Introduction: What Is Modal Logic?

Strictly speaking, modal logic studies reasoning that involves the use of the expressions 'necessarily' and 'possibly'. The main idea is to introduce the symbols □ (necessarily) and ◊ (possibly) to a system of logic so that it is able to distinguish three different *modes* of assertion: □ A (A is necessary), A (A is true), and ◊ A (A is possible). Introducing these symbols (or operators) would seem to be essential if logic is to be applied to judging the accuracy of philosophical reasoning, for the concepts of necessity and possibility are ubiquitous in philosophical discourse.

However, at the very dawn of the invention of modal logics, it was recognized that necessity and possibility have kinships with many other philosophically important expressions. So the term 'modal logic' is also used more broadly to cover a whole family of logics with similar rules and a rich variety of different operators. To distinguish the narrow sense, some people use the term 'alethic logic' for logics of necessity and possibility. A list describing some of the better known of these logics follows.

System	Symbols	Expression Symbolized
Modal logic	□	It is necessary that
(or Alethic logic)	◊	It is possible that
Tense logic	G	It will always be the case that
	F	It will be the case that
	H	It has always been the case that
	P	It was the case that
Deontic logic	O	It is obligatory that
	P	It is permitted that
	F	It is forbidden that

Locative logic	Tx	It is the case at x that
Doxastic logic	Bx	x believes that
Epistemic logic	Kx	x knows that

This book will provide you with an introduction to all these logics, and it will help sketch out the relationships among the different systems. The variety found here might be somewhat bewildering, especially for the student who expects uniformity in logic. Even within the above subdivisions of modal logic, there may be many different systems. I hope to convince you that this variety is a source of strength and flexibility and makes for an interesting world well worth exploring.

1

The System K: A Foundation for Modal Logic

1.1. The Language of Propositional Modal Logic

We will begin our study of modal logic with a basic system called K in honor of the famous logician Saul *K*ripke. K serves as the foundation for a whole family of systems. Each member of the family results from strengthening K in some way. Each of these logics uses its own symbols for the expressions it governs. For example, modal (or alethic) logics use $\Box$ for necessity, tense logics use **H** for what has always been, and deontic logics use **O** for obligation. The rules of K characterize each of these symbols and many more. Instead of rewriting K rules for each of the distinct symbols of modal logic, it is better to present K using a generic operator. Since modal logics are the oldest and best known of those in the modal family, we will adopt $\Box$ for this purpose. So $\Box$ need not mean *necessarily* in what follows. It stands proxy for many different operators, with different meanings. In case the reading does not matter, you may simply call $\Box$A 'box A'.

First we need to explain what a *language for propositional modal logic* is. The symbols of the language are $\bot$, $\rightarrow$, $\Box$; the propositional variables: p, q, r, p′, and so forth; and parentheses. The symbol $\bot$ represents a contradiction, $\rightarrow$ represents 'if . . then', and $\Box$ is the modal operator. A *sentence of propositional modal logic* is defined as follows:

$\bot$ and any propositional variable is a sentence.
If A is a sentence, then $\Box$A is a sentence.
If A is a sentence and B is a sentence, then (A$\rightarrow$B) is a sentence.
No other symbol string is a sentence.

In this book, we will use letters 'A', 'B', 'C' for sentences. So A may be a propositional variable, p, or something more complex like (p→q), or ((p→ ⊥)→q). To avoid eyestrain, we usually drop the outermost set of parentheses. So we abbreviate (p→q) to p→q. (As an aside for those who are concerned about use-mention issues, here are the conventions of this book. We treat '⊥', '→', '□', and so forth as *used* to refer to symbols with similar shapes. It is also understood that '□A', for example, refers to the result of concatenating □ with the sentence A.)

The reader may be puzzled about why our language does not contain negation: ~ and the other familiar logical connectives: &, ∨, and ↔. Although these symbols are not in our language, they may be introduced as abbreviations by the following definitions:

(Def~) ~A $=_{df}$ A→⊥
(Def&) A&B $=_{df}$ ~(A→~B)
(Def∨) A∨B $=_{df}$ ~A→B
(Def↔) A↔B $=_{df}$ (A→B)&(B→A)

Sentences that contain symbols introduced by these definitions are understood as shorthand for sentences written entirely with → and ⊥. So, for example, ~p abbreviates p→⊥, and we may replace one of these with the other whenever we like. The same is true of complex sentences. For example, ~p&q is understood to be the abbreviation for (p→⊥)&q, which by (Def&) amounts to ~((p→⊥)→~q). Replacing the two occurrences of ~ in this sentence, we may express the result in the language of K as follows: ((p→⊥)→(q→⊥))→⊥. Of course, using such primitive notation is very cumbersome, so we will want to take advantage of the abbreviations as much as possible. Still, it simplifies much of what goes on in this book to assume that when the chips are down, all sentences are written with only the symbols ⊥, →, and □.

