Logics For Computer Science

Arindama Singh
Department of Mathematics
Indian Institute of Technology Madras
asingh@iitm.ac.in

Logics For Computer Science

Contents

Preface

Each chapter in this book has an introduction, the first section. But there is no introduction to the book itself. So, let me use the mask of the Preface for this purpose. Of course, a plain introduction to the book is just to place the book in your hand; but while you are reading it, you yourself have already done that. Well done!

This is the way I will be talking to you throughout. I will ask you to do exercises on the spot, often waiting for you up to some point, and then give you a hint to proceed. It is something like the following commercial for the book:

> I would like to ask you three questions, would you answer them with a plain 'Yes' or 'No'?
>
> Good, you have answered 'Yes', whatever be the reason. But see, that was my first question. Would you answer the same to the second as to the third?
>
> Very good, you have answered 'Yes' again; of course, it does not matter. If you have not bought a copy of this book, are you going to buy it soon?
>
> Look, if you have answered 'Yes' to the second question, you are also answering 'Yes' to the third, and if you have answered 'No' to the second, you are not answering 'No' to the third, i.e., your answer to the third is undoubtedly, 'Yes'. Excellent. That is the end of the commercial.

You have participated well in the commercial; I hope you will be with me throughout. It is easy to learn logic and easier to teach it; that is the spirit of this book. I would not reveal what is logic; you will discover it eventually. My aim is to equip you with the logical methods so that when you take up computer science as your profession, you will feel like a fish in water. A warning: though the book is ideal for self-learning, it would not replace a teacher. At least on one count: you can ask a question to your teacher and, hopefully, he[1] will give you an answer, or you will be inspired

[1] The masculine gender is used throughout the book not because of some chauvinistic reasons, as it is easy to do that. It is easy to write 'he' rather than 'she' or 'one' or 'a person' or 'a human being' etc. You need not be offended by it.

by him for finding your own answer; the book cannot do that always.

If you are a teacher, you may not need to learn all the topics, for you had probably learnt them. But you can really teach logic better by helping your students in the exercises. You can supplement the book with more computer science applications which I had to omit, for it is already a somewhat lengthy book. It is ideal for a two-semester course for not so advanced students. But you can cover it in one semester[2] by omitting certain proof procedures.

In Chapter 1, you will find the logic of statements. The approach is semantic. The semantic considerations give rise to the calculational method of proof quite naturally. Calculations have been used very informally for proving valid formulas just as you solve problems in school algebra. All the metaresults have been discussed except compactness. That is done via proof theory, again, for ease. In Chapter 2, a similar approach is taken for the logic of predicates or relations, more commonly known as the first order logic. The first order logic discussed here includes the equality relation, and no preference is given to one without equality, because applications demand it. This chapter also deals with the Herbrand's theorems without waiting for the resolution method. The resolution method is the topic of Chapter 3. Chapter 4 is exclusively meant for various proof techniques. In a one-semester course, you may go for only one proof method. Or, you may do one of the proof methods formally in detail and present others in one lecture each.

In Chapter 5, you come across a fundamental area of computer science, the *program verification.* It is also one of the most useful applications of first order logic. You should not stop just after that; you must hunt for some more materials and pursue the topic a bit further. You must have a similar approach to the model checking algorithms that have been introduced in Chapter 6 as an application of modal logics. This chapter gives a very comprehensive account of modal logics used in computer science. It does cover the essential ingredients leaving one: decidability issues, for which you may look through the References. Similarly, Chapter 7 introduces the so-called nonstandard logics which are upcoming, but not entirely new. We discuss only those nonstandard logics that have general frameworks yet; the list is not exhaustive. You are led through the chapters towards logic engineering, an activity that is becoming increasingly important in current research in computer science.

The approach of the book is mathematical, in the sense that it involves a continual interplay between the abstract and the concrete. The yearning for understanding the concrete phenomena gives rise to abstractions and to

[2]It takes around 60 lectures to complete the book. In fact, it is developed from my notes for a logic course offered to Engineering undergraduates at IIT Madras taking theoretical computer science as their minor subject. Some of the postgraduate students in Mathematics also credit the course as an elective.

understand the abstract, concrete phenomena are a necessity. The typical feeling in any concrete problem is that when you abstract it, an existence of a solution is an opaque mystery. The concrete problems do not come with a label that such and such method would be applicable. It is you who may think of applying an existing method to solve it. Often, patient and continuous involvement with the problems help you to gradually gain insight, leading to a solution. So, be patient and persistent, rely on yourself as far as possible; start the journey now.

I have to pause here for a second. I would like to thank all those who have made this happy journey of yours possible. Though the book has taken shape now, there has been a continual effort throughout. There must be many good reasons for this, the most basic of all these being my interest in higher studies. I thank my parents and my school teacher Mr. Rupakar Sarangi who nourished me physically and mentally by keeping me fit for higher studies. I thank my wife Archana for continuing in a similar vein after our marriage. She, along with my children, Anindya Ambuj and Ananya Asmita, did suffer a lot due to my post-marriage engagement with this book. I thank them for their patience and encouragement. I am indebted to my friend and logic tutor Professor Chinmoy Goswami of the University of Hyderabad. My students helped me considerably to learn logic; in a way, this book is a reward for that. I thank my colleague Dr. M. Thamban Nair of IIT Madras for encouraging me to bring out the book from some scratches on a note book. I also thank the Publishers, Prentice Hall of India, particularly, Mr. K. C. Devasia for his expert guidance and the production team for their assistance.

Any constructive suggestions for improving the contents would be most welcome. These may be emailed to me at asingh@iitm.ac.in.

Arindama Singh

1

Propositional Logic

1.1 Introduction

Logic is all about how an object *follows from* other objects. In Propositional logic, our objects are propositions such as

Bapuji was indeed a Mahatma.

The title of a book on logic could be mis-spelt.

Five men cannot have eleven eyes.

Buddha's original name was Arindama.

Alexander the great did not set foot in India.

The woman who committed the crime did not have three legs.

No bachelor is married.

Some unmarried men get married.

Propositions are declarative sentences which may be asserted to be true or false. It is quite possible that you may not be able to say for certain whether a given proposition is true (or false), without going to its meanings or external factors. But that does not really matter. All that we need is that it is either true or false. The so-called conjectures or open problems are propositions, the truth of which you do not know yet. For example, Goldbach's conjecture: "Every even number bigger than 2 is a sum of two prime numbers", or the $\mathcal{P} \neq \mathcal{NP}$: "There is at least one nondeterministic polynomial time solvable problem which is not deterministic polynomial time solvable". As of now, we do not have any way of showing the truth or falsity of these propositions. However, each of them is either true or false.

