Chapter 1

Propositional Logic

A logic is defined by its syntax and semantics. The syntax of a logic defines the class of statements that can be used to convey information. The semantics of a logic defines properties of logical statements, such as satisfiability and validity, in addition to relationships among these statements, such as equivalence and implication. The goal of this chapter is to define the syntax and semantics of propositional logic: one of the simplest and most commonly used logics.

1.1 Syntax

Propositional logic statements are formed using a set of propositional variables, \( P_1, \ldots, P_n \). These variables—which are also called Boolean variables, propositional symbols, atomic propositions, and atoms—are variables that assume one of two values, typically indicated by true and false. The simplest statement one can write in propositional logic has the form \( P_i \). It is called an atomic statement and is interpreted as saying that variable \( P_i \) takes on the value true.

More complex statements in propositional logic are formed using logical connectives. Specifically, sentences in propositional logic are formed according to the following rules:

- Every propositional variable is a sentence.
- If \( \alpha \) and \( \beta \) are sentences, then \( \neg\alpha \), \( \alpha \land \beta \), and \( \alpha \lor \beta \) are also sentences.

The symbols \( \neg \), \( \land \) and \( \lor \) are called logical connectives and they stand for negation, conjunction and disjunction, respectively. Other connectives can also be introduced such as implication \( \Rightarrow \) and equivalence \( \Leftrightarrow \) but these are defined in terms of the three primitive connectives given above. In particular, the sentence \( \alpha \Rightarrow \beta \) is a shorthand for \( \neg\alpha \lor \beta \). Similarly, the sentence \( \alpha \Leftrightarrow \beta \) is a shorthand for \( (\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha) \).

An important syntactic notion in propositional logic is the propositional literal, which is a propositional variable \( P_i \), called a positive literal, or the negation of a propositional variable \( \neg P_i \), called a negative literal.
Propositional knowledge bases

Another important syntactic notion is the propositional knowledge base, which is a set of propositional sentences $\Delta = \alpha_1, \alpha_2, \ldots$. The knowledge base $\Delta$ is a shorthand for the sentence $\alpha_1 \land \alpha_2 \land \ldots$, which is formed by conjoining all sentences in the knowledge base. Notating such a potentially large sentence using a set of smaller sentences is quite convenient as it allows us to break up the large sentence into pieces that are written on multiple lines:

$$\Delta_1 = \neg A \lor B$$

to describe the circuit behavior. In this case, we are asserting conditions on inputs $A$ and $B$ under which the circuit would output a high value.

Another knowledge base one can write involves the set of propositional variables $\Sigma_2 = A, B, C$:

$$\Delta_2 = \{ \neg A \lor B \implies C \land \neg(\neg A \lor B) \implies \neg C \}$$

which includes a propositional variable for the circuit output.
A third knowledge base one can write involves the set of propositional variables $\Sigma_3 = A, B, C, X, Y$:

$$\Delta_3 = \left\{ \begin{array}{cccc}
A & \Rightarrow & \neg X \\
\neg A & \Rightarrow & X \\
A \land B & \Rightarrow & Y \\
\neg(A \land B) & \Rightarrow & \neg Y \\
X \lor Y & \Rightarrow & C \\
\neg(X \lor Y) & \Rightarrow & \neg C,
\end{array} \right. $$

which includes propositional variables for the internal wires in the circuit. Obviously, these knowledge bases vary in the amount of information they convey, where knowledge base $\Delta_3$ is the most informative, yet also the most complex. Which one of these knowledge bases is most suitable depends on the application we have in mind. For example, if our goal is to check whether two circuits over inputs $A$ and $B$ have the same functionality, then knowledge base $\Delta_1$ is sufficient since it completely characterizes the functionality of given circuit. However, if we are interested in reasoning about the internal behavior of the circuit—such as whether $X$ being high implies that $Y$ is low—then knowledge base $\Delta_3$ is what we need since the other two knowledge bases do not even allow us to phrase logical questions about the state of internal wires in the circuit.

The main point behind the previous example is that the choice of propositional variables is quite important, and must be made in the context of given application, which in turns fixes the kind of logical questions one may want to pose with respect to the knowledge base.

### 1.2 Semantics

The syntax of propositional logic specifies rules for composing grammatically correct sentences. The semantics of propositional logic defines logical properties of such sentences, including consistency and validity, and logical relationships among sentences, including implication and equivalence.

These logical properties and relationships are easy to figure out for simple sentences. For example, most people would agree that

- $A \land \neg A$ is inconsistent,
- $A \lor \neg A$ is valid,
- $A$ and $A \Rightarrow B$ imply $B$, and
- $A \lor B$ is equivalent to $B \lor A$.

Yet, it may not be as obvious that $A \Rightarrow B$ and $\neg B \Rightarrow \neg A$ are equivalent, or that $(A \Rightarrow B) \land (A \Rightarrow \neg B)$ implies $\neg A$.\footnote{It is not uncommon for people to claim that $(A \Rightarrow B) \land (A \Rightarrow \neg B)$ is inconsistent, which is not correct.} For this reason, one needs a formal definition of logical properties and relationships, especially if one is to automate the decision of these notions.
1.2.1 Truth at a world

The semantics of propositional logic hinges on a key relationship between sentences and worlds. A world $\omega$ is a function that assigns a value of true/false to each propositional variable $P_i$, where $\omega(P_i)$ denotes the value assigned by world $\omega$ to variable $P_i$. Intuitively, a world represents a particular state of affairs in which the value of each propositional variable is known. The semantics of propositional logic is based on a simple definition which tells us whether a sentence $\alpha$ is true at a particular world $\omega$, which is written as $\omega \models \alpha$. This definition is given inductively as follows:

- $\omega \models P_i$ iff $\omega(P_i) = \text{true}$.
- $\omega \models \neg \alpha$ iff $\omega \not\models \alpha$.
- $\omega \models \alpha \lor \beta$ iff $\omega \models \alpha$ or $\omega \models \beta$.
- $\omega \models \alpha \land \beta$ iff $\omega \models \alpha$ and $\omega \models \beta$.

A world is also called a truth assignment or an interpretation. Moreover, the relationship $\omega \models \alpha$ has many alternative readings: $\omega$ satisfies $\alpha$; $\omega$ entails $\alpha$; $\alpha$ is satisfied by $\omega$; $\alpha$ holds at $\omega$; in addition to $\alpha$ is true at $\omega$. We should note here that deciding the truth of a sentence at a particular world is quite easy algorithmically as it can be done in time which is linear in the size of sentence.