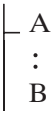
EXERCISE 1.1 Convert the following sentences into the primitive notation of K.
a) ~~p
b) ~p&~q
c) p∨(q&r)
d) ~(p∨q)
e) ~(p↔q)

Our use of ⊥ and the definition for negation (Def~) may be unfamiliar to you. However, it is not difficult to see why (Def~) works. Since ⊥ indicates a contradiction, ⊥ is always false. By the truth table for material implication, A→⊥ is true (T) iff either A is false (F) or ⊥ is T. But, as we said, ⊥ cannot be T. Therefore A→⊥ is T iff A is F. So the truth table for A→⊥ corresponds exactly to the truth table for negation.

The notion of an argument is fundamental to logic. In this book, an *argument* H / C is composed of a list of sentences H, which are called the *hypotheses,* and a sentence C called the *conclusion.* In the next section, we will introduce rules of proof for arguments. When argument H / C is provable (in some system), we write 'H ⊢ C'. Since there are many different systems in this book, and it may not be clear which system we have in mind, we subscript the name of the system S (thus: H $\vdash_S$ C) to make matters clear. According to these conventions, p, ~q→~p / q is the argument with hypotheses p and ~q→~p and conclusion q. The expression 'p, ~q→~p $\vdash_K$ q' indicates that the argument p, ~q→~p / q has a proof in the system K.

1.2. Natural Deduction Rules for Propositional Logic: PL

Let us begin the description of K by introducing a system of rules called PL (for *p*ropositional *l*ogic). We will use natural deduction rules in this book because they are especially convenient both for presenting and finding proofs. In general, natural deduction systems are distinguished by the fact that they allow the introduction of (provisional) assumptions or hypotheses, along with some mechanism (such as vertical lines or dependency lists) for keeping track of which steps of the proof depend on which hypotheses. Natural deduction systems typically include the rules Conditional Proof (also known as Conditional Introduction) and Indirect Proof (also known as Reductio ad Absurdum or Negation Introduction). We assume the reader is already familiar with some natural deduction system for propositional logic. In this book, we will use vertical lines to keep track of subproofs. The notation:



indicates that B has been proven from the hypothesis A. The dots indicate intervening steps, each of which follows from previous steps by one of the following five rules. The abbreviations for rule names to be used in proofs are given in parentheses.

The System PL

Hypothesis

```
|_ A
```

A new hypothesis A may be added to a proof at any time, as long as A begins a new subproof.

Modus Ponens

```
A
A→B
------
B        (MP)
```

This is the familiar rule Modus Ponens. It is understood that A, A→B, and B must all lie in *exactly the same subproof*.

Conditional Proof

```
|_ A
|  :
|  B
-------
A→B    (CP)
```

When a proof of B is derived from the hypothesis A, it follows that A→B, where A→B lies outside hypothesis A.

Double Negation

```
~~ A
--------
A    (DN)
```

The rule allows the removal of double negations. As with (MP), ~~A and A must lie in the same subproof.

Reiteration

```
A
:
|_ B
|  :
|-----
|  A    (Reit)
```

Sentence A may be copied into a new subproof. (In this case, into the subproof headed by B.)

These five rules comprise a system for propositional logic called PL. The rules say that if you have proven what appears above the dotted line, then

you may write down what appears below the dotted line. Note that in applying (MP) and (DN), *all sentences involved must lie in the same subproof.* Here is a sample proof of the argument p→q, ~q / ~p, to illustrate how we present proofs in this book.

```
| p→q
|- ~q
| q→⊥        (Def~)
|  |- p
|  | p→q     (Reit)
|  | q       (MP)
|  | q→⊥     (Reit)
|  | ⊥       (MP)
| p→⊥        (CP)
| ~p         (Def~)
```

The proof begins by placing the premises of the argument (namely, p→q and ~q) at the head of the outermost subproof. Then the conclusion (~p) is derived from these using the five rules of PL. Since there are no rules concerning the negation sign, it is necessary to use (Def~) to convert all occurrences of ~ into → and ⊥ as we have done in the third and last steps. We do not bother writing the name (Hyp) where we have used the hypothesis rule. That the (Hyp) rule is being used is already clear from the presence of the subproof bracket (the horizontal "diving board" at the head of a subproof).