Note that we are not defining here what a proposition is. In fact, we will not even attempt to do so. We are only getting familiarized with the kind of objects in question.

The sentences which are not propositions include questions, orders, exclamations, etc., for which we may not like to associate a truth value. We do not know how to say whether "Are you OK?" is true or false. Similarly, we may not assert that "How beautiful is the morning sky!" is true.

Our building blocks here are propositions; we will not try to go beyond them. It is not our concern to determine whether 'each bachelor is married', for we do not know the meanings of the words uttered in the proposition. This is so because our units here are propositions and nothing less (or more). However, we seem to know that two propositions such as "I know logic" and "You know logic" can be composed to get another proposition such as "I and you know logic". We are only interested in propositions and how they are composed to yield other propositions. This is what we mean when we say that propositions are our building blocks. Thus we are interested in the forms rather than the meanings of propositions. Since propositions can be true or false, we must know how to assign truth values to the compound propositions.

If indeed I like logic and you like logic, then we must agree that the proposition "I and you like logic" is true. But what about the proposition "I like logic and you like logic or you do not like logic"? This is problematic, for we do not know exactly how this compound proposition has been composed of or formed. We do not know which way to parse it:

(I like logic and you like logic) or (you do not like logic)

OR

(I like logic) and (you like logic or you do not like logic)

We must use some such device, say, use of parentheses for disambiguating compound propositions. Further, we must know the connectives like 'and', 'or', as used above. It is enough to start with some commonly used connectives. If need arises, we will enrich our formalization later by adding new ones. Furthermore, we have the responsibility to say what 'follows from' means formally.

1.2 Syntax of PL

For any simple proposition, called a **propositional variable**, we will use any of the symbols $p_0, p_1, \ldots$ For **connectives** 'not', 'and', 'or', 'if ... then ...', 'if and only if', we use the symbols $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$, respectively; their names are negation, conjunction, disjunction, conditional, biconditional. We use the parentheses ')' and '(' as punctuation marks. We also have the special propositions $\top$ and $\bot$, called **propositional constants**, which stand for propositions which are 'true' and 'false', respectively. Both propositional variables and propositional constants are commonly called **atomic propositions**. So, the **alphabet** of Propositional Logic, **PL**, is the set

$$\{\,), (, \neg, \wedge, \vee, \rightarrow, \leftrightarrow, \top, \bot, p_0, p_1, p_2, \ldots\}$$

consisting of all these symbols. Any **expression** over this alphabet is a string of symbols such as

$$(\neg p_0 \rightarrow () \wedge p_1 \vee, \ \ \neg p_{100})(\rightarrow \vee, \ \ (\neg p_0 \rightarrow p_1)$$

Only the last one of these is a propositional formula or a proposition. In fact, we are interested only in such expressions. Now how to define them? Here are some of the ways of defining the set of all propositions, PROP.

Definition 1.1 The set of all propositions, PROP is the smallest set of expressions such that $\{\top, \bot, p_0, p_1, \ldots\} \subseteq$ PROP, and whenever $x, y \in$ PROP, the expressions $\neg x, (x \wedge y), (x \vee y), (x \rightarrow y), (x \leftrightarrow y)$ are also in PROP. The 'smallest' is in the sense of the relation $\subseteq$, i.e., A is smaller than B iff (if and only if) $A \subseteq B$.

The set PROP can also be defined inductively (or recursively).

Definition 1.2

$$\text{PROP}(0) = \{p_0, p_1, \ldots\} \cup \{\top, \bot\}$$

$$\text{PROP}(i+1) = \text{PROP}(i) \cup \{\neg x, (x \wedge y), (x \vee y), (x \rightarrow y), (x \leftrightarrow y) : x, y \in \text{PROP}(i)\}$$

$$\text{PROP} = \cup_{i \in \mathbb{N}} \text{PROP}(i)$$

Can you see that both the definitions above define the same set PROP? What are the sets PROP(i)?

PROP(0) contains all atomic propositions which do not contain any connective. PROP(1) contains all propositions having a single occurrence of a connective, along with those having no connectives. In general, PROP(i) contains all propositions having i or less number of occurrences of connectives. It is of course clear that any proposition has only a finite number of occurrences of connectives,and thus, it must be in some PROP(i). Now, can you prove it? What is there to prove? Well, if PROP(i) has been defined by Definition 1.2, PROP has been defined by Definition 1.1, and if PROP$'$ $= \cup_{i \in \mathbb{N}}$ PROP(i), then you must show that PROP $\subseteq$ PROP$'$ and PROP$'$ $\subseteq$ PROP. Do the following exercise before reading further.

***Exercise* 1.1** Show that PROP = PROP$'$.

We see that $\{\top, \bot, p_0, p_1, \ldots\} \subseteq$ PROP(0) $\subseteq$ PROP$'$. For $x, y \in$ PROP$'$, we also see that $x \in$ PROP(i) and $y \in$ PROP(j) for some $i, j \in \mathbb{N}$. Then, both $x, y \in$ PROP(k), where $k = \max(i, j)$. This shows that the expressions $\neg x, (x \wedge y), (x \vee y), (x \rightarrow y), (x \leftrightarrow y)$ are in PROP($k+1$). Since PROP($k+1$) $\subseteq$ PROP$'$, whenever $x, y \in$ PROP$'$, all of $\neg x, (x \wedge y), (x \vee y), (x \rightarrow y), (x \leftrightarrow y)$ are also in PROP$'$. Since PROP is the smallest set of expressions with this property, we have PROP $\subseteq$ PROP$'$.

To see the converse, it is enough to see that PROP(i) $\subseteq$ PROP for each $i \in \mathbb{N}$. This is proved by induction on i. To start with, we have already seen that PROP(0) $\subseteq$ PROP. Assume that (the induction hypothesis) PROP(i) $\subseteq$ PROP. Now let $w \in$ PROP($i+1$). By Definition 1.2, w is one of $\neg x, (x \wedge y), (x \vee y), (x \rightarrow y), (x \leftrightarrow y)$, for some $x, y \in$ PROP(i). Due to the induction hypothesis, $x, y \in$ PROP. By Definition 1.1, the expressions $\neg x, (x \wedge y), (x \vee y), (x \rightarrow y), (x \leftrightarrow y)$ are in PROP. That is, $w \in$ PROP, proving that PROP($i+1$) $\subseteq$ PROP. This completes the induction proof.

We summarize the above discussion in the following statement.

