the expressions specified by these rules. Given this semantics, we talk about evaluation, satisfaction, and the properties of sentences. We then define the concept of propositional entailment, which identifies for us, at least in principle, all of the logical conclusions one can draw from any set of propositional sentences.

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 *propositional 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 logical 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.

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 a simple or compound 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.

$$(p \lor q)$$

An *implication* consists of a pair of sentences separated by the ⇒ 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.

$$(p \Rightarrow q)$$

An *equivalence*, or *biconditional*, is a combination of an implication and a reduction. 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.

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

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.

$$p \lor q \Rightarrow r$$

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.

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

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

The solution to this problem is the use of *operator precedence*. The following table gives a hierarchy of precedences for our operators. The ¬ operator has higher precedence than ∧; ∧ has higher precedence than ∨; and ∨ has higher precedence than ⇒ and ⇔.

$$\begin{gathered} \neg \\ \land \\ \lor \\ \Rightarrow \ \Leftrightarrow \end{gathered}$$

In unparenthesized sentences, 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 operator 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 unparenthesized sentence 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.

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 logical 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 Propositional Logic is a function assigning a truth value to each of the proposition constants of the language. 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 logical language 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 language.

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

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. It does not directly fix the truth assignment of individual proposition constants.

On the other hand, *given* a truth assignment for the logical 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 1, the truth value of its negation is 0. If the truth value of a sentence is 0, the truth value of its negation is 1.

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

- The truth value of a disjunction is *true* if and only if the truth value of at least one its conjuncts is *true*; otherwise, the truth value is *false*. Note that this is the *inclusive or* interpretation of the ∨ 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 false.

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

- 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*.

Given the semantic definitions in the last section, we can easily determine the truth value for any sentence, wether simple or compound, given a truth assignment for our proposition constants. The technique is simple. 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 $(p \vee q) \wedge (\neg q \vee r)$.

$$\begin{gathered}(p \vee q) \wedge (\neg\, q \vee r)\\(1 \vee 0) \wedge (\neg\, 0 \vee 1)\\1 \wedge (\neg\, 0 \vee 1)\\1 \wedge (1 \vee 1)\\1 \wedge 1\\1\end{gathered}$$

Now consider truth assignment *j* defined as follows.

$$\begin{gathered}p^j = 1\\q^j = 1\\r^j = 0\end{gathered}$$

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

$$\begin{gathered}(p \vee q) \wedge (\neg q \vee r)\\(1 \vee 1) \wedge (\neg 1 \vee 0)\\1 \wedge (\neg 1 \vee 0)\\1 \wedge (0 \vee 0)\\1 \wedge \mathit{0}\\0\end{gathered}$$

Using this technique, we can evaluate the truth of arbitrary sentences in our language. The cost is proportional to the size of the sentence.

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 it is *true* under that truth assignment. We say that a truth assignment *falsifies* a sentence if and only if it 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 satisfies *at least* one sentence in the set.

2.4 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 way to do this is using a truth table for the language. A *truth table* for a propositional language is a table showing all of the possible truth assignments for the propositional constants in the language.

The following figure shows a truth table for a propositional language with just three propositional 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 seventh 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* logical constants, there are *n* columns in the truth tables and 2^n rows.

Here is an intuitively simple method of finding truth assignments that satisfy given sentences. (1) We start with a truth table for the proposition constants of our language, and we add columns for each sentence in our set. (2) We then evaluate each sentence for each of the rows of the truth table. (3) Finally, any row that satisfies all sentences in the set is a solution to the problem as a whole. Note that there might be more than one.

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 crossing out truth assignments in the truth table 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.

2.5 Logical Properties of Propositional Sentences

Evaluation and satisfaction are processes that involve specific sentences and specific truth assignments. In Logic, we are sometimes more interested in the properties of sentences that hold across truth assignments. In particular, the notion of satisfaction imposes a classification of sentences in a language based on whether there are truth assignments that satisfy that sentence.