1.2.2 Logical properties

We are now ready to define the most central logical property of sentences, that of consistency. Specifically, we say that sentence $\alpha$ is consistent iff there is at least one world $\omega$ at which $\alpha$ is true, otherwise it is inconsistent. It is also common to use the terms satisfiable/unsatisfiable instead of consistent/inconsistent, respectively.\(^2\) The property of satisfiability is quite important since many other logical notions can be reduced to satisfiability. Moreover, deciding the satisfiability of a propositional sentence is the first problem proven to be NP–complete.

Consider now a situation involving earthquakes, burglaries and alarms, leading to three propositional variables:

- Earthquake: An earthquake took place.
- Burglary: A burglary took place.
- Alarm: The alarm was triggered.

This set of propositional variables leads to eight possible worlds:

\(^2\)Hence, satisfiability is both a property and a relationship. As a property, satisfiability applies to sentences. And as a relationship, satisfiability relates a world to a sentence. Moreover, we have defined the satisfiability property in terms of the satisfiability relationship.
In general, a set of $n$ propositional variables leads to a total of $2^n$ possible worlds. Given the above set of worlds, we have the following:

Earthquake is true at $\omega_1, \ldots, \omega_4$.

$\neg$Earthquake is true at $\omega_5, \ldots, \omega_8$ (the complement of worlds $\omega_1, \ldots, \omega_4$).

$\neg$Burglary is true at $\omega_3, \omega_4, \omega_7, \omega_8$.

Alarm is true at $\omega_1, \omega_3, \omega_5, \omega_7$.

$\neg$($\text{Earthquake} \lor \text{Burglary}$) is true at $\omega_7, \omega_8$.

$\neg$($\text{Earthquake} \lor \text{Burglary}$) $\lor$ Alarm is true at $\omega_1, \omega_3, \omega_5, \omega_7, \omega_8$.

($\text{Earthquake} \lor \text{Burglary}$) $\Rightarrow$ Alarm is true at $\omega_1, \omega_3, \omega_5, \omega_7, \omega_8$.

$\neg$Burglary $\land$ Burglary is not true at any world, hence, it is not satisfiable.

The symbol false is often used to denote a sentence which is unsatisfiable. Note that we have also used false to denote one of the values that propositional variables can assume. The symbol false is therefore overloaded in propositional logic.

We now turn to another logical property, that of validity. Specifically, we say that sentence $\alpha$ is valid iff it is true at every world. For example, $A \lor \neg\ A$ is valid, and so is ($A \land (A \Rightarrow B)) \Rightarrow B$. It is not uncommon for one to confuse invalidity and inconsistency even though these are very distinct notions. If the sentence $\alpha$ is invalid, then one can identify a world $\omega$ at which $\alpha$ is false. That does not imply though that $\alpha$ is false at every other world, in which case $\alpha$ would be inconsistent. Hence, inconsistency is a stronger property than invalidity. Note, however, that a sentence is valid iff its negation is inconsistent. This follows because for every world $\omega$, either $\alpha$ is true at $\omega$, or $\neg\alpha$ is true at $\omega$, but not both. Hence, the worlds at which $\alpha$ is true and those at which $\neg\alpha$ is true form a partition of the whole set of worlds.

Going back to the above example, the following sentences are valid:

Earthquake $\lor$ $\neg$Earthquake.

Alarm $\Rightarrow$ (($\text{Earthquake} \lor \text{Burglary}) \Rightarrow \text{Alarm}$).
The symbol $true$ is often used to denote a sentence which is valid. Moreover, it is common to write $\models \alpha$ in order to indicate that sentence $\alpha$ is valid.

We close this section by defining a third logical property of sentences, that of completeness. Specifically, we say that a sentence is complete if it is true at exactly one world. Intuitively, a complete sentence is very affirmative as it communicates knowledge of the exact state of the world. Going back to our previous example, the sentence $\text{Earthquake} \lor \text{Burglary}$ is not complete as it is true at six possible worlds, $\omega_1, \ldots, \omega_6$. The sentence $\text{Earthquake} \land \neg \text{Burglary} \land \neg \text{Alarm}$ is complete though as it is true at exactly one world, $\omega_7$.

### 1.2.3 Logical relationships

A logical property applies to a single sentence, while a logical relationship applies to two or more sentences. We now define a few logical relationships among propositional sentences:

- Two sentences are equivalent iff they are true at the same set of worlds.
- Two sentences are mutually exclusive iff they are never true at the same world.
- Two sentences are exhaustive iff one of them is true at each world.

A fourth logical relationship which deserves special attention is that of implication. Intuitively, a sentence $\alpha$ implies another sentence $\beta$ if in every situation where $\alpha$ is true, $\beta$ must also be true. In propositional logic, situations are represented by worlds. Hence,

- Sentence $\alpha$ implies sentence $\beta$ iff whenever $\alpha$ is true at world $\omega$, then $\beta$ is also true at world $\omega$. That is, if $\omega \models \alpha$ then $\omega \models \beta$ for all worlds $\omega$.

We have earlier used the symbol $\models$ to denote the satisfiability relationship between a world and a sentence. Specifically, we wrote $\omega \models \alpha$ to indicate that world $\omega$ satisfies sentence $\alpha$. This symbol is also used to indicate implication between sentences, where we write $\alpha \models \beta$ to say that sentence $\alpha$ implies sentence $\beta$. As an example statement which mixes the two uses of this symbol, we provide again the definition of implication:

$$\alpha \models \beta \text{ iff } (\text{if } \omega \models \alpha \text{, then } \omega \models \beta \text{ for every world } \omega).$$

We observe again that either a sentence $\beta$ or its negation $\neg \beta$ must be true at each world $\omega$: $\omega \models \beta$ or $\omega \models \neg \beta$. This property does not hold for implication, however, as it is perfectly possible that we have $\alpha \not\models \beta$ and $\alpha \not\models \neg \beta$, that is, $\alpha$ neither implies $\beta$ nor implies $\neg \beta$. In fact, this property holds for implication only when the sentence $\alpha$ is complete (that is, $\alpha$ is true at exactly one world). In this case, we must have $\alpha \models \beta$ or $\alpha \models \neg \beta$ for every sentence $\beta$. We also note that the implication relation among sentences is also known as entailment. That is, we say that $\alpha$ entails $\beta$ to mean that $\alpha$ implies $\beta$.

---

3Again, we are overloading the symbol $true$ since it also denotes one of the values that a propositional variable may assume.
1.3 Knowledge as a set of possible worlds

The set of worlds that satisfy a sentence $\alpha$ are called the models of $\alpha$ and denoted by $\text{Mods}(\alpha) = \{ \omega : \omega \models \alpha \}$.