Theorem 1.1 *Definitions 1.1 and 1.2 are equivalent. If* $W_0 = \mathrm{PROP}(0)$ *and* $W_{i+1} = \mathrm{PROP}(i+1) - \mathrm{PROP}(i)$, *then* $\mathrm{PROP} = \cup_{i \in \mathbb{N}} W_i$.

It is of course easy to see the second part in the above theorem. But what are the W_i's? The set W_i is simply the set of all propositions having exactly i-number of occurrences of connectives. Thus the second part of the theorem says that each proposition has i occurrences of connectives for some natural number i, a triviality indeed. We have yet another way of defining propositions. See the following **formation rules** of propositions.

(a) $\top$ and $\bot$ are propositions.
(b) Each p_i is a proposition, where $i \in \mathbb{N}$.
(c) If x is a proposition, then $\neg x$ is a proposition.
(d) If x, y are propositions, then $(x \wedge y), (x \vee y), (x \to y), (x \leftrightarrow y)$ are propositions.
(e) Nothing else is a proposition unless it satisfies some or all of the rules (a–d).

This is another way of writing Definition 1.1. Note that the formation rule (e) simply states that "PROP is the smallest set that satisfies (a–d)". Such a rule, which does not add anything to the previous rules but only says that nothing else can be added, is called a *closure rule.*

The formation rules can be written in a more compact way, as **production rules**:

$$\begin{aligned}
\langle \text{ proposition } \rangle &\longmapsto \langle \top \rangle\\
\langle \text{ proposition } \rangle &\longmapsto \langle \bot \rangle\\
\langle \text{ proposition } \rangle &\longmapsto \langle \text{ propositional variable } \rangle\\
\langle \text{ proposition } \rangle &\longmapsto \neg\langle \text{ proposition } \rangle\\
\langle \text{ proposition } \rangle &\longmapsto (\langle \text{ proposition } \rangle \wedge \langle \text{ proposition } \rangle)\\
\langle \text{ proposition } \rangle &\longmapsto (\langle \text{ proposition } \rangle \vee \langle \text{ proposition } \rangle)\\
\langle \text{ proposition } \rangle &\longmapsto (\langle \text{ proposition } \rangle \to \langle \text{ proposition } \rangle)\\
\langle \text{ proposition } \rangle &\longmapsto (\langle \text{ proposition } \rangle \leftrightarrow \langle \text{ proposition } \rangle)
\end{aligned}$$

Read the symbol $\longmapsto$ as 'can be', and the token $\langle$ proposition $\rangle$ as 'a proposition'. A more compact way is given in the following definition.

Definition 1.3

$$w ::= \top \mid \bot \mid p \mid \neg w \mid (w \wedge w) \mid (w \vee w) \mid (w \to w) \mid (w \leftrightarrow w)$$

where p stands for any propositional variable and w stands for any **proposition** (or PL-formulas, also called *well-formed formulas* (**wff**). The language PL consists of all such propositions.

Here we write p for any generic propositional variable, w for any generic proposition, and the vertical bar '$\mid$' describes alternate possibilities (read it as 'or'). The same symbol w may be replaced by different propositions. That is what the word 'generic' means. This way of writing the grammatical rules is called the **Bacus-Naur form**.

A problem: given any string of symbols from the alphabet of PL, can you determine whether it is a proposition or not? Of course; it is not so difficult a grammar like that of English. You can always find by inspection, whether any string follows the above rules or not. For example, $p_1 \to p_2$ is not a proposition; you need a pair of parentheses; $(p_0 \wedge p_1 \to p_2)$ is also not a proposition. But that is not the question. The question is: can you give a procedure to do this for every possible expression?

To start with, you can see that $x = \neg((p_0 \wedge \neg p_1) \to (p_2 \vee (p_3 \leftrightarrow \neg p_4)))$ is a proposition. But if I ask you to show that it is indeed a proposition, what would you do? The key fact is that any object that has been formed (generated) by this grammar can also be parsed. That is, you can always find out the last rule that has been applied and then proceed backward. For the proposition x, the last rule applied was $w \longmapsto \neg w$. This means that x is a proposition if the expression $((p_0 \wedge \neg p_1) \to (p_2 \vee (p_3 \leftrightarrow \neg p_4)))$ is a proposition. This is so if the expressions $(p_0 \wedge \neg p_1), (p_2 \vee (p_3 \leftrightarrow \neg p_4))$ are propositions (due to the rule $w \longmapsto (w \to w)$). If you proceed, you would arrive at the **parse tree** given in Figure 1.1.

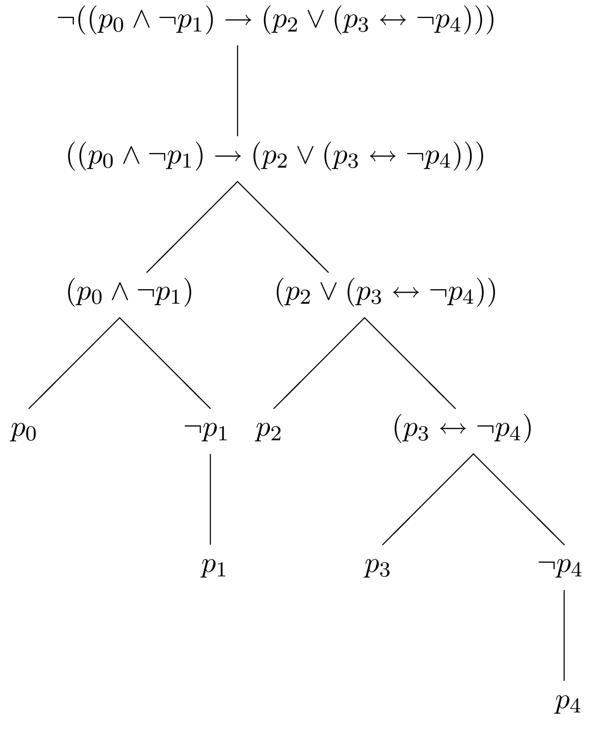

$\neg((p_0 \wedge \neg p_1) \to (p_2 \vee (p_3 \leftrightarrow \neg p_4)))$
$((p_0 \wedge \neg p_1) \to (p_2 \vee (p_3 \leftrightarrow \neg p_4)))$
$(p_0 \wedge \neg p_1)$
$(p_2 \vee (p_3 \leftrightarrow \neg p_4))$
p_0
$\neg p_1$
p_2
$(p_3 \leftrightarrow \neg p_4)$
p_1
p_3
$\neg p_4$
p_4

Figure 1.1 A parse tree.

The corresponding abbreviated tree is shown in Figure 1.2.