A sentence is *valid* if and only if it is satisfied by *every* truth assignment. For example, the sentence $p \vee \neg p$ is valid.

A sentence is *unsatisfiable* if and only if it is not satisfied by any truth assignment. For example, teh sentence $p \Leftrightarrow \neg p$ is unsatisfiable. No matter what truth assignment we take, the sentence is always false.

A sentence is *contingent* if and only if there is some truth assignment that satisfies it and some truth assignment that falsifies it.

It is frequently useful to group these classifications into two groups. A sentence is *satisfiable* if and only if it is valid or contingent. A sentence is *falsifiable* if and only if it is unsatisfiable or contingent. We have already seen several examples of satisfiable and falsifiable sentences.

In one sense, valid sentences and unsatisfiable sentences are useless. Valid sentences do not rule out any possible truth assignments; unsatisfiable sentences rule out all truth assignments; thus they say nothing about the world. On the other hand, from a logical perspective, they are extremely useful in that, as we shall see, they serve as the basis for legal transformations that we can perform on other logical sentences.

2.6 Propositional Entailment

Validity, satisfiability, and unsatisfiability are properties of individual sentences. In logical reasoning, we are not so much concerned with individual sentences as we are with the relationships between sentences. In particular, we would like to know, given some sentences, whether other sentences are or are not logical conclusions. This relative property is known as *logical entailment*. When we are speaking about Propositional Logic, we use the phrase *propositional entailment*.

A set of sentences Δ *logically entails* a sentence φ (written Δ |= φ) if and only if every truth assignment that satisfies Δ also satisfies φ.

For example, the sentence *p* logically entails the sentence (*p* ∨ *q*). Since a disjunction is true whenever one of its disjuncts is true, then (*p* ∨ *q*) must be true whenever *p* is true. On the other hand, the sentence *p* does *not* logically entail (*p* ∧ *q*). A conjunction is true if and only if *both* of its conjuncts are true, and *q* may be false. Of course, any set of sentences containing both *p* and *q* does logically entail (*p* ∧ *q*).

Note that the relationship of logical entailment is a logical one. Even if the premises of a problem do not logically entail the conclusion, this does not mean that the conclusion is necessarily false, even if the premises are true. It just means that it is *possible* that the conclusion is false.

Once again, consider the case of (*p* ∧ *q*). Although *p* does not logically entail this sentence, it is *possible* that both *p* and *q* are true and, therefore, (*p* ∧ *q*) is true. However, the logical entailment does not hold because it is also possible that *q* is false and, therefore, (*p* ∧ *q*) is false.

Note that logical entailment is not the same as logical equivalence. The sentence *p* logically entails (*p* ∨ *q*), but (*p* ∨ *q*) does not logically entail *p*. Logical entailment is not analogous to arithmetic equality; it is closer to arithmetic inequality.

One way of determining whether or not a set of premises logically entails a possible conclusion is to check the truth table for the proposition constants in the language. This is called the truth table method. (1) First, we form a truth table for the proposition constants and add a column for the premises and a column for the conclusion. (2) We then evaluate the premises. (3) We evaluate the conclusion. (4) Finally, we compare the results. If every row that satisfies the premises also satisfies the conclusion, then the premises logically entail the conclusion.

As an example, let's use this method to show that *p* logically entails (*p* ∨ *q*). We set up our truth table and add a column for our premise and a column for our conclusion. In this case the premise is just p and so evaluation is straightforward; we just copy the column. The conclusion is true if and only if *p* is true or *q* is true. Finally, we notice that every row that satisfies the the premise also satisfies the conclusion.

p	***q***	***p***	***p* ∨ *q***
1	1	1	1
1	0	1	1
0	1	0	1
0	0	0	0

Now, let's do the same for the premise *p* and the conclusion (*p* ∧ *q*). We set up our table as before and evaluate our premise. In this case, there is only one row that satisfies our conclusion. Finally, we notice that the assignment in the second row satisfies our premise but does not satisfy our conclusion; so logical entailment does not hold.