Using the definition of satisfaction ($\models$), it is not hard to prove that the function $\text{Mods}$ satisfies the following properties:

- $\text{Mods}(\alpha \land \beta) = \text{ Mods}(\alpha) \cap \text{ Mods}(\beta)$.
- $\text{Mods}(\alpha \lor \beta) = \text{ Mods}(\alpha) \cup \text{ Mods}(\beta)$.
- $\text{Mods}(\neg \alpha) = \text{ Mods}(\alpha)$.

The function $\text{Mods}$ is very useful. First, it allows us to define logical notions tersely. If $\Omega$ denotes the set of all worlds, then

- Sentence $\alpha$ is satisfiable iff $\text{ Mods}(\alpha) \neq \emptyset$.
- Sentence $\alpha$ is valid iff $\text{ Mods}(\alpha) = \{ \omega \}$.
- Sentence $\alpha$ is complete iff $\text{ Mods}(\alpha) = \{ \omega \}$.
- Sentence $\alpha$ implies sentence $\beta$ iff $\text{ Mods}(\alpha) \subseteq \text{ Mods}(\beta)$.
- Sentence $\alpha$ is equivalent to sentence $\beta$ iff $\text{ Mods}(\alpha) = \text{ Mods}(\beta)$.
- Two sentences $\alpha$ and $\beta$ are mutually exclusive iff $\text{ Mods}(\alpha) \cap \text{ Mods}(\beta) = \emptyset$.
- Two sentences $\alpha$ and $\beta$ are exhaustive iff $\text{ Mods}(\alpha) \cup \text{ Mods}(\beta) = \Omega$.

Second, the function $\text{Mods}$ plays an important role in providing insights into the semantics of propositional logic. The key concept here is to view one’s knowledge as a set of worlds that characterize all possibilities. In a complete state of ignorance, every world is a possibility and, hence, our knowledge consists in the set of all worlds $\Omega$. As we know more, some of these worlds are deemed as impossible and, hence, the set of possible worlds starts shrinking. Specifically, when someone communicates sentence $\alpha$ to us, they are telling us that only worlds in $\text{ Mods}(\alpha)$ are possible, and every world outside $\text{ Mods}(\alpha)$ is impossible. Therefore, our state of knowledge in this case is $\text{ Mods}(\alpha)$. If we learn $\beta$ in addition, our state of knowledge will be $\text{ Mods}(\alpha \land \beta) = \text{ Mods}(\alpha) \cap \text{ Mods}(\beta)$ since only these worlds are possible, and every other world is impossible. We reach a complete state of knowledge when we deem all worlds as impossible, except for exactly one world.

Let us illustrate this concept of knowledge as a set of possible worlds using the earthquake–burglary–alarm example that we introduced earlier. There are eight worlds in this example, corresponding to eight possible ways the world

---

4 This can be generalized to an arbitrary number of sentences as follows. Sentences $\alpha_1, \ldots, \alpha_n$ are mutually exclusive iff $\text{ Mods}(\alpha_i) \cap \text{ Mods}(\alpha_j) = \emptyset$ for $i \neq j$.

5 This can be generalized to an arbitrary number of sentences as follows. Sentences $\alpha_1, \ldots, \alpha_n$ are exhaustive iff $\text{ Mods}(\alpha_1) \cup \ldots \cup \text{ Mods}(\alpha_n) = \Omega$. 
could be. For example, the second world $\omega_2$ corresponds to the situation where we have an earthquake and a burglary, but the alarm did not go off. In a complete state of ignorance about this example, all of these eight situations are possible. Hence, our state of knowledge is the set of all eight worlds $\omega_1, \ldots, \omega_8$. Suppose now that someone communicates to us the following sentence,

$$\alpha : (\text{Earthquake} \lor \text{Burglary}) \Rightarrow \text{Alarm}.$$ 

By accepting $\alpha$, we are considering some of these eight worlds as impossible. In particular, any world that does not satisfy $\alpha$ is ruled out. Therefore, our state of knowledge now corresponds to the set,

$$\text{Mods}(\alpha) = \{\omega_1, \omega_3, \omega_5, \omega_7, \omega_8\}.$$ 

This is depicted in the following table, in which the sentence $\alpha$ can be viewed as having ruled out all worlds that are outside $\text{Mods}(\alpha)$:

<table>
<thead>
<tr>
<th>world</th>
<th>Earthquake</th>
<th>Burglary</th>
<th>Alarm</th>
<th>Possible?</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\omega_1$</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>yes</td>
</tr>
<tr>
<td>$\omega_2$</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>○</td>
</tr>
<tr>
<td>$\omega_3$</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>yes</td>
</tr>
<tr>
<td>$\omega_4$</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>○</td>
</tr>
<tr>
<td>$\omega_5$</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>yes</td>
</tr>
<tr>
<td>$\omega_6$</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>○</td>
</tr>
<tr>
<td>$\omega_7$</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>yes</td>
</tr>
<tr>
<td>$\omega_8$</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>yes</td>
</tr>
</tbody>
</table>

Suppose now that we also learn,

$$\beta : \text{Earthquake} \Rightarrow \text{Burglary},$$

for which $\text{Mods}(\beta) = \{\omega_1, \omega_2, \omega_5, \omega_6, \omega_7, \omega_8\}$. Our state of knowledge is now given by,

$$\text{Mods}(\alpha \land \beta) = \text{Mods}(\alpha) \cap \text{Mods}(\beta) = \{\omega_1, \omega_5, \omega_7, \omega_8\},$$

which is one less world than it used to be. This is depicted by the following table:

<table>
<thead>
<tr>
<th>world</th>
<th>Earthquake</th>
<th>Burglary</th>
<th>Alarm</th>
<th>Possible?</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\omega_1$</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>yes</td>
</tr>
<tr>
<td>$\omega_2$</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>○</td>
</tr>
<tr>
<td>$\omega_3$</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>○</td>
</tr>
<tr>
<td>$\omega_4$</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>○</td>
</tr>
<tr>
<td>$\omega_5$</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>yes</td>
</tr>
<tr>
<td>$\omega_6$</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>○</td>
</tr>
<tr>
<td>$\omega_7$</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>yes</td>
</tr>
<tr>
<td>$\omega_8$</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>yes</td>
</tr>
</tbody>
</table>
Table 1.1: Some equivalences among sentence schemas. The last two can be viewed as definitions of the $\Rightarrow$ and $\Leftrightarrow$ connectives.