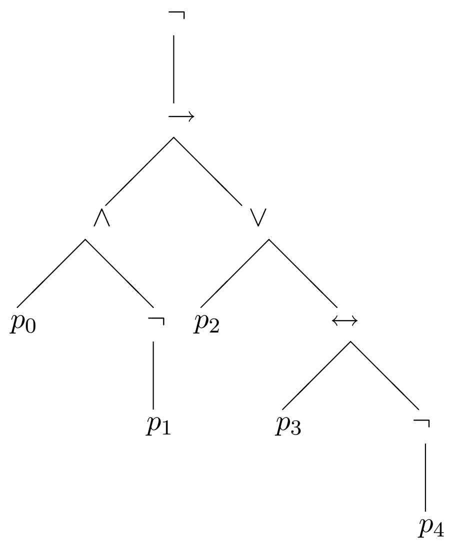


Figure 1.2 Abbreviated parse tree.

What is the parse tree for the expression $(\vee(p_1 \wedge p_2) \rightarrow (\neg p_1 \leftrightarrow p_2))$ and for $(\vee \rightarrow \neg p_1 \leftrightarrow p_2)$? See Figure 1.3. Remember, while constructing a parse tree you will be unfolding applications of appropriate production rules.

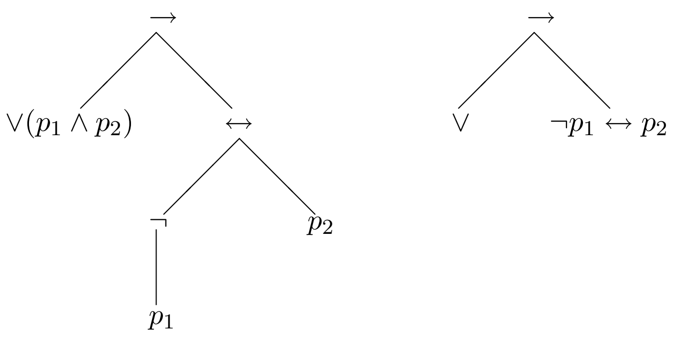


Figure 1.3 Parse trees.

We cannot parse $\vee(p_1 \wedge p_2)$ further, neither can we parse $\vee$, nor $\neg p_1 \leftrightarrow p_2$ since none of the production rules could have produced them. What is the difference between earlier parse trees and these? In the earlier trees, the leaves are atomic propositions, while the leaves of the trees in Figure 1.3 are not. The corresponding expressions in the latter cases are not propositions. Parse trees can be employed to decide whether a given expression is a proposition or not. But it still involves some intuition regarding the choice of an appropriate rule. Can you make it better? Look at the parse tree in Figure 1.2, say, the second leaf from the left, p_1. You can see that there is a subtree with parent node $\neg$ whose only child is p_1. This corresponds to the proposition $\neg p_1$. You can replace this subtree, say, by p_1. In the new tree (see T_1 in Figure 1.4), there is again a subtree with parent node $\wedge$, whose children are p_0 and p_1. This corresponds to the proposition $(p_0 \wedge p_1)$. You can replace this subtree, say by p_0. In the new tree (T_2), you have a leaf p_4 which is the sole child of the node $\neg$. So, first replace $\neg p_4$ by p_4.

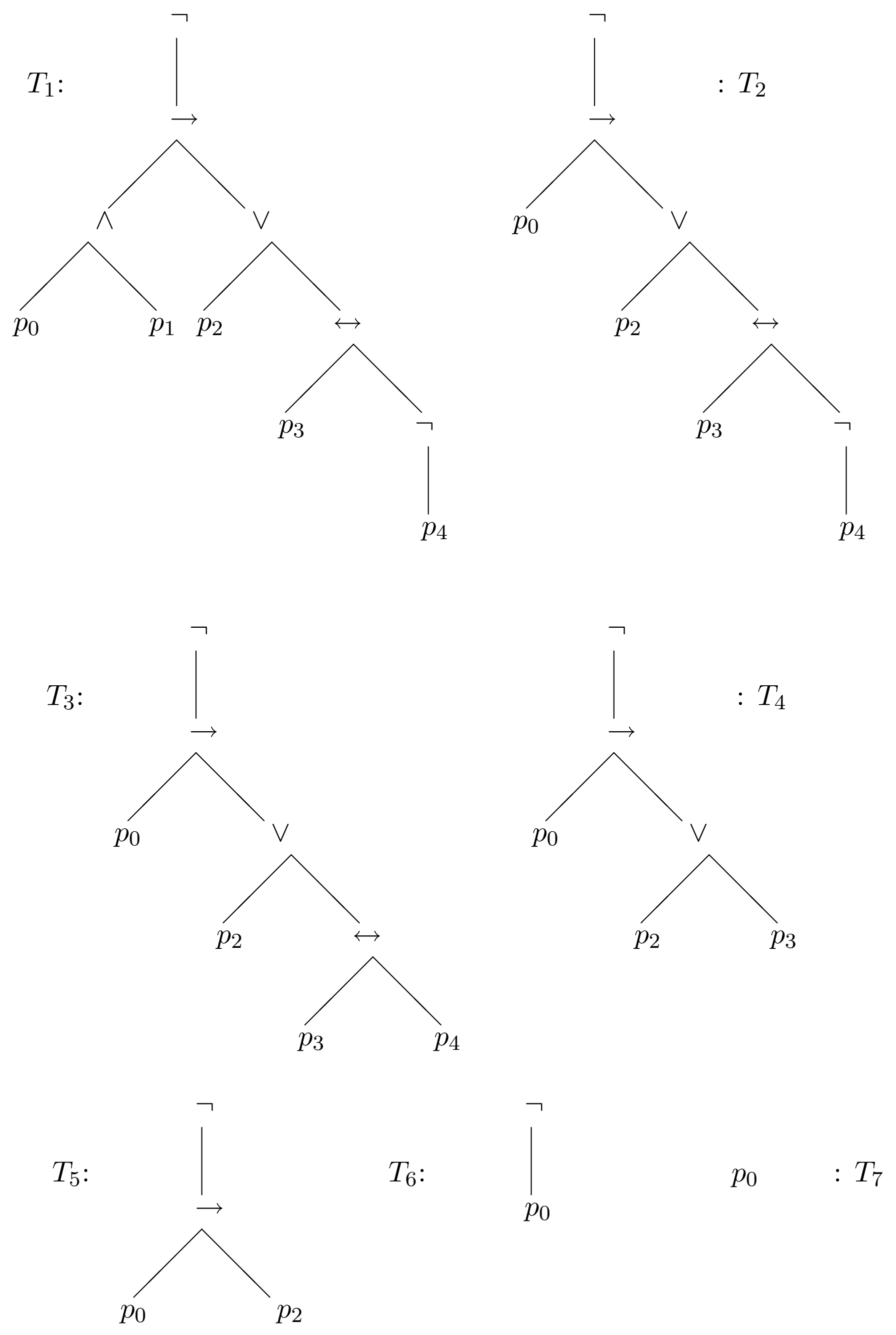


Figure 1.4 Sequence of parse trees.

