6.4 Propositional Logic: A Very Simple Logic

Despite its limited expressiveness, propositional logic serves to illustrate many of the concepts of logic just as well as first-order logic. We will describe its syntax, semantics, and associated inference procedures.

Syntax

The **syntax** of propositional logic is simple. The symbols of propositional logic are the logical constants *True* and *False*, proposition symbols such as P and Q, the logical connectives $\wedge$, $\vee$, $\Leftrightarrow$, $\Rightarrow$, and $\neg$, and parentheses, ( ). All sentences are made by putting these symbols together using the following rules:

- The logical constants *True* and *False* are sentences by themselves.
- A propositional symbol such as P or Q is a sentence by itself.
- Wrapping parentheses around a sentence yields a sentence, for example, $(P \wedge Q)$.
- A sentence can be formed by combining simpler sentences with one of the five logical connectives:

CONJUNCTION (LOGIC)

DISJUNCTION

$\wedge$ (and). A sentence whose main connective is $\wedge$, such as $P \wedge (Q \vee R)$, is called a **conjunction (logic)**; its parts are the **conjuncts**. (The $\wedge$ looks like an "A" for "And.")

$\vee$ (or). A sentence using $\vee$, such as $A \vee (P \wedge Q)$, is a **disjunction** of the **disjuncts** A and $(P \wedge Q)$. (Historically, the $\vee$ comes from the Latin "vel," which means "or." For most people, it is easier to remember as an upside-down and.)

IMPLICATION
PREMISE
CONCLUSION

$\Rightarrow$ (implies). A sentence such as $(P \wedge Q) \Rightarrow R$ is called an **implication** (or conditional). Its **premise** or **antecedent** is $P \wedge Q$, and its **conclusion** or **consequent** is R. Implications are also known as **rules** or **if–then** statements. The implication symbol is sometimes written in other books as $\supset$ or —.

EQUIVALENCE

$\Leftrightarrow$ (equivalent). The sentence $(P \wedge Q) \Leftrightarrow (Q \wedge P)$ is an **equivalence** (also called a **biconditional**).

NEGATION

$\neg$ (not). A sentence such as $\neg P$ is called the **negation** of P. All the other connectives combine two sentences into one; $\neg$ is the only connective that operates on a single sentence.

ATOMIC SENTENCES
COMPLEX SENTENCES
LITERAL

Figure 6.8 gives a formal grammar of propositional logic; see page 854 if you are not familiar with the BNF notation. The grammar introduces **atomic sentences**, which in propositional logic consist of a single symbol (e.g., P), and **complex sentences**, which contain connectives or parentheses (e.g., $P \wedge Q$). The term **literal** is also used, meaning either an atomic sentences or a negated atomic sentence.

$$
\begin{array}{rcl}
\mathit{Sentence} & \text{—} & \mathit{AtomicSentence} \mid \mathit{ComplexSentence} \\
 & & \\
\mathit{AtomicSentence} & \text{—} & \mathbf{\mathit{True}} \mid \mathbf{\mathit{False}} \\
 & \mid & P \mid Q \mid R \mid \ldots \\
\mathit{ComplexSentence} & \text{—} & (\ \mathit{Sentence}\ ) \\
 & \mid & \mathit{Sentence}\ \mathit{Connective}\ \mathit{Sentence} \\
 & \mid & \neg \mathit{Sentence} \\
 & & \\
\mathit{Connective} & \text{—} & \wedge \mid \vee \mid \Leftrightarrow \mid \Rightarrow
\end{array}
$$

Figure 6.8 A BNF (Backus–Naur Form) grammar of sentences in propositional logic.

Strictly speaking, the grammar is ambiguous—a sentence such as $P \wedge Q \vee R$ could be parsed as either $(P \wedge Q) \vee R$ or as $P \wedge (Q \vee R)$. This is similar to the ambiguity of arithmetic expressions such as $P + Q \times R$, and the way to resolve the ambiguity is also similar: we pick an order of precedence for the operators, but use parentheses whenever there might be confusion. The order of precedence in propositional logic is (from highest to lowest): $\neg$, $\wedge$, $\vee$, $\Rightarrow$, and $\Leftrightarrow$. Hence, the sentence

$$\neg P \vee Q \wedge R \Rightarrow S$$

is equivalent to the sentence

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

Semantics

The **semantics** of propositional logic is also quite straightforward. We define it by specifying the interpretation of the proposition symbols and constants, and specifying the meanings of the logical connectives.