According to the view of knowledge as a set of possible worlds, the more we know, the smaller the set of possible worlds becomes. Let us now recall the definition of implication. A knowledge base $\Delta$ implies a sentence $\beta$ iff $\text{Mods}(\Delta) \subseteq \text{Mods}(\beta)$. Hence, the more information we have in $\Delta$, the smaller the set $\text{Mods}(\Delta)$ is, and the more sentences $\beta$ are implied by $\Delta$. The extreme case is when the knowledge base $\Delta$ is unsatisfiable, $\text{Mods}(\Delta) = \emptyset$, leading to $\Delta \models \beta$ for every sentence $\beta$. Inconsistent knowledge bases are then useless since they imply everything, including a sentence and its negation. Note also that if $\beta$ is implied by $\Delta$, then adding $\beta$ to $\Delta$ does not change our state of knowledge since $\text{Mods}(\Delta \land \beta) = \text{Mods}(\Delta)$ in this case. Therefore, adding an implied sentence to a knowledge base does not change the informational content of that knowledge base.

### 1.4 Some equivalences among sentences

We now consider some equivalences between propositional sentences, which can be quite useful when working with propositional logic. The equivalences are given in Table 1.1 and are actually between schemas, which are templates that can generate a large number of specific sentences (instances). For example, $\alpha \Rightarrow \beta$ is a schema, generating instances such as $\neg A \Rightarrow (B \lor \neg C)$, where $\alpha$ is replaced by $\neg A$ and $\beta$ is replaced by $(B \lor \neg C)$.

It is worth noting here that the first two equivalences on Table 1.1 follow as a special case of the following more general rule:

- If $\alpha \models \beta$, then $\alpha \land \beta$ is equivalent to $\alpha$.

Similarly, the third and fourth equivalences in Table 1.1 follow from the following rule:
Figure 1.2: The relationship between the models of $\alpha$ and $\beta$ assuming that $\alpha$ implies $\beta$.

Table 1.2: Some reductions among logical relationships and properties.

<table>
<thead>
<tr>
<th>Relationship</th>
<th>Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$ implies $\beta$</td>
<td>$\alpha \land \neg \beta$ is unsatisfiable</td>
</tr>
<tr>
<td>$\alpha$ implies $\beta$</td>
<td>$\alpha \Rightarrow \beta$ is valid</td>
</tr>
<tr>
<td>$\alpha$ and $\beta$ are equivalent</td>
<td>$\alpha \Leftrightarrow \beta$ is valid</td>
</tr>
<tr>
<td>$\alpha$ and $\beta$ are mutually exclusive</td>
<td>$\alpha \land \beta$ is unsatisfiable</td>
</tr>
<tr>
<td>$\alpha$ and $\beta$ are exhaustive</td>
<td>$\alpha \lor \beta$ is valid</td>
</tr>
</tbody>
</table>

- If $\alpha \models \beta$, then $\alpha \lor \beta$ is equivalent to $\beta$.

These equivalences and others can be usually proven using the $\text{Mods}$ function and its properties. We will now provide an example proof of the following: If $\alpha \models \beta$, then $\alpha \land \beta$ is equivalent to $\alpha$.

Suppose that $\alpha \models \beta$. We then have $\text{Mods}(\alpha) \subseteq \text{Mods}(\beta)$, leading to the picture in Figure 1.2. It should be clear now that $\text{Mods}(\alpha) \cap \text{Mods}(\beta) = \text{Mods}(\alpha)$ and, hence, $\text{Mods}(\alpha \land \beta) = \text{Mods}(\alpha)$ by the properties of $\text{Mods}$. It then follows immediately that $\alpha \land \beta$ is equivalent to $\alpha$ by definition of equivalence.

## 1.5 Reductions among logical relationships and properties

One can state a number of reductions between logical properties and relationships, some of which are shown in Table 1.2. Specifically, this table shows how the relationships of implication, equivalence, mutual exclusiveness, and exhaustiveness can all be defined in terms of the properties of satisfiability and validity. These reductions are useful in practise. Suppose for example that we have an algorithm for testing satisfiability. We can then immediately use this algorithm to test whether $\alpha$ implies $\beta$ by testing whether $\alpha \land \neg \beta$ is satisfiable.
Figure 1.3: The relationship between the models of arbitrary sentence $\alpha$ and $\beta$. The shaded area represents the set of worlds $\text{Mods}(\alpha) \cap \text{Mods}(\beta)$.

One can also prove these reductions using the $\text{Mods}$ function and its properties. To illustrate this technique, we will next prove two of the reductions in Table 1.2, which are known as the Refutation and Deduction theorems.

1.5.1 The Refutation Theorem

The Refutation Theorem states that:

$\alpha$ implies $\beta$, $\alpha \models \beta$, iff $\alpha \land \neg \beta$ is inconsistent.

This theorem is quite commonly used in mathematical proofs under the label “proof by contradiction.” That is, to show that $\alpha$ implies $\beta$, we assume $\alpha$, assume $\neg \beta$, and then try to show a contradiction. When one is found, we declare that $\alpha$ implies $\beta$. This method of reducing the implication test to a satisfiability test is also predominant in automated reasoning systems as we shall discuss later.

To prove this theorem we recall that

$\alpha \models \beta$ iff $\text{Mods}(\alpha) \subseteq \text{Mods}(\beta)$, and

$\alpha \land \neg \beta$ is inconsistent iff $\text{Mods}(\alpha \land \neg \beta) = \emptyset$.

It suffices then to show that $\text{Mods}(\alpha) \subseteq \text{Mods}(\beta)$ iff $\text{Mods}(\alpha \land \neg \beta) = \emptyset$.

Consider now Figure 1.3 which depicts the relationship between $\text{Mods}(\alpha)$ and $\text{Mods}(\beta)$ for arbitrary sentences $\alpha$ and $\beta$. It should be clear from this figure that $\text{Mods}(\alpha) \subseteq \text{Mods}(\beta)$ iff the set $\text{Mods}(\alpha) \cap \text{Mods}(\beta) = \emptyset$. By properties of the $\text{Mods}$ function, we have $\text{Mods}(\alpha) \cap \text{Mods}(\beta) = \text{Mods}(\alpha \land \neg \beta)$. Hence, $\text{Mods}(\alpha) \subseteq \text{Mods}(\beta)$ iff $\text{Mods}(\alpha \land \neg \beta) = \emptyset$, which proves the theorem.

1.5.2 The Deduction Theorem

The Deduction Theorem states that:

$\alpha$ implies $\beta$, $\alpha \models \beta$, iff $\alpha \Rightarrow \beta$ is valid.
We will now prove this theorem assuming that we have accepted the Refutation theorem, and by showing that $\alpha \Rightarrow \beta$ is valid iff $\alpha \land \lnot \beta$ is inconsistent.