Then replace the subtree corresponding to the proposition $(p_3 \leftrightarrow p_4)$, say, by p_3. As you continue this replacements, you get the sequence of trees (Figure 1.4): $T_0, T_1, T_2, T_3, T_4, T_5, T_6, T_7$, where the tree T_0 is the one in Figure 1.2.

In generating the sequence of trees in Figure 1.2, we had taken a subtree of depth 1, starting from the leaves upward, the leaves being chosen from left, whenever such a subtree exists. (This sentence may not be well understood, think and reword it.) You can of course choose the leaves and the subtrees of depth 1 in other ways. Try to do that for the tree in Figure 1.2. If you do that, you will be getting a different sequence of trees. But where does that terminate?

Now apply the same process as described in Figure 1.4 on the trees in Figure 1.3. You will get finally the trees in Figure 1.5.

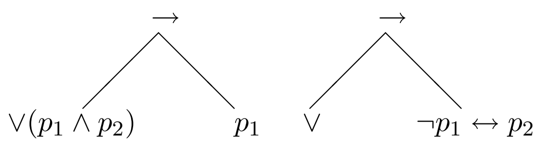


Figure 1.5 Computing with parse trees of Figure 1.3.

So, what do you observe? Computing with a parse tree, if you end up at an atomic proposition, then the original expression is a proposition, else, it is not. Can you think of a procedure to work with the given expression directly than going via a parse tree? Look at the expressions corresponding to the sequence of parse trees $T_0, \ldots, T_7$. Recollect T_0 stands for the tree in Figure 1.2. They are:

$$\neg((p_0 \wedge \neg p_1) \to (p_2 \vee (p_3 \leftrightarrow \neg p_4)))$$
$$\neg((p_0 \wedge p_1) \to (p_2 \vee (p_3 \leftrightarrow \neg p_4)))$$
$$\neg(p_0 \to (p_2 \vee (p_3 \leftrightarrow \neg p_4)))$$
$$\neg(p_0 \to (p_2 \vee (p_3 \leftrightarrow p_4)))$$
$$\neg(p_0 \to (p_2 \vee p_3))$$
$$\neg(p_0 \to p_2)$$
$$\neg p_0$$
$$p_0$$

***Exercise* 1.2** Construct parse trees for the following propositions in as many ways as possible:

(a) $((p_0 \to p_1) \vee (p_2 \to (\neg p_1 \wedge (p_0 \leftrightarrow \neg p_2))))$

(b) $(((p_5 \to (p_6 \vee p_8)) \leftrightarrow (p_3 \wedge \neg p_2)) \vee (\neg(p_1 \leftrightarrow p_3) \to p_{10}))$

***Exercise* 1.3** Looking at the parse trees in Figures 1.2 and 1.4, and the above sequence of expressions corresponding to the trees, write a procedure to determine whether a given expression is a proposition or not.

```
PROCEDURE : PropDet
    Input: Any expression $e$ over the alphabet of PL.
    Output: 'Yes', if $e$ is a proposition, else, 'No'.
    1. If $e$ is an atomic proposition, then print 'yes' and stop.
    2. Scan $e$ from left to get a subexpression $w$ of one of the forms $\neg x$,
       $(x \wedge y), (x \vee y), (x \to y), (x \leftrightarrow y)$, where $x, y$ are atomic propositions.
    3. If not found, then print 'no' and stop.
    4. If found, then replace $w$ by $x$.
    5. Go to Step 1.
```

***Exercise* 1.4** Prove by induction on the length of e that the procedure *PropDet* works correctly. [*Hint*: The length of an expression is the number of occurrences of symbols in it.]

What are the results of your work with Exercise 1.2? (If you have not yet done it, do it now; then proceed further.) Could you get two parse trees for (a)? For (b)? Why? It seems that if w is a proposition, there is one and only one parse tree corresponding to it. If a proposition w has been formed, then the way it has been formed is the only way it could have been formed. Each proposition can be read in a unique way. The set PROP is a freely generated set. The grammar of propositions (PL) is unambiguous. All these statements express the same fact stated in Theorem 1.2 below, referred to as **unique parsing**, **unique formation**, or **unique readability** theorem.

By now you must have noticed a nice property of propositions:

Property 1: Each proposition has the same number of left and right parentheses.

It is clear from the grammar we have adopted. But formally, we require a proof. How do you prove it?

***Exercise* 1.5** Prove by induction on the number of occurrences of connectives that each proposition has the same number of left and right parentheses.

Further, take a proposition, say,

$$w = \neg\neg(\top \vee ((p_2 \to (p_3 \leftrightarrow p_4)) \to (\bot \wedge (p_2 \wedge \neg p_3))))$$

The prefixes of this expression are:

$$\neg,\ \neg\neg,\ \neg\neg(,\ \neg\neg(\top,\ \neg\neg(\top\vee,\ \ldots,\ \neg\neg(\top \vee ((\cdots \neg p_3)))$$

To get the prefixes, you just start from its leftmost position, go to the right, and stop at some place. The proper prefixes are those excluding w itself, and excluding the empty string (having no symbols at all), which we denote by ϵ. You can see here that none of these proper prefixes has the same number of left and right parentheses; parentheses do not match, leaving of course the first two prefixes $\neg$, and $\neg\neg$. We have thus the second property.

Property 2: If u is a prefix of a proposition w, then $u = \epsilon$, or $u = w$, or u is a sequence of $\neg$'s, or the number of left parentheses is more than the number of right parentheses in u.

***Exercise* 1.6** Prove Property 2 by induction on the number of occurrences of connectives in w.

It is now clear from Properties 1 and 2 that if u is a proper prefix of a proposition w, then u is not a proposition. This statement is often expressed elegantly as

Property 3: If u and w are propositions and u is a prefix of w, then $u = w$.

You can prove Property 3 directly by induction (On what?). We use these properties to prove the following statement.

Theorem 1.2 (Unique Parsing) *Each proposition is parsed in exactly one way; it has a unique parse tree.*

Proof Let w be a proposition. We prove the theorem by induction on $\nu(w)$, the number of occurrences of connectives in w.

If $\nu(w) = 0$, then $w \in \{\top, \bot, p_0, p_1, \ldots\}$, in which case, its parse tree has only one node containing w itself, and it is obviously unique.