Most books use line numbers in the justification of steps of a proof. Since we only have four rules, the use of line numbers is really not necessary. For example, when (CP) is used, the steps at issue must be the beginning and end of the preceding subproof; when (DN) is used to produce A, it is easy to locate the sentence ~~A to which it was applied; when (MP) is used to produce B, it is easy enough to find the steps A and A→B to which (MP) was applied. On occasion, we will number steps to highlight some parts of a proof under discussion, but step numbers will not be part of the official notation of proofs, and they are not required in the solutions to proof exercises.

Proofs in PL generally require many uses of Reiteration (Reit). That is because (MP) cannot be applied to A and A→B unless both of these

sentences lie in the same subproof. This constant use of (Reit) is annoying, especially in longer proofs, so we will adopt a convention to leave out the (Reit) steps where it is clear that an official proof could be constructed by adding them back in. According to this more relaxed policy, the proof just given may be abbreviated as follows:

```
| p→q
|_ ~q
| q→⊥    (Def~)
|  |_ p
|  | q    (MP)
|  | ⊥    (MP)
| p→⊥     (CP)
| ~p      (Def~)
```

We will say that an argument H / C is provable in PL (in symbols: H $\vdash_{PL}$ C) exactly when it is possible to fill in a subproof headed by members of H to obtain C.

```
|_H
|  :
|  C
```

It is possible to prove some sentences outside of any subproof. These sentences are called *theorems*. Here, for example, is a proof that p→(q→p) is a theorem.

```
|_ p
|  |_q
|  | p       (Reit)
| q→p        (CP)
p→(q→p)      (CP)
```

EXERCISE 1.2 Prove the following in PL.

a) p→q / (q→⊥)→(p→⊥)
b) p→q, p→(q→⊥) / p→⊥
c) Show (p→q)→(~q→~p) is a theorem of PL.

1.3. Derivable Rules of PL

PL is a complete system for propositional logic. Every valid argument written in the language of propositional logic has a proof in PL. However, proofs involving the abbreviations ~, &, ∨, and ↔ may be very complicated. The task of proof finding is immensely simplified by introducing derivable rules to govern the behavior of the defined connectives. (A rule is derivable in a system iff it can be proven in the system.) It is easy to show that the rule Indirect Proof (IP) is derivable in PL. Once this is established, we may use (IP) in the future, with the understanding that it abbreviates a sequence of steps using the original rules of PL.

Proof of Derivability:

~A		~A	
:		:	
⊥		⊥	
------		-------	
A	(IP)	~A→⊥	(CP)
		~~A	(Def~)
		A	(DN)

The (IP) rule has been stated at the left, and to the right we have indicated how the same result can be obtained using only the original rules of PL. Instead of using (IP) to obtain A, (CP) is used to obtain ~A→⊥. This by (Def~) is really ~~A, from which we obtain A by (DN). So whenever we use (IP), the same result can be obtained by the use of these three steps instead. It follows that adding (IP) to PL cannot change what is provable.

We may also show derivable a rule (⊥In) that says that ⊥ follows from a contradictory pair of sentences A, ~A.

Proof of Derivability:

A		A	
~A		~A	
-----		-----	
⊥	(⊥In)	A→⊥	(Def~)
		⊥	(MP)

Once (IP) and (⊥In) are available, two more variations on the rule of Indirect Proof may be shown derivable.

Proof of Derivability:

```
| ~A                    | ~A
|  :                    |  :
|  B                    |  B
|  :                    |  :
| ~B                    | ~B
-------                 --------
A        (~Out)         ⊥       (⊥In)
                        A       (IP)
```

EXERCISE 1.3 Show that the following variant of Indirect Proof is also derivable. (Feel free to appeal to (⊥In) and (IP), since they were previously shown derivable.)

```
| A
|  :
|  B
|  :
| ~B
-------
~A      (~In)
```

With (~Out) available it is easy to show the derivability of (~~In), a variant of Double Negation.

Proof of Derivability:

```
 A                  A
----                ----
~~A (~~In)          |~~~A
                    | ~A      (DN)
                    |  A      (Reit)
                    ~~A       (~Out)
```