First, we recall that sentence $\alpha$ is valid iff its negation $\lnot \alpha$ is inconsistent:

$$
\text{Mods}(\alpha) = \Omega \text{ iff } \text{Mods}(\lnot \alpha) = \emptyset.
$$

This follows immediately from the properties of $\text{Mods}$ since $\text{Mods}(\lnot \alpha) = \text{Mods}(\alpha)$.

All we have to show now is that $\alpha \Rightarrow \beta$ is equivalent to the negation of $\alpha \land \lnot \beta$, which proves that $\alpha \Rightarrow \beta$ is valid iff $\alpha \land \lnot \beta$ is inconsistent:

$$
\lnot (\alpha \land \lnot \beta) \text{ is equivalent to } \lnot \alpha \lor \beta \text{ by de Morgan’s law and double negation.}
$$

$$
\lnot \alpha \lor \beta \text{ is equivalent to } \alpha \Rightarrow \beta \text{ by definition of implication.}
$$

This completes the proof.

### 1.6 Quantified propositional logic

One can define a few more operators on propositional logic sentences, which enrich the language and lead to what is known as quantified propositional logic.

**Conditioning**

The first of these operators is known as conditioning and allows one to eliminate a variable from a knowledge base, given that we know the value of that variables. Specifically, to condition a knowledge base $\Delta$ on a positive literal $P$ is to replace every occurrence of $P$ in $\Delta$ by $\text{true}$. Moreover, to condition a knowledge base $\Delta$ on a negative literal $\lnot P$ is to replace every occurrence of $P$ in $\Delta$ by $\text{false}$. The result of conditioning knowledge base $\Delta$ on literal $\alpha$ is denoted by $\Delta \mid \alpha$ and a key property of $\Delta \mid \alpha$ is that it does not mention the variable of literal $\alpha$.

Using the conditioning operator, one can define Boole’s expansion which has major applications in recursive algorithms that we discuss later. Given a knowledge base $\Delta$ and a variable $P$, Boole’s expansion allows us to re-write $\Delta$ as follows:

$$
P \land \Delta \mid P \lor \lnot P \land \Delta \mid \lnot P.
$$

The key property of this expansion is that both $\Delta \mid P$ and $\Delta \mid \lnot P$ do not mention variable $P$. Therefore, this expansion allows us to reduce a query with respect to a knowledge base $\Delta$ over $n$ variables into two queries on knowledge bases over $n-1$ variables. If Boole’s expansion is applied recursively to the newly created knowledge bases, it can then be used to decompose a complex knowledge base down to trivial knowledge bases that are equivalent to either $\text{true}$ or $\text{false}$. We will see many examples of such application in future chapters.

If the value of a variable is not known, we can still eliminate it from a knowledge base while preserving important properties of the original knowledge base. There are two methods for eliminating a variable in this case, which we discuss next.
Existential quantification

The first operator for eliminating a variable \( P \) from a knowledge \( \Delta \)—given that the value of \( P \) is unknown—is to existentially quantify variable \( P \) out of knowledge base \( \Delta \):

\[
\exists P \Delta \overset{\text{def}}{=} \Delta|P \lor \Delta|\neg P.
\]

Consider the knowledge base

\[
\Delta = \begin{cases} 
A & \Rightarrow B \\
B & \Rightarrow C,
\end{cases}
\]

which can be written as

\[
\Delta = \begin{cases} 
\neg A \lor B \\
\neg B \lor C.
\end{cases}
\]

We then have

\[
\Delta|B = (\neg A \lor \text{true}) \land (\text{true} \lor C) = C,
\]

and

\[
\Delta|\neg B = (\neg A \lor \text{false}) \land (\neg \text{false} \lor C) = \neg A,
\]

leading to

\[
\exists B \Delta = \neg A \lor C = A \Rightarrow C.
\]

Note here that \( \exists B \Delta \) is implied by \( \Delta \). This is true in general for any knowledge base and any variable:

\[
\Delta \models \exists P \Delta.
\]

We even have a stronger property for existential quantification. Let \( \Sigma \) be the set of all propositional variables. Then not only is \( \exists P \Delta \) implied by \( \Delta \), but it also contains all the information that \( \Delta \) contains about the variables \( \Sigma \setminus P \). That is, if \( \alpha \) is a sentence which is constructed using the variables \( \Sigma \setminus P \), then

\[
\Delta \models \alpha \iff \exists P \Delta \models \alpha.
\]

Therefore, one can interpret \( \exists P \Delta \) as the strongest sentence over variables \( \Sigma \setminus P \) that is implied by \( \Delta \). This is why \( \exists P \Delta \) is usually referred to as the forgetting of variable \( P \) from \( \Delta \), as it represents the result of erasing what \( \Delta \) says about variable \( P \) (without affecting what \( \Delta \) says about other variables). This is also why \( \exists P \Delta \) is sometimes referred to as the projection of \( \Delta \) on variables \( \Sigma \setminus P \), as it represents all that \( \Delta \) says above variables \( \Sigma \setminus P \).

Consider now the circuit in Figure 1.5, and the following knowledge base describing its behavior:

\[
\Delta = \begin{cases} 
A & \Rightarrow \neg X \\
\neg A & \Rightarrow X \\
A \land B & \Rightarrow Y \\
\neg (A \land B) & \Rightarrow \neg Y \\
X \lor Y & \Rightarrow C \\
\neg (X \lor Y) & \Rightarrow \neg C.
\end{cases}
\]
Using the definition of existential quantification, one can verify that
\[
\exists X \exists Y \Delta = \left\{ \begin{array}{l}
\neg A \lor B \Rightarrow C \\
A \land \neg B \Rightarrow \neg C,
\end{array} \right.
\]
which corresponds to a less informative description of the circuit behavior. Finally, one can verify that
\[
\exists X \exists Y \Delta | C = \neg A \lor B,
\]
which is even a less informative description of the given circuit as it only characterizes circuit input under which the circuit will produce a high output.

**Universal quantification**

The other method for eliminating a variable \( P \) from a propositional knowledge base \( \Delta \) is by universally quantifying \( P \) out of knowledge base \( \Delta \):
\[
\forall P \Delta \overset{\text{def}}{=} \Delta | P \land \Delta | \neg P.
\]
Universal quantification satisfies properties which are dual to those satisfied by existential quantification:
\[
\forall P \Delta \models \Delta.
\]
Moreover, if \( \alpha \) is a sentence constructed from variables \( \Sigma \setminus P \), then
\[
\alpha \models \Delta \iff \alpha \models \forall P \Delta.
\]
Therefore, one can interpret \( \forall P \Delta \) as the *weakest sentence* over variables \( \Sigma \setminus P \) that implies \( \Delta \).

