

7.4 Propositional Logic: A Very Simple Logic

We now present **propositional logic**. We describe its syntax (the structure of sentences) and its semantics (the way in which the truth of sentences is determined). From these, we derive a simple, syntactic algorithm for logical inference that implements the semantic notion of entailment. Everything takes place, of course, in the wumpus world.

Propositional logic

7.4.1 Syntax

The **syntax** of propositional logic defines the allowable sentences. The **atomic sentences** consist of a single **proposition symbol**. Each such symbol stands for a proposition that can be true or false. We use symbols that start with an uppercase letter and may contain other letters or subscripts, for example: P , Q , R , $W_{1,3}$, and *FacingEast*. The names are arbitrary but are often chosen to have some mnemonic value—we use $W_{1,3}$ to stand for the proposition that the wumpus is in $[1,3]$. (Remember that symbols such as $W_{1,3}$ are *atomic*, i.e., W , 1 , and 3 are not meaningful parts of the symbol.) There are two proposition symbols with fixed meanings: *True* is the always-true proposition and *False* is the always-false proposition. **Complex sentences** are constructed from simpler sentences, using parentheses and operators called **logical connectives**. There are five connectives in common use:

Atomic sentences

Proposition symbol

Complex sentences

Logical connectives

\neg (not). A sentence such as $\neg W_{1,3}$ is called the **negation** of $W_{1,3}$. A **literal** is either an atomic sentence (a **positive literal**) or a negated atomic sentence (a **negative literal**).

Negation

Literal

\wedge (and). A sentence whose main connective is \wedge , such as $W_{1,3} \wedge P_{3,1}$, is called a **conjunction**; its parts are the **conjuncts**. (The \wedge looks like an “A” for “And.”)

Conjunction

\vee (or). A sentence whose main connective is \vee , such as $(W_{1,3} \wedge P_{3,1}) \vee W_{2,2}$, is a **disjunction**; its parts are **disjuncts**—in this example, $(W_{1,3} \wedge P_{3,1})$ and $W_{2,2}$.

Disjunction

\Rightarrow (implies). A sentence such as $(W_{1,3} \wedge P_{3,1}) \Rightarrow \neg W_{2,2}$ is called an **implication** (or conditional). Its **premise** or **antecedent** is $(W_{1,3} \wedge P_{3,1})$, and its **conclusion** or **consequent** is $\neg W_{2,2}$. Implications are also known as **rules** or **if-then** statements. The implication symbol is sometimes written in other books as \supset or \rightarrow .

Implication


Premise

Conclusion

Rules

\Leftrightarrow (if and only if). The sentence $W_{1,3} \Leftrightarrow \neg W_{2,2}$ is a **biconditional**.

Biconditional

Figure 7.7  gives a formal grammar of propositional logic. (BNF notation is explained on page 1030.) The BNF grammar is augmented with an operator precedence list to remove ambiguity when multiple operators are used. The “not” operator (\neg) has the highest precedence, which means that in the sentence $\neg A \wedge B$ the \neg binds most tightly, giving us the equivalent of $(\neg A) \wedge B$ rather than $\neg(A \wedge B)$. (The notation for ordinary arithmetic is the

same: $-2 + 4$ is 2, not -6 .) When appropriate, we also use parentheses and square brackets to clarify the intended sentence structure and improve readability.

Figure 7.7

$$\begin{aligned}
 \textit{Sentence} &\rightarrow \textit{AtomicSentence} \mid \textit{ComplexSentence} \\
 \textit{AtomicSentence} &\rightarrow \textit{True} \mid \textit{False} \mid P \mid Q \mid R \mid \dots \\
 \textit{ComplexSentence} &\rightarrow (\textit{Sentence}) \\
 &\mid \neg \textit{Sentence} \\
 &\mid \textit{Sentence} \wedge \textit{Sentence} \\
 &\mid \textit{Sentence} \vee \textit{Sentence} \\
 &\mid \textit{Sentence} \Rightarrow \textit{Sentence} \\
 &\mid \textit{Sentence} \Leftrightarrow \textit{Sentence}
 \end{aligned}$$

OPERATOR PRECEDENCE : $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$

A BNF (Backus–Naur Form) grammar of sentences in propositional logic, along with operator precedences, from highest to lowest.