A proposition symbol can mean whatever you want. That is, its interpretation can be any arbitrary fact. The interpretation of P might be the fact that Paris is the capital of France or that the wumpus is dead. A sentence containing just a proposition symbol is satisfiable but not valid: it is true just when the fact that it refers to is the case.

With logical constants, you have no choice; the sentence *True* always has as its interpretation the way the world actually is—the true fact. The sentence *False* always has as its interpretation the way the world is not.

A complex sentence has a meaning derived from the meaning of its parts. Each connective can be thought of as a function. Just as addition is a function that takes two numbers as input and returns a number, so *and* is a function that takes two truth values as input and returns a truth value. We know that one way to define a function is to make a table that gives the output value for every possible input value. For most functions (such as addition), this is impractical because of the size of the table, but there are only two possible truth values, so a logical function with two arguments needs a table with only four entries. Such a table is called a **truth table**. We give truth tables for the logical connectives in Figure 6.9. To use the table to determine, for example, the value of *True* $\vee$ *False*, first look on the left for the row where P is *true* and Q is *false* (the third row). Then look in that row under the $P \vee Q$ column to see the result: *True*.

TRUTH TABLE

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
False	*False*	*True*	*False*	*False*	*True*	*True*
False	*True*	*True*	*False*	*True*	*True*	*False*
True	*False*	*False*	*False*	*True*	*False*	*False*
True	*True*	*False*	*True*	*True*	*True*	*True*

Figure 6.9 Truth tables for the five logical connectives.

Truth tables define the semantics of sentences such as *True* $\wedge$ *True*. Complex sentences such as $(P \vee Q) \wedge \neg S$ are defined by a process of decomposition: first, determine the meaning of $(P \wedge Q)$ and of $\neg S$, and then combine them using the definition of the $\wedge$ function. This is exactly analogous to the way a complex arithmetic expression such as $(p \times q) + -s$ is evaluated.

The truth tables for "and," "or," and "not" are in close accord with our intuitions about the English words. The main point of possible confusion is that $P \vee Q$ is true when either *or both* P and Q are true. There is a different connective called "exclusive or" ("xor" for short) that gives false when both disjuncts are true.[7] There is no consensus on the symbol for exclusive or; two choices are $\dot{\vee}$ and $\oplus$.

In some ways, the implication connective $\Rightarrow$ is the most important, and its truth table might seem puzzling at first, because it does not quite fit our intuitive understanding of "P implies Q"

[7] Latin has a separate word, *aut*, for exclusive or.

or "if P then Q." For one thing, propositional logic does not require any relation of causation or relevance between P and Q. The sentence "5 is odd implies Tokyo is the capital of Japan" is a true sentence of propositional logic (under the normal interpretation), even though it is a decidedly odd sentence of English. Another point of confusion is that any implication is true whenever its antecedent is false. For example, "5 is even implies Sam is smart" is true, regardless of whether Sam is smart. This seems bizarre, but it makes sense if you think of "$P \Rightarrow Q$" as saying, "If P is true, then I am claiming that Q is true. Otherwise I am making no claim."

Validity and inference

Truth tables can be used not only to define the connectives, but also to test for valid sentences. Given a sentence, we make a truth table with one row for each of the possible combinations of truth values for the proposition symbols in the sentence. For each row, we can calculate the truth value of the entire sentence. If the sentence is true in every row, then the sentence is valid. For example, the sentence

$$((P \vee H) \wedge \neg H) \Rightarrow P$$

is valid, as can be seen in Figure 6.10. We include some intermediate columns to make it clear how the final column is derived, but it is not important that the intermediate columns are there, as long as the entries in the final column follow the definitions of the connectives. Suppose P means that there is a wumpus in [1,3] and H means there is a wumpus in [2,2]. If at some point we learn $(P \vee H)$ and then we also learn $\neg H$, then we can use the valid sentence above to conclude that P is true—that the wumpus is in [1,3].