**Duality**

Existential and universal quantifiers are related by the following dualities:
\[
\exists P \Delta = \neg (\forall P \neg \Delta),
\]
\[
\forall P \Delta = \neg (\exists P \neg \Delta).
\]
It is always possible to quantify over more than one variable and the order of quantification does not matter for the same quantifier. Hence, \( \exists P \exists Q \Delta \) is equivalent to \( \exists Q \exists P \Delta \) and similarly for universal quantification. We will therefore simply write \( \exists P, Q \Delta \) and \( \forall P, Q \Delta \) instead of \( \exists P \exists Q \Delta \) and \( \forall P \forall Q \Delta \), respectively, whenever convenient.
Semantics of quantified propositional logic

We have defined conditioning, existential quantification, and universal quantification as transformations on classical propositional sentences. Another way of defining these operators is by extending the satisfaction relationship between worlds and propositional sentences to cover sentences that include these operators. Specifically, let $\omega[P]$ denote a world which agrees with $\omega$ on every variable except possibly on $P$, where $\omega[P]$ sets $P$ to true. Similarly, let $\omega[\neg P]$ denote a world which agrees with $\omega$ on every variable except possibly on $P$, where $\omega[\neg P]$ sets $P$ to false. We then have:

- $\omega \models \alpha|P$ iff $\omega[P] \models \alpha$.
- $\omega \models \alpha|\neg P$ iff $\omega[\neg P] \models \alpha$.
- $\omega \models \exists P \alpha$ iff $\omega[P] \models \alpha$ or $\omega[\neg P] \models \alpha$.
- $\omega \models \forall P \alpha$ iff $\omega[P] \models \alpha$ and $\omega[\neg P] \models \alpha$.

Given the above extensions to the satisfiability relationship between worlds and sentences, one can immediately extend the function $\text{Mods}$ to sentences that include conditioning and quantification operators. All logical properties and relationships are then defined, therefore, providing formal semantics to quantified propositional logic.

1.7 Queries and transformations on KBs

We have defined the syntax and semantics of propositional logic in previous sections, which allowed us to formally define the representational aspects of our approach to automated reasoning. We will identify in this section a class of operations on knowledge bases that will constitute the building blocks from which we will compose more complex operations as we address applications in future chapters.

We will divide our operations into two classes: queries and transformations. Queries are operations that return information about a knowledge base, while transformations are operations that modify a knowledge base.

Queries

The set of queries we have identified are as follows, where $\Delta$ and $\Gamma$ are knowledge bases:

- **CO** Testing whether a $\Delta$ is satisfiable.
- **VA** Testing whether a $\Delta$ is valid.
- **CT** Counting the models of $\Delta$.
- **EN** Enumerating the models of $\Delta$.
IP  Testing whether \( \Delta \) implies a disjunction of literals (clause) \( \alpha \).

PI  Testing whether a conjunction of literals (term) \( \beta \) implies \( \Delta \).

IM  Testing whether \( \Delta \) implies \( \Gamma \).

EQ  Testing whether \( \Delta \) is equivalent to \( \Gamma \).

Probably the key observation about the above queries is that their difficulty depends on the syntactic form of given knowledge bases. For example, \textit{CO} is hard on arbitrary propositional sentences, but becomes easy if the sentence is a disjunction of terms, known as Disjunctive Normal Form (DNF). Moreover, validity is also quite difficult on arbitrary sentences, but becomes easy if the sentence is a conjunction of clauses, known as Conjunctive Normal Form (CNF).

Another observation about the above queries is that some are more general than others. For example, enumerating models is more general than counting these models as one can count models as a side effect of enumeration. Moreover, \textit{IM} is more general than \textit{IP} since clauses are a special case of knowledge bases. We make this distinction, however, since \textit{IM} is much more difficult than \textit{IP}, yet \textit{IP} is quite useful in many applications.

Transformations

We now turn to transformations of knowledge bases, where \( \Delta \) and \( \Gamma \) are knowledge bases:

\textbf{NEG}  Negating \( \Delta \).

\textbf{CON}  Conjoining \( \Delta \) and \( \Gamma \).

\textbf{DIS}  Disjoining \( \Delta \) and \( \Gamma \).

\textbf{CND}  Conditioning \( \Delta \) on a literal.

\textbf{EQT}  Existentially quantifying variable \( P \) out of \( \Delta \).

\textbf{FQT}  Universally quantifying variable \( P \) out of \( \Delta \).

Given the definition of these transformations, it may appear that they are easy to perform as they all can be implemented in time linear in the size of given KBs. This ceases to be true, however, once we restrict the syntactic form of KBs. For example, negating a KB is straightforward, but becomes difficult if we insist that both the original and negated KB are in CNF. We will in later chapters discuss a large number of syntactic forms and enumerate the difficulty of the above operations on each of these forms.
Putting the operations to use

To provide an example of how these queries and transformations can be put to use in implementing reasoning applications, let us consider a simple circuit with two cascaded inverters as given in Figure 1.4. The knowledge base $\Delta$ for this circuit is shown below:

$$
\Delta = \begin{cases} 
ok(X) \land A \Rightarrow \neg B, \\
ok(X) \land \neg A \Rightarrow B, \\
ok(Y) \land B \Rightarrow \neg C, \\
ok(Y) \land \neg B \Rightarrow C
\end{cases}.
$$

Suppose now that we observe the input $A$ to be high, the output $C$ to be low, and we are interested in knowing whether this is a normal device behavior. This can be decided by asking whether the extended KB, $\Delta \cup \{A, \neg C, \ ok(X), \ ok(Y)\}$ is satisfiable. The answer is no in this case, indicating that the observed behavior is abnormal. Hence, we used the satisfiability query to decide whether a particular device behavior is abnormal.

Let us now define a device state to be an assignment of $true/false$ to the health variables $ok(X)$ and $ok(Y)$. And suppose that our goal is to count the number of states that the circuit can be in, given that we have observed the input $A$ to be high, and the output $C$ to be low. One way to do this is using two more operations on the KB: existential quantification and model counting. Specifically, we existentially quantify variables $A$, $B$ and $C$ out of the KB $\Delta \cup \{A, \neg C\}$, leading to the new KB, $\{\neg ok(X) \lor \neg ok(Y)\}$, which characterizes the circuit health under given observation. We then count the models of this new
KB, which turns out to be 3. These models correspond to the possible states of given circuit and we may want to enumerate them, leading to the following: \{ok(X), \neg ok(Y)\}, \{\neg ok(X), ok(Y)\}, and \{\neg ok(X), \neg ok(Y)\}.