7.4.2 Semantics

Having specified the syntax of propositional logic, we now specify its semantics. The semantics defines the rules for determining the truth of a sentence with respect to a particular model. In propositional logic, a model simply sets the **truth value**—*true* or *false*—for every proposition symbol. For example, if the sentences in the knowledge base make use of the proposition symbols $P_{1,2}$, $P_{2,2}$, and $P_{3,1}$, then one possible model is

$$m_1 = \{P_{1,2} = \textit{false}, P_{2,2} = \textit{false}, P_{3,1} = \textit{true}\}.$$

Truth value

With three proposition symbols, there are $2^3 = 8$ possible models—exactly those depicted in [Figure 7.5](#). Notice, however, that the models are purely mathematical objects with no necessary connection to wumpus worlds. $P_{1,2}$ is just a symbol; it might mean “there is a pit in $[1,2]$ ” or “I’m in Paris today and tomorrow.”

The semantics for propositional logic must specify how to compute the truth value of *any* sentence, given a model. This is done recursively. All sentences are constructed from atomic sentences and the five connectives; therefore, we need to specify how to compute the truth of atomic sentences and how to compute the truth of sentences formed with each of the five connectives. Atomic sentences are easy:

- *True* is true in every model and *False* is false in every model.
- The truth value of every other proposition symbol must be specified directly in the model. For example, in the model m_1 given earlier, $P_{1,2}$ is false.

For complex sentences, we have five rules, which hold for any subsentences P and Q (atomic or complex) in any model m (here “iff” means “if and only if”):

- $\neg P$ is true iff P is false in m .
- $P \wedge Q$ is true iff both P and Q are true in m .
- $P \vee Q$ is true iff either P or Q is true in m .
- $P \Rightarrow Q$ is true unless P is true and Q is false in m .
- $P \Leftrightarrow Q$ is true iff P and Q are both true or both false in m .

The rules can also be expressed with **truth tables** that specify the truth value of a complex sentence for each possible assignment of truth values to its components. Truth tables for the five connectives are given in [Figure 7.8](#). From these tables, the truth value of any sentence s can be computed with respect to any model m by a simple recursive evaluation. For example, the sentence $\neg P_{1,2} \wedge (P_{2,2} \vee P_{3,1})$, evaluated in m_1 , gives $true \wedge (false \vee true) = true \wedge true = true$. Exercise [Z.TRUEV](#) asks you to write the algorithm $PL\text{-}TRUE?(s, m)$, which computes the truth value of a propositional logic sentence s in a model m .

Figure 7.8

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

Truth tables for the five logical connectives. To use the table to compute, for example, the value of $P \vee Q$ when P is true and Q is 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

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 P is true or Q is true *or both*. A different connective, called “exclusive or” (“xor” for short), yields false when both disjuncts are true.⁸ There is no consensus on the symbol for exclusive or; some choices are $\dot{\vee}$ or \neq or \oplus .

⁸ Latin uses two separate words: “vel” is inclusive or and “aut” is exclusive or.

The truth table for \Rightarrow may not quite fit one’s intuitive understanding of “ P implies Q ” 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.” The only way for this sentence to be *false* is if P is true but Q is false.

The biconditional, $P \Leftrightarrow Q$, is true whenever both $P \Rightarrow Q$ and $Q \Rightarrow P$ are true. In English, this is often written as “ P if and only if Q .” Many of the rules of the wumpus world are best written using \Leftrightarrow . For example, a square is breezy *if* a neighboring square has a pit, and a square is breezy *only if* a neighboring square has a pit. So we need a biconditional,

$$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1}),$$

where $B_{1,1}$ means that there is a breeze in $[1,1]$.

7.4.3 A simple knowledge base

Now that we have defined the semantics for propositional logic, we can construct a knowledge base for the wumpus world. We focus first on the *immutable* aspects of the wumpus world, leaving the mutable aspects for a later section. For now, we need the following symbols for each $[x,y]$ location:

$P_{x,y}$ is true if there is a pit in $[x,y]$.

$W_{x,y}$ is true if there is a wumpus in $[x,y]$, dead or alive.

$B_{x,y}$ is true if there is a breeze in $[x,y]$.

$S_{x,y}$ is true if there is a stench in $[x,y]$.

$L_{x,y}$ is true if the agent is in location $[x,y]$.