Lay out the induction hypothesis that for any proposition u, if $\nu(u) \leq n$, then it has a unique parse tree. Suppose that w is a proposition with $\nu(w) = n + 1$. By the grammar of propositions, $w = \neg x$ or $w = (x * y)$, where $* \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$, x, y being some propositions. In the first case, $\nu(x) = n$. By hypothesis, x has a unique parse tree T_x. Then w has the unique parse tree $\neg$ —T_x, written vertically, i.e., with root node as $\neg$ and then a vertical bar, and below it is appended the tree T_x as its only child. Essentially it amounts to showing that if $\neg u = \neg v$, then $u = v$.

Now, if $w = (x * y)$, then similarly the parse tree will have root node as $*$ and then appended to it are two trees T_x and T_y as left and right children, respectively. But why is it unique? Because, both the trees T_x and T_y are unique; is it so? But then the uniqueness will break down if we have two other propositions, say v, y such that $w = (v \circ z)$. It needs to be shown that w cannot be written in a different way. That is, we have to show that if $(x * y) = (v \circ z)$ for propositions x, y, v, z, then $x = v, y = z$, and $* = \circ$. In such a case, comparing the strings $(x * y)$ and $(v \circ z)$, we see that $x * y) = v \circ z)$. Then the proposition x is either a prefix of the proposition v or v is a prefix of x. (Just match from the left and see that this happens; alternatively, prove it by induction on the lengths of the strings). By Property 3, we have $x = v$. We are left with the string equality $*y) = \circ z)$. Their first symbols match, giving $* = \circ$. Their last symbols match; delete them. Then we are left with $y = z$. ∎

Parse trees, being unique for propositions, can be used for defining many concepts. For example, a **subformula** or a **subproposition** of a proposition w can be thought of as a proposition corresponding to any subtree of the parse tree of w. An **immediate subformula** or an **immediate subproposition** of a proposition w is any proposition corresponding to a subtree of the parse tree T_w of w whose depth is one less than that of T_w. Of course, you can define these concepts directly from w. A subproposition of w is any proposition that is a substring of w, and an immediate subproposition of $\neg x$ is x, while the immediate subpropositions of $(x * y)$ are x and y for any connective $* \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$.

Yet one more use of the uniqueness of parse trees is that functions from the set of atomic propositions can be extended uniquely to functions on PROP by compositionality. For example, suppose that f is a function with domain as AP, the set of atomic propositions, and co-domain as a set S. Suppose further that we have five operations $+_1, +_2, +_3, +_4, +_5$ defined on S, where $+_1$ is unary and all the others are binary. That is, for any $s, t \in S$, we know what are $+_1 s, s +_2 t, s +_3 t, s +_4 t, s +_5 t$, these being again members of S. Then we can associate $\neg$ with $+_1$, $\wedge$ with $+_2$, etc., and extend the function f by this association. That is, since $f(p_0), f(p_1)$ are already defined, we can have $f(\neg p_0) = +_1 f(p_0)$. Similarly, $f(p_0 \wedge p_1) = f(p_0) +_2 f(p_1) \ldots$ We will make use of this idea in the next section while giving meaning to the propositions.

After understanding what parsing is all about, we put some conventions so that writing propositions will become easier. First, we drop the subscripts i from p_i's. Instead, we write propositional variables, propositions just as $p, q, r, s, t, \ldots$, if no confusion arises. We use less number of parentheses by putting down some **precedence rules**. Remember the precedence rule that multiplication has more precedence than addition? This means that the expression $x \times y + z$ is to be rewritten as $((x \times y) + z)$, and not as $((x \times (y + z))$. Our precedence rules are the following:

$\neg$ has the highest precedence.

$\wedge, \vee$ have the next precedence, their precedence being equal.

$\rightarrow, \leftrightarrow$ have the lowest precedence, their precedence being equal.

This will not eliminate the use of parentheses altogether, but fewer will be required. Further, we will omit the outer parentheses. Using the precedence rules, the proposition $((p \vee (q \wedge s)) \rightarrow (t \leftrightarrow \neg p))$ can be abbreviated to $p \vee (q \wedge s) \rightarrow (t \leftrightarrow \neg p)$.

***Exercise* 1.7** Insert parentheses at appropriate places to make the following abbreviated propositions members of PROP:

(a) $(p \rightarrow q) \wedge \neg(r \vee q \leftrightarrow p) \leftrightarrow (\neg p \vee q \rightarrow r)$
(b) $(p \rightarrow q) \leftrightarrow (r \rightarrow t \vee p) \wedge (p \vee q \rightarrow \neg p \wedge t)$
(c) $p \vee (\neg q \leftrightarrow r \wedge p) \leftrightarrow (p \vee p \rightarrow \neg q)$

1.3 Semantics of PL

Syntax of a logic defines the primary objects we are interested in. It says how our objects look like. Semantics supplies some sort of meaning to these objects. PL, propositional logic, is the simplest of all logics we talk about in this book. Obviously, it has the simplest semantics. Each proposition in PROP has a meaning as a truth value. We do not go to the real meaning of any proposition, even when it is written as an English sentence, except that it may be either true or false. This is the concept of meaning of propositions here. We also use the notion of compositionality in giving meanings to propositions. In a way, we are going to interpret all our propositions in our small set $\{0, 1\}$. This set is sometimes written as BOOL after the logician George Boole. Note that, due to unique parsing, it is enough to start with giving meaning to the atomic propositions. Given that atomic propositions are given some meanings (as truth values of 0 or 1, with 0 for 'false' and 1 for 'true'), we must say how to take care of the connectives. For example, if $p \wedge q$ is a proposition, its truth value will be determined by the truth values of p and q, once we specify how to deal with $\wedge$.

Since $\wedge$ is a formal version of the English 'and', we hope $p \wedge q$ will be true when both p and q are true. We can employ **truth tables** (see Table 1.1) to take care of compositionality.

Table 1.1 Truth Values

$\top$
1

$\bot$
0

p	$\neg p$
1	0
0	1

p	q	$p \wedge q$	$p \vee q$	$p \to q$	$p \leftrightarrow q$
1	1	1	1	1	1
1	0	0	1	0	0
0	1	0	1	1	0
0	0	0	0	1	1

That is, the propositional constant $\top$ is assigned the truth value 1 (for true) and $\bot$ is assigned 0 (for false). The connective $\neg$ reverses the truth values. The connectives $\wedge, \vee, \leftrightarrow$ are just translations of the English 'and', 'or', and 'if and only if'. The connective $\to$ is a bit problematic. The table above says that the only way a proposition $p \to q$ can be false is when p is true and q is false. Given this condition, no doubt that $p \to q$ will be false. It is the phrase 'the only way' that is problematic. However, it is not so counter intuitive as the following illustration shows.