P	H	$P \vee H$	$(P \vee H) \wedge \neg H$	$((P \vee H) \wedge \neg H) \Rightarrow P$
False	*False*	*False*	*False*	*True*
False	*True*	*True*	*False*	*True*
True	*False*	*True*	*True*	*True*
True	*True*	*True*	*False*	*True*

Figure 6.10 Truth table showing validity of a complex sentence.

This is important. It says that if a machine has some premises and a possible conclusion, it can determine if the conclusion is true. It can do this by building a truth table for the sentence *Premises* $\Rightarrow$ *Conclusion* and checking all the rows. If every row is true, then the conclusion is entailed by the premises, which means that the fact represented by the conclusion follows from the state of affairs represented by the premises. Even though the machine has no idea what the conclusion means, the user could read the conclusions and use his or her interpretation of the proposition symbols to see what the conclusions mean—in this case, that the wumpus is in [1,3]. Thus, we have fulfilled the promise made in Section 6.3.

It will often be the case that the sentences input into the knowledge base by the user refer to a world to which the computer has no independent access, as in Figure 6.11, where it is the user who observes the world and types sentences into the computer. It is therefore essential that a reasoning system be able to draw conclusions that follow from the premises, regardless of the world to which the sentences are intended to refer. But it is a good idea for a reasoning system to follow this principle in any case. Suppose we replace the "user" in Figure 6.11 with a camera-based visual processing system that sends input sentences to the reasoning system. It makes no difference! Even though the computer now has "direct access" to the world, inference can still take place through direct operations on the syntax of sentences, without any additional information as to their intended meaning.

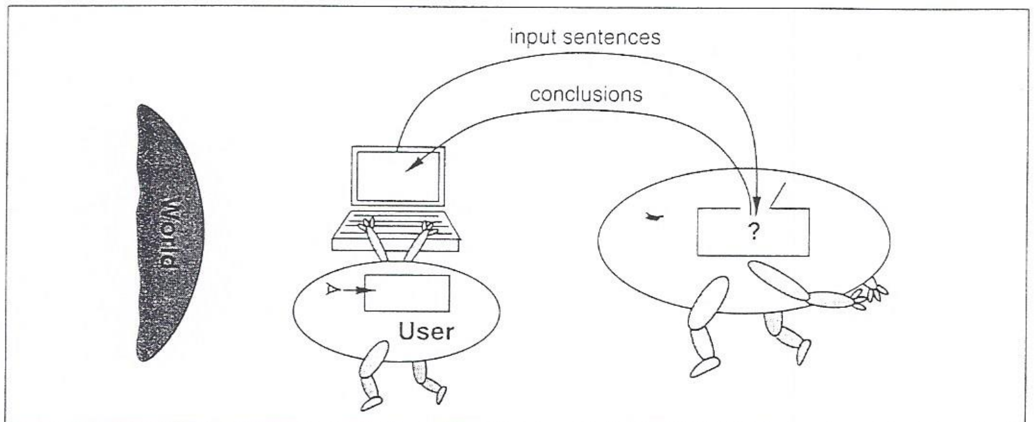

Figure 6.11 Sentences often refer to a world to which the agent has no independent access.

Models

MODEL

Any world in which a sentence is true under a particular interpretation is called a **model** of that sentence under that interpretation. Thus, the world shown in Figure 6.2 is a model of the sentence "$S_{1,2}$" under the interpretation in which it means that there is a stench in [1,2]. There are many other models of this sentence—you can make up a world that does or does not have pits and gold in various locations, and as long as the world has a stench in [1,2], it is a model of the sentence. The reason there are so many models is because "$S_{1,2}$" makes a very weak claim about the world. The more we claim (i.e., the more conjunctions we add into the knowledge base), the fewer the models there will be.

Models are very important in logic, because, to restate the definition of **entailment**, *a sentence α is entailed by a knowledge base KB if the models of KB are all models of α*. If this is the case, then whenever *KB* is true, α must also be true.

In fact, we could define the meaning of a sentence by means of set operations on sets of models. For example, the set of models of $P \wedge Q$ is the intersection of the models of P and the models of Q. Figure 6.12 diagrams the set relationships for the four binary connectives.