1.8 Boolean functions

The operations we defined in the previous section on propositional knowledge bases correspond to similar operations on Boolean functions. In fact, some of the more important results on these operations have originated in the literature on Boolean functions and then translated into the language of propositional logic. Understanding the connection between propositional knowledge bases and Boolean functions is therefore important for a comprehensive coverage of the literature on propositional reasoning. Moreover, some of the notions that exist in both literatures are more intuitively expressed in one representation than in the other. Hence, our ability to freely navigate between the two representations can aid our comprehension of such notions. Our goal in this section is then to provide an introduction to Boolean functions and how they relate to propositional knowledge bases.

1.8.1 Basics of Boolean functions

A Boolean variable is one which takes values in 1, 0. A Boolean function \(f\) over Boolean variables \(X_1, \ldots, X_n\) is a function that maps each value of these variables into 1 or 0. Boolean functions can be constructed using three operations on Boolean values:

- If \(b\) is a Boolean value, then \(\bar{b}\) is the complement of that value: \(\bar{b} = 1 - b\).
- If \(b_1\) and \(b_2\) are Boolean values, then \(b_1 \cdot b_2\) is the product of these values, and \(b_1 + b_2\) is the maximum of these values.

Using these operations, one can build complex Boolean functions from simpler ones. For example, if \(f(X_1, \ldots, X_n)\) and \(g(X_1, \ldots, X_n)\) are two Boolean functions, then \(f(X_1, \ldots, X_n), f(X_1, \ldots, X_n), g(X_1, \ldots, X_n), f(X_1, \ldots, X_n) + g(X_1, \ldots, X_n)\) are Boolean functions.

The conditioning and quantification operators have parallels on Boolean functions. Specifically, the conditioning operator corresponds to the restriction operator, where the restriction of function \(f(X_1, \ldots, X_n)\) to \(X_i = b\), denoted \(f|_{X_i = b}\), is the function \(f(X_1, \ldots, X_{i-1}, b, X_{i+1}, \ldots, X_n)\), which results from fixing the value of variable \(X_i\) to \(b\). Note that the restriction \(f|_{X_i = b}\) is a Boolean function over \(n - 1\) variables.

The Shannon expansion of Boolean function \(f(X_1, \ldots, X_n)\) with respect to variable \(X_i\) is given by the following identity:

\[ f = X_i \cdot f|_{X_i = 1} + \overline{X_i} \cdot f|_{X_i = 0}. \]
This expansion allows one to express a Boolean function over \( n \) variables in terms of two Boolean functions, each over \( n - 1 \) variables. This expansion parallels Boole’s expansion for propositional logic.

One can existentially quantify over variables in a Boolean function:

\[
\exists X_i f \overset{\text{def}}{=} f|_{X_i=1} + f|_{X_i=0}.
\]

The expression \( \exists X_i f(X_1, \ldots, X_n) \) is then a Boolean function over only \( n - 1 \) variables, \( X_1, \ldots, X_{i-1}, X_{i+1}, \ldots, X_n \).

A solution to a Boolean function \( f(X_1, \ldots, X_n) \) is an assignment of values to variables \( X_1, \ldots, X_n \) under which the function \( f \) evaluates to 1. Suppose now that \( g \) is the result of existentially quantifying variable \( X_i \) in function \( f \), \( g = \exists X_i f \); let \( \sigma \) be a solution to function \( g \). There are two ways in which the solution \( \sigma \) can be extended: by assigning value 1 to \( X_i \), leading to \( \sigma^+ \), or by assigning value 0 to \( X_i \), leading to \( \sigma^- \). By construction of \( g \), we are guaranteed that either \( \sigma^+ \) or \( \sigma^- \) must be a solution to function \( f \). That is, some extension of a solution to function \( g \) is guaranteed to be a solution to function \( f \).

One can also universally quantify over variables in a Boolean function:

\[
\forall X_i f \overset{\text{def}}{=} f|_{X_i=1} . f|_{X_i=0}.
\]

The expression \( \forall X_i f(X_1, \ldots, X_n) \) is also a Boolean function over only \( n - 1 \) variables, \( X_1, \ldots, X_{i-1}, X_{i+1}, \ldots, X_n \). Suppose now that \( g \) is the result of universally quantifying variable \( X_i \) in function \( f \), \( g = \forall X_i f \); let \( \sigma \) be a solution to function \( g \); and let \( \sigma^+ \) and \( \sigma^- \) be its two extensions with respect to variable \( X_i \). In this case, we are guaranteed that both \( \sigma^+ \) and \( \sigma^- \) are solutions to function \( f \). That is, every extension of a solution to function \( g \) is guaranteed to be a solution to function \( f \).

One can define notions such as satisfiability, validity, and implication on Boolean functions. For example, a Boolean function \( f \) is satisfiable, written \( f \neq 0 \), iff it has at least one solution; and \( f \) is valid, written \( f = 1 \), iff every assignment of values to its variables is a solution. A Boolean function \( f \) implies another Boolean function \( g \), written \( f \leq g \), iff every solution of \( f \) is also a solution of \( g \). Other logical notions can be defined similarly.

### 1.8.2 Boolean functions of digital circuits

Consider the digital circuit in Figure 1.5. One can define at least three types of Boolean functions with respect to this circuit (and any other similar circuit), that have different information contents.

A Type III Boolean function for a circuit is a function over the circuit inputs. There is one Type III function \( f_O \) for each circuit output \( O \). The function \( f_O \) evaluates to 1 under a particular assignment of values to circuit inputs iff the circuit output \( O \) assumes value 1 under that particular assignment. Since the circuit in Figure 1.5 has only one output, it then has only one Type III Boolean function:
Type II Boolean function for a circuit is a function over the circuit inputs and outputs. When evaluated under a particular assignment of values to inputs and outputs, the Boolean function returns 1 iff the circuit generates the given outputs under the given inputs. The Type II Boolean function for the circuit in Figure 1.5 is:

\[
\begin{array}{ccc}
A & B & f_C(A, B) \\
0 & 0 & 1 \\
0 & 1 & 1 \\
1 & 0 & 0 \\
1 & 1 & 1 \\
\end{array}
\]

Type I Boolean function for a circuit is a function over the circuit inputs, outputs, and all internal wires (that are neither inputs nor outputs). When evaluated under a particular assignment of values to wires, the Boolean function returns 1 iff the given assignment is compatible with the circuit behavior. The Type I Boolean function for the circuit in Figure 1.5 is a function \( h \) over variables \( A, B, C, X, Y \). It is shown partially below:

\[
\begin{array}{ccc}
A & B & C & g(A, B, C) \\
0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 \\
0 & 1 & 0 & 0 \\
0 & 1 & 1 & 1 \\
1 & 0 & 0 & 1 \\
1 & 0 & 1 & 0 \\
1 & 1 & 0 & 0 \\
1 & 1 & 1 & 1 \\
\end{array}
\]
We can always obtain Type III and Type II functions from a Type I function using existential quantification. For the above circuit, we have $g = \exists X \exists Y h$. Moreover, we have $f_C = \exists X \exists Y h|_{C=1}$. The Type III function $f_C$ can also be obtained from the Type II function using $f_C = g|_{C=1}$. These constructions are general and work for other circuits as well.