Your friend asks you whether you have got an umbrella, and you answer, "If I have got an umbrella, then I would not have been wet".

Suppose, you do not have an umbrella. Then is your statement true or false? Certainly it is not false if you had been really wet. It is also not false even if you had not been wet since you were at home when it was raining, etc. That is, the statement is not false whenever its antecedent "I have got an umbrella" is false. Since each sentence is either true or false, the sentence 'if ... then ...' is true, if its antecedent is false.

Consider one more situation. Your friend promises you to buy you the book *Logics for Computer Science* provided you help him in solving some problems. Certainly he had not broken his promise, when he bought you the book in spite of the fact that you did not actually help him solve those problems. That is, the statement is not false, when the consequent 'I will buy you the book' is true.

Both these instances are taken care of in the truth values for the connective $\to$. This connective is referred to as 'material implication' due to the vacuous assignment of 'true' when the antecedent is 'false'. Of course, this is the view of an implication adopted in main stream mathematics.

Given any proposition, having many occurrences of connectives in it, it is a matter of patience to construct its truth table in accordance with Table 1.1. If you know the truth values of propositional variables in it, then you may be able to determine the truth value of the proposition. If no specific truth values for the propositional variables are given, then also by taking all possibilities (as in Table 1.1), you may proceed to find all possible truth values of the given proposition. The truth table for the proposition $u = \neg(p \wedge q) \to (p \vee (r \leftrightarrow \neg q))$ is given in Table 1.2.

Table 1.2 A Truth Table

p	q	r	$\neg q$	$p \wedge q$	$\neg(p \wedge q)$	$r \leftrightarrow \neg q$	$p \vee (r \leftrightarrow \neg q)$	u
0	0	0	1	0	1	0	0	0
1	0	0	1	0	1	0	1	1
0	1	0	0	0	1	1	1	1
1	1	0	0	1	0	1	1	1
0	0	1	1	0	1	1	1	1
1	0	1	1	0	1	1	1	1
0	1	1	0	0	1	0	0	0
1	1	1	0	1	0	0	1	1

The truth tables can also be constructed from the parse trees. For example, the parse tree for $u = \neg(p \wedge q) \to (p \vee (r \leftrightarrow \neg q))$ is the left tree in Figure 1.6. For the assignment of 0 to p, 1 to q, and 0 to r (see the third line of Table 1.2), we write the truth values to the right of the corresponding leaves. Then we travel upwards in the tree for evaluating the connectives following the truth table rules. The evaluation is given on the right tree of Figure 1.6. Work it out.

Look at the last column of Table 1.2 and try the evaluation of u in the parse tree. Note that the evaluation is unambiguous due to the uniqueness of parsing. Now, you can have a formal definition of truth assignments and evaluations of propositions under such assignments.

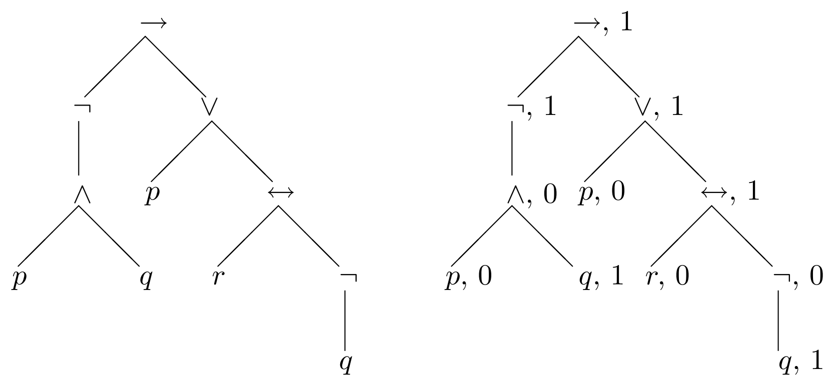


Figure 1.6 Evaluation in a parse tree.

***Exercise* 1.8** For each of the following propositions, construct the truth table and draw its parse tree. See how a row in the truth table evaluates the proposition v, w via the corresponding parse tree.

(a) $v = \neg(p \wedge q) \vee r \to \neg p \vee (\neg q \vee r)$,

(b) $w = (p \wedge q \to r) \leftrightarrow p \wedge (q \wedge \neg r)$.

Definition 1.4 A **truth assignment** is any partial function $t : AT \to \{0,1\}$, where $AT = \{\top, \bot, p_0, p_1, \ldots\}$, the set of atomic propositions with $t(\top) = 1$, and $t(\bot) = 0$. A **truth valuation** is any partial function which is an extension of a truth assignment to the set PROP of all propositions satisfying the following properties:

(a) $v(\neg x) = 1$ if $v(x) = 0$, else, $v(\neg x) = 1$,

(b) $v(x \wedge y) = 1$ if $v(x) = v(y) = 1$, else, $v(x \wedge y) = 0$,

(c) $v(x \vee y) = 0$ if $v(x) = v(y) = 0$, else, $v(x \vee y) = 1$,

(d) $v(x \to y) = 0$ if $v(x) = 1, v(y) = 0$, else, $v(x \to y) = 1$,

(e) $v(x \leftrightarrow y) = 1$ if $v(x) = v(y)$, else, $v(x \leftrightarrow y) = 0$,

for any propositions $x, y \in$ PROP.

***Exercise* 1.9** Construct truth tables for each of the following propositions. See how each row in the truth tables represents a valuation.

(a) $(p \to (\neg p \to p)) \to (p \to (p \to \neg p))$

(b) $(p \to q) \to ((p \to \neg q) \to \neg p)$

A valuation is also called a **Boolean valuation**. It is a precise recursive definition of the idea of a truth table. Can you see this? However, there is something awkward about this definition as an alternative to truth tables. For example, while computing the truth value of $\neg p_0$ in a truth table, we consider only two possibilities of assigning p_0 to 0 or 1, whereas there are infinitely many truth assignments with $t(p_0) = 0$. To give an example, take for each $i \geq 1$, t_i with $t_i(p_0) = 0, t_i(p_i) = 1$ and $t_i(p_j) = 1$ whenever $i \neq j$.