The sentences we write will suffice to derive $\neg P_{1,2}$ (there is no pit in $[1,2]$), as was done informally in [Section 7.3](#). We label each sentence R_i so that we can refer to them:

- There is no pit in $[1,1]$:

$$R_1 : \quad \neg P_{1,1}.$$

- A square is breezy if and only if there is a pit in a neighboring square. This has to be stated for each square; for now, we include just the relevant squares:

$$R_2 : \quad B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1}).$$

$$R_3 : \quad B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1}).$$

- The preceding sentences are true in all wumpus worlds. Now we include the breeze percepts for the first two squares visited in the specific world the agent is in, leading up to the situation in [Figure 7.3\(b\)](#).

$$R_4 : \neg B_{1,1} .$$

$$R_5 : B_{2,1} .$$

7.4.4 A simple inference procedure

Our goal now is to decide whether $KB \models \alpha$ for some sentence α . For example, is $\neg P_{1,2}$ entailed by our KB ? Our first algorithm for inference is a model-checking approach that is a direct implementation of the definition of entailment: enumerate the models, and check that α is true in every model in which KB is true. Models are assignments of *true* or *false* to every proposition symbol. Returning to our wumpus-world example, the relevant proposition symbols are $B_{1,1}$, $B_{2,1}$, $P_{1,1}$, $P_{1,2}$, $P_{2,1}$, $P_{2,2}$, and $P_{3,1}$. With seven symbols, there are $2^7 = 128$ possible models; in three of these, KB is true (Figure 7.9). In those three models, $\neg P_{1,2}$ is true, hence there is no pit in $[1,2]$. On the other hand, $P_{2,2}$ is true in two of the three models and false in one, so we cannot yet tell whether there is a pit in $[2,2]$.

Figure 7.9

$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	R_1	R_2	R_3	R_4	R_5	KB
false	false	false	false	false	false	false	true	true	true	true	false	false
false	false	false	false	false	false	true	true	true	false	true	false	false
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
false	true	false	false	false	false	false	true	true	false	true	true	false
false	true	false	false	false	false	true	true	true	true	true	true	<u>true</u>
false	true	false	false	false	true	false	true	true	true	true	true	<u>true</u>
false	true	false	false	false	true	true	true	true	true	true	true	<u>true</u>
false	true	false	false	true	false	false	true	false	false	true	true	false
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
true	true	true	true	true	true	true	false	true	true	false	true	false

A truth table constructed for the knowledge base given in the text. KB is true if R_1 through R_5 are true, which occurs in just 3 of the 128 rows (the ones underlined in the right-hand column). In all 3 rows, $P_{1,2}$ is false, so there is no pit in $[1,2]$. On the other hand, there might (or might not) be a pit in $[2,2]$.

Figure 7.9 reproduces in a more precise form the reasoning illustrated in Figure 7.5. A general algorithm for deciding entailment in propositional logic is shown in Figure 7.10. Like the BACKTRACKING-SEARCH algorithm on page 192, TT-ENTAILS? performs a recursive enumeration of a finite space of assignments to symbols. The algorithm is **sound** because it implements directly the definition of entailment, and **complete** because it works for any KB and α and always terminates—there are only finitely many models to examine.


Figure 7.10

```
function TT-ENTAILS?(KB,  $\alpha$ ) returns true or false
  inputs: KB, the knowledge base, a sentence in propositional logic
            $\alpha$ , the query, a sentence in propositional logic

  symbols  $\leftarrow$  a list of the proposition symbols in KB and  $\alpha$ 
  return TT-CHECK-ALL(KB,  $\alpha$ , symbols, { })

function TT-CHECK-ALL(KB,  $\alpha$ , symbols, model) returns true or false
  if EMPTY?(symbols) then
    if PL-TRUE?(KB, model) then return PL-TRUE?( $\alpha$ , model)
    else return true      // when KB is false, always return true
  else
    P  $\leftarrow$  FIRST(symbols)
    rest  $\leftarrow$  REST(symbols)
    return (TT-CHECK-ALL(KB,  $\alpha$ , rest, model  $\cup$  {P = true})
           and
           TT-CHECK-ALL(KB,  $\alpha$ , rest, model  $\cup$  {P = false })))
```

A truth-table enumeration algorithm for deciding propositional entailment. (TT stands for truth table.) PL-TRUE? returns *true* if a sentence holds within a model. The variable *model* represents a partial model—an assignment to some of the symbols. The keyword **and** here is an infix function symbol in the pseudocode programming language, not an operator in proposition logic; it takes two arguments and returns *true* or *false*.

Of course, “finitely many” is not always the same as “few.” If *KB* and α contain n symbols in all, then there are 2^n models. Thus, the time complexity of the algorithm is $O(2^n)$. (The space complexity is only $O(n)$ because the enumeration is depth-first.) Later in this chapter we show algorithms that are much more efficient in many cases. Unfortunately, propositional entailment is co-NP-complete (i.e., probably no easier than NP-complete—see [Appendix A](#) ) , so every known inference algorithm for propositional logic has a worst-case complexity that is exponential in the size of the input.