1.8.3 Knowledge bases and Boolean functions

There is a one-to-one correspondence between Boolean functions and equivalence classes of propositional knowledge bases. Consider the Boolean function $f(X_1, \ldots, X_n)$. A propositional knowledge base $\Delta$ over variables $X_1, \ldots, X_n$ corresponds to Boolean function $f$ iff the models of $\Delta$ correspond to the solutions of $f$. We also say in this case that $f$ is the characteristic function of $\Delta$.

Note that there are many propositional knowledge bases that correspond to a particular Boolean function, but all these knowledge bases must be equivalent. Consider the following Boolean function $f$ over variables $A, B$:

<table>
<thead>
<tr>
<th>$A$</th>
<th>$B$</th>
<th>$f(\sigma_i)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma_1$</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$\sigma_2$</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$\sigma_3$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$\sigma_4$</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

The solutions of this function are $\sigma_2, \sigma_3, \sigma_4$. Consider now the knowledge base $\Delta = \neg(A \land B)$ and its corresponding set of possible worlds:

<table>
<thead>
<tr>
<th>world</th>
<th>$A$</th>
<th>$B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\omega_1$</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>$\omega_2$</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>$\omega_3$</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>$\omega_4$</td>
<td>false</td>
<td>false</td>
</tr>
</tbody>
</table>

The knowledge base $\Delta$ corresponds to the Boolean function $f$ since $\text{Mods}(\Delta) = \omega_2, \omega_3, \omega_4$. The knowledge base $\Gamma = \neg A \lor \neg B$ also corresponds to function $f$ since $\Gamma$ is equivalent to $\Delta$.

The language of propositional logic can therefore be viewed as a specification language for Boolean functions, since each knowledge base $\Delta$ corresponds to a
unique Boolean function which we shall denote by \( f_\Delta \). Moreover, the tools for reasoning within propositional logic can be viewed as an apparatus for reasoning about Boolean functions.

### 1.9 Propositional logic with discrete variables

Standard propositional logic assumes that propositional variables have Boolean values \( true/false \). Hence, the statement \( P_i \) can be interpreted as a shorthand for \( P_i = true \). It is possible to generalize propositional logic by allowing discrete variables that can take any finite number of values, not just Boolean values. For example, we may extend the earthquake–burglary–alarm example so that variable Alarm is discrete and has three values off, low and high. In this case, the simplest statements one can write in the logic would be of the form \( P_i = v \), indicating that variable \( P_i \) takes on value \( v \). The new syntax is a simple extension of what we have given before:

- If \( P_i \) is a variable and \( v \) is one of its values, then \( P_i = v \) is a sentence.
- If \( \alpha \) and \( \beta \) are sentences, then \( \neg \alpha \), \( \alpha \land \beta \), and \( \alpha \lor \beta \) are also sentences.

We can also use \( P_i \) as a sentence, assuming that variable \( P_i \) has values \( true/false \), in which case the sentence is interpreted as a shorthand for \( P_i = true \). Going back to the earthquake–burglary–alarm example, and assuming that variables Earthquake and Burglary are Boolean, one may write the following sentences:

- Earthquake \( \Rightarrow \) Alarm = low
- Burglary \( \Rightarrow \) Alarm = high
- \( \neg \text{Earthquake} \land \neg \text{Burglary} \Rightarrow \) Alarm = off.

The semantics of this extended logic is based on a more general notion of a world, which is an assignment of values (not necessarily Boolean) to variables. Continuing with the earthquake–burglary–alarm example, we have twelve possible worlds:

<table>
<thead>
<tr>
<th>world</th>
<th>Earthquake</th>
<th>Burglary</th>
<th>Alarm</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \omega_1 )</td>
<td>true</td>
<td>true</td>
<td>high</td>
</tr>
<tr>
<td>( \omega_2 )</td>
<td>true</td>
<td>true</td>
<td>low</td>
</tr>
<tr>
<td>( \omega_3 )</td>
<td>true</td>
<td>true</td>
<td>off</td>
</tr>
<tr>
<td>( \omega_4 )</td>
<td>true</td>
<td>false</td>
<td>high</td>
</tr>
<tr>
<td>( \omega_5 )</td>
<td>true</td>
<td>false</td>
<td>low</td>
</tr>
<tr>
<td>( \omega_6 )</td>
<td>true</td>
<td>false</td>
<td>off</td>
</tr>
<tr>
<td>( \omega_7 )</td>
<td>false</td>
<td>true</td>
<td>high</td>
</tr>
<tr>
<td>( \omega_8 )</td>
<td>false</td>
<td>true</td>
<td>low</td>
</tr>
<tr>
<td>( \omega_9 )</td>
<td>false</td>
<td>true</td>
<td>off</td>
</tr>
<tr>
<td>( \omega_{10} )</td>
<td>false</td>
<td>false</td>
<td>high</td>
</tr>
<tr>
<td>( \omega_{11} )</td>
<td>false</td>
<td>false</td>
<td>low</td>
</tr>
<tr>
<td>( \omega_{12} )</td>
<td>false</td>
<td>false</td>
<td>off</td>
</tr>
</tbody>
</table>
One can also define $\omega[P_i=v]$ to be a world that agrees with $\omega$ on all variables, except possibly on $P$ where $\omega[P_i=v]$ assigns the value $v$ to $P_i$. We can then extend the satisfiability relationship $\models$ as follows:

- $\omega \models P_i = v$ iff $\omega(P_i) = v$.
- $\omega \models \neg\alpha$ iff $\omega \not\models \alpha$.
- $\omega \models \alpha \lor \beta$ iff $\omega \models \alpha$ or $\omega \models \beta$.
- $\omega \models \alpha \land \beta$ iff $\omega \models \alpha$ and $\omega \models \beta$.
- $\omega \models \alpha[P_i=v]$ iff $\omega[P_i=v] \models \alpha$.
- $\omega \models \exists P\alpha$ iff $\omega[P_i=v] \models \alpha$ for some value $v$.
- $\omega \models \forall P\alpha$ iff $\omega[P_i=v] \models \alpha$ for all value $v$.

Based on this relationship, other logical notions such as satisfiability, validity, implication and equivalence can be defined, therefore, providing formal semantics to propositional logic with discrete variables.