p	q	p	$p \wedge q$
1	1	1	1
1	0	1	0
0	1	0	0
0	0	0	0

Finally, let's look at the problem of determining whether the set of propositions $\{p, q\}$ logically entails $(p \wedge q)$. Here we set up our table as before, but this time we have two premises to satisfy. Only one truth assignment satisfies both premises, and this truth assignment also satisfies the conclusion; hence in this case logical entailment does hold.

p	q	p	q	$p \wedge q$
1	1	1	1	1
1	0	1	0	0
0	1	0	1	0
0	0	0	0	0

As a final example, let's return to the love life of the fickle Mary. Here is the problem from the course introduction. We know $(p \Rightarrow q)$, i.e. if Mary loves Pat, then Mary loves Quincy. We know $(m \Rightarrow p \vee q)$, i.e. if it is Monday, then Mary loves Pat or Quincy. Let's confirm that, if it is Monday, then Mary loves Quincy. We set up our table and evaluate our premises and our conclusion. Both premises are satisfied by the truth assignments on rows 1, 3, 5, 7, and 8; and we notice that those truth assignments make the conclusion true. Hence, the logical entailment holds.

m	p	q	$m \Rightarrow p \vee q$	$p \Rightarrow q$	$m \Rightarrow q$
1	1	1	1	1	1
1	1	0	1	0	0
1	0	1	1	1	1
1	0	0	0	1	0
0	1	1	1	1	1
0	1	0	1	0	1
0	0	1	1	1	1
0	0	0	1	1	1

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. We discuss some of these in Chapter 5. The other approach is to use symbolic manipulation (i.e. logical reasoning and proofs) in place of truth table checking. We discuss these methods in the next chapter.

Before we end this section, it is worth noting that logical entailment and satisfiability have an interesting relationship to each other. In particular, if a set Δ of sentences logically entails a sentence φ, then Δ together with the negation of φ must be unsatisfiable. The reverse is also true. If Δ and the negation of φ is unsatisfiable, then Δ must logically entail φ. This is guaranteed by a metatheorem called the unsatisfiability theorem.

Unsatisfiability Theorem: $\Delta \models \phi$ if and only if $\Delta \cup \{\neg\phi\}$ is unsatisfiable.

Proof: Suppose that $\Delta \models \phi$. If a truth assignment satisfies Δ, then it must also satisfy φ. But then it cannot satisfy ¬φ. Therefore, $\Delta \cup \{\neg\phi\}$ is unsatisfiable. Suppose that $\Delta\cup\{\neg\phi\}$ is unsatisfiable. Then every truth assignment that satisfies Δ must fail to satisfy ¬φ, i.e. it must satisfy φ. Therefore, $\Delta \models \phi$.

An interesting consequence of this result is that we can determine logical entailment by checking for unsatisfiability. This turns out to be useful in various logical proof methods.

Recap

The syntax of propositional logic begins with a set *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 propositional constants in the language. The truth or falsity of compound sentences can be determined from a truth assignment using rules based on the six logical operators of the language. A truth assignment *i satisfies* a sentence if and only if the sentences is *true* under that truth assignment. A sentence is *valid* if and only if it is satisfied by *every* truth assignment. A sentence is *unsatisfiable* if and only if it is not satisfied by any truth assignment. A sentence is *contingent* if and only if it is both satisfiable and falsifiable, i.e. it is neither valid nor unsatisfiable. A sentence is *satisfiable* if and only if it is either valid or contingent. A sentence is *falsifiable* if and only if it is falsifiable or contingent. A set of sentences Δ *logically entails* a sentence φ (written $\Delta \models \phi$) if and only if every truth assignment that satisfies Δ also satisfies φ.