We have said that models are worlds. One might feel that real worlds are rather messy things on which to base a formal system. Some authors prefer to think of models as *mathematical* objects. In this view, a model in propositional logic is simply a mapping from proposition symbols

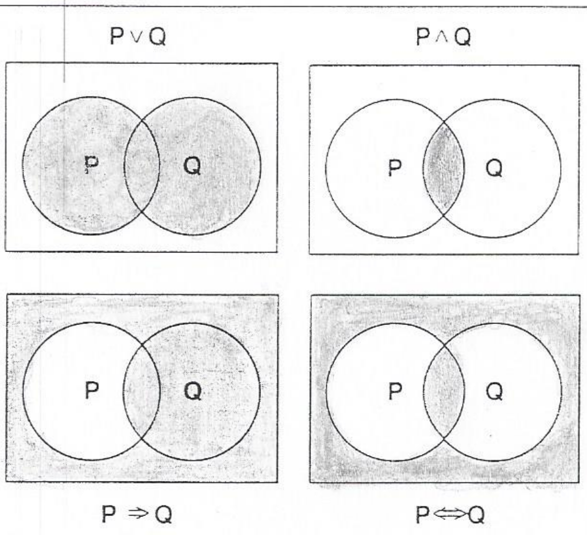

Figure 6.12 Models of complex sentences in terms of the models of their components. In each diagram, the shaded parts correspond to the models of the complex sentence.

directly to truth and falsehood, that is, the label for a row in a truth table. Then the models of a sentence are just those mappings that make the sentence true. The two views can easily be reconciled because each possible assignment of true and false to a set of proposition symbols can be viewed as an equivalence class of worlds that, under a given interpretation, have those truth values for those symbols. There may of course be many different "real worlds" that have the same truth values for those symbols. The only requirement to complete the reconciliation is that each proposition symbol be *either true or false* in each world. This is, of course, the basic ontological assumption of propositional logic, and is what allows us to expect that manipulations of symbols lead to conclusions with reliable counterparts in the actual world.

Rules of inference for propositional logic

The process by which the soundness of an inference is established through truth tables can be extended to entire *classes* of inferences. There are certain patterns of inferences that occur over and over again, and their soundness can be shown once and for all. Then the pattern can be captured in what is called an **inference rule**. Once a rule is established, it can be used to make inferences without going through the tedious process of building truth tables.

INFERENCE RULE

We have already seen the notation $\alpha \vdash \beta$ to say that β can be derived from α by inference. There is an alternative notation,

$$\frac{\alpha}{\beta}$$

which emphasizes that this is not a sentence, but rather an inference rule. Whenever something in the knowledge base matches the pattern above the line, the inference rule concludes the premise below the line. The letters α, β, etc., are intended to match any sentence, not just individual proposition symbols. If there are several sentences, in either the premise or the conclusion, they are separated by commas. Figure 6.13 gives a list of seven commonly used inference rules.

An inference rule is sound if the conclusion is true in all cases where the premises are true. To verify soundness, we therefore construct a truth table with one line for each possible model of the proposition symbols in the premise, and show that in all models where the premise is true, the conclusion is also true. Figure 6.14 shows the truth table for the resolution rule.

◇ **Modus Ponens** or **Implication-Elimination**: (From an implication and the premise of the implication, you can infer the conclusion.)

$$\frac{\alpha \Rightarrow \beta, \qquad \alpha}{\beta}$$

◇ **And-Elimination**: (From a conjunction, you can infer any of the conjuncts.)

$$\frac{\alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n}{\alpha_i}$$

◇ **And-Introduction**: (From a list of sentences, you can infer their conjunction.)

$$\frac{\alpha_1, \ \alpha_2, \quad \ldots, \quad \alpha_n}{\alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n}$$

◇ **Or-Introduction**: (From a sentence, you can infer its disjunction with anything else at all.)

$$\frac{\alpha_i}{\alpha_1 \vee \alpha_2 \vee \ldots \vee \alpha_n}$$

◇ **Double-Negation Elimination**: (From a doubly negated sentence, you can infer a positive sentence.)

$$\frac{\neg\neg\alpha}{\alpha}$$

◇ **Unit Resolution**: (From a disjunction, if one of the disjuncts is false, then you can infer the other one is true.)

$$\frac{\alpha \vee \beta, \qquad \neg\beta}{\alpha}$$

◇ **Resolution**: (This is the most difficult. Because β cannot be both true and false, one of the other disjuncts must be true in one of the premises. Or equivalently, implication is transitive.)

$$\frac{\alpha \vee \beta, \qquad \neg\beta \vee \gamma}{\alpha \vee \gamma} \quad \text{or equivalently} \quad \frac{\neg\alpha \Rightarrow \beta, \qquad \beta \Rightarrow \gamma}{\neg\alpha \Rightarrow \gamma}$$

Figure 6.13 Seven inference rules for propositional logic. The unit resolution rule is a special case of the resolution rule, which in turn is a special case of the full resolution rule for first-order logic discussed in Chapter 9.

α	β	γ	$\alpha \vee \beta$	$\neg\beta \vee \gamma$	$\alpha \vee \gamma$
False	*False*	*False*	*False*	*True*	*False*
False	*False*	*True*	*False*	*True*	*True*
False	*True*	*False*	*True*	*False*	*False*
False	*True*	*True*	*True*	*True*	*True*
True	*False*	*False*	*True*	*True*	*True*
True	*False*	*True*	*True*	*True*	*True*
True	*True*	*False*	*True*	*False*	*True*
True	*True*	*True*	*True*	*True*	*True*

Figure 6.14 A truth table demonstrating the soundness of the resolution inference rule. We have underlined the rows where both premises are true.

As we mentioned above, a logical proof consists of a sequence of applications of inference rules, starting with sentences initially in the KB, and culminating in the generation of the sentence whose proof is desired. To prove that P follows from $(P \vee H)$ and $\neg H$, for example, we simply require one application of the resolution rule, with α as P, β as H, and γ empty. The job of an inference procedure, then, is to construct proofs by finding appropriate sequences of applications of inference rules.

Complexity of propositional inference

The truth-table method of inference described on page 169 is complete, because it is always possible to enumerate the 2^n rows of the table for any proof involving n proposition symbols. On the other hand, the computation time is exponential in n, and therefore impractical. One might wonder whether there is a polynomial-time proof procedure for propositional logic based on using the inference rules from Section 6.4.

In fact, a version of this very problem was the first addressed by Cook (1971) in his theory of NP-completeness. (See also the appendix on complexity.) Cook showed that checking a set of sentences for satisfiability is NP-complete, and therefore unlikely to yield to a polynomial-time algorithm. However, this does not mean that all instances of propositional inference are going to take time proportional to 2^n. In many cases, the proof of a given sentence refers only to a small subset of the KB and can be found fairly quickly. In fact, as Exercise 6.15 shows, really hard problems are quite rare.

MONOTONICITY

The use of inference rules to draw conclusions from a knowledge base relies implicitly on a general property of certain logics (including propositional and first-order logic) called **monotonicity**. Suppose that a knowledge base KB entails some set of sentences. A logic is monotonic if when we add some new sentences to the knowledge base, all the sentences entailed by the original KB are still entailed by the new larger knowledge base. Formally, we can state the property of monotonicity of a logic as follows:

$$\textbf{if } KB_1 \models \alpha \textbf{ then } (KB_1 \cup KB_2) \models \alpha$$

This is true regardless of the contents of KB_2—it can be irrelevant or even contradictory to KB_1.

It is fairly easy to show that propositional and first-order logic are monotonic in this sense; one can also show that probability theory is not monotonic (see Chapter 14). An inference rule such as Modus Ponens is **local** because its premise need only be compared with a small portion of the KB (two sentences, in fact). Were it not for monotonicity, we could not have any local inference rules because the rest of the KB might affect the soundness of the inference. This would potentially cripple any inference procedure.

LOCAL

There is also a useful class of sentences for which a polynomial-time inference procedure exists. This is the class called **Horn sentences**.[3] A Horn sentence has the form:

HORN SENTENCES

$$P_1 \land P_2 \land \ldots \land P_n \Rightarrow Q$$

where the P_i and Q are nonnegated atoms. There are two important special cases: First, when Q is the constant *False*, we get a sentence that is equivalent to $\neg P_1 \lor \ldots \lor \neg P_n$. Second, when $n = 1$ and $P_1 = True$, we get $True \Rightarrow Q$, which is equivalent to the atomic sentence Q. Not every knowledge base can be written as a collection of Horn sentences, but for those that can, we can use a simple inference procedure: apply Modus Ponens wherever possible until no new inferences remain to be made. We discuss Horn sentences and their associated inference procedures in more detail in the context of first-order logic (Section 9.4).