Of course, it is irrelevant what a truth assignment assigns to propositional variables which do not occur in a proposition. On the other hand, for a valuation to work for a proposition, the underlying truth assignment must have been defined for all the propositional variables occurring in it. For example, if v is a valuation extended from a truth assignment t which has been defined only for the atomic propositions $\bot, p_0, p_1$, then $v(p_0 \wedge p_2 \to \bot)$ has no meaning. Such a valuation, which is appropriate for a proposition, is called an interpretation of the proposition. The underlying truth assignment is also called an interpretation. It will be clear from the context, whether we mean an interpretation to be a truth assignment or a valuation. In most cases, it will not really matter whatever way you take it.

Suppose that you have an interpretation of a proposition $A = (p \vee q)$ and you have another interpretation of the proposition $B = (q \vee r)$. How do you talk about an interpretation of the formula $A \wedge B$? The interpretation of A may not assign any truth value to r, and similarly, the interpretation of B may not assign a truth value to p, though both the interpretations assign the same truth value to the common propositional variable q. Such eventualities are tackled by introducing a propositional language. A **propositional language** has an alphabet which is a subset of that of PL including all the connectives, the punctuation marks and the constants $\top$ and $\bot$. In such a case, the connectives,the punctuation marks, and the symbols $\top, \bot$ are called the **logical constants** and the propositional variables are called the **nonlogical constants**. A propositional language is characterized by the nonlogical constants it uses. For example, in this case, we would start with a propositional language having the nonlogical constants as p, q, r. Then an interpretation is said to be an interpretation of a propositional language instead of a proposition. When we have another language containing all the logical constants of a given language, then any interpretation of the bigger language is again considered as an interpretation of the smaller one. We do not use this terminology here, but you must be able to read and understand this terminology if used elsewhere.

Definition 1.5 Let w be a proposition. An **interpretation** of w is a valuation whose domain includes all the atomic propositions occurring in w. An interpretation is also called a state, a situation, or a place.

Intuitively, the truth value of a proposition under an interpretation should not depend upon the propositional variables that do not occur in it. Can you show this?

Theorem 1.3 (Relevance Lemma) *Let w be a proposition and A_w be the set of all atomic subpropositions of w. Let s, t be two interpretations of w. If $s(p) = t(p)$ for every $p \in A_w$, then $s(w) = t(w)$.*

Proof Let w, A_w, s, t be as in the above statement with $s(p) = t(p)$ for every $p \in A_w$. If w is atomic, then $A_w = \{w\}$, and $s(w) = t(w)$. This proves the basis step of induction. Lay out the induction hypothesis that for all propositions having number of occurrences of connectives fewer than n, the statement holds. If w has n occurrences of connectives, then either $w = \neg x$ or $w = (x * y)$ for unique propositions x, y and a unique connective $* \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$, due to unique parsing. Then? ∎

***Exercise* 1.10** Complete the proof of the relevance lemma.

The relevance lemma shows that the truth tables method of evaluation of a proposition has been correctly formalized by Definitions 1.4−1.5. It also says that an interpretation of a proposition is simply a row in the truth table of the proposition. Look at the pattern of values for u in Table 1.2 and for v, w in Exercise 1.8. You notice that the truth values of u are sometimes 0 and sometimes 1, while those of v are all 1, and of w are all 0. Such propositions are called contingent, tautology (or valid propositions), and contradiction (or unsatisfiable propositions), respectively. It is one of the aims of semantics to filter the tautologies out of the whole lot of propositions. In order to do that we will increase our vocabulary slightly. It will help us in defining formally these terms.

Definition 1.6 Let w be a proposition. A **model** of w is an interpretation i of w such that $i(w) = 1$. The fact that i is a model of w is written as $i \models w$ (read as i verifies w or i satisfies w). The fact that i is not a model of w is written as $i \nvDash w$ (read as, i does not satisfy, does not verify, or falsifies w).

In Table 1.2, let i be the interpretation as given in the first row. That is, $i(p) = i(q) = i(r) = 0$. The table says that $i \nvDash u, i \models v$ and $i \nvDash w$. The interpretation j defined by $j(p) = 1, j(q) = j(r) = 0$ is a model of u. Which line in Table 1.2 is the interpretation j?

The concepts of an interpretation and of a model can be presented in at least three different ways. One way is to view the connectives as **truth functions**, i.e., functions which map 0's and 1's to 0's and 1's. In this view, $\neg$ is taken as a function; $\neg : \{0,1\} \to \{0,1\}$ with $\neg(0) = 1$ and $\neg(1) = 0$. Similarly, $\wedge : \{0,1\} \times \{0,1\} \to \{0,1\}$ with $\wedge(0,0) = \wedge(0,1) = \wedge(1,0) = 0$ and $\wedge(1,1) = 1$.

***Exercise* 1.11** Define $\vee, \rightarrow, \leftrightarrow$ as truth functions.

***Exercise* 1.12** How many functions are there from $\{0,1\}$ to $\{0,1\}$? From $\{0,1\} \times \{0,1\}$ to $\{0,1\}$? Functions of the first type are called unary truth functions, and of the second type are called binary truth functions. How many n-ary truth functions are there?

In this view, any proposition can be thought of as a function (a circuit or a gate as is called by Electrical Engineers) that maps a truth assignment to $\{0,1\}$. For example, if i assigns p to 0 and q to 1, then the formula $p \wedge \neg q$ takes i to 0. A formalization of this view would look like:

$$p(i) = i(p),\ \neg x(i) = \neg(x(i)),\ (x*y)(i) = *(x(i), y(i)) \text{ for } * \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}.$$

In this language, a truth assignment is simply a mapping of voltages on a wire, when the wires represent propositional variables.

***Exercise* 1.13** What do $\top$ and $\bot$ represent?

***Exercise* 1.14** Let w be any proposition. Show by induction on the number of occurrences of connectives in w that for any truth assignment i in the above view, $i \models w$ iff $w(i) = 1$.

In another alternative presentation of semantics, a proposition is seen as a set of truth assignments. Writing $\mathcal{M}(x)$ for the set of models of a proposition x, the driving idea is to define $\mathcal{M}(x)$ in an alternative way. Due to compositionality, the whole idea rests on the sets of models of atomic propositions. Denote by $\mathcal{T}$ the set of all possible truth assignments, by i any truth assignment in $\mathcal{T}$, by p any propositional variable, and by x, y any proposition(s). Then the sets of models is defined to satisfy the following properties:

$$\mathcal{M}(\top) = \mathcal{T},\ \mathcal{M}(\bot) = \emptyset$$

$$\mathcal{M}(p) = \{i : i(p) = 1\}$$

$$\mathcal{M}(\neg x) = \mathcal{T} - \mathcal{M}(x)$$

$$\mathcal{M}(x \wedge y) = \mathcal{M}(x) \cap \mathcal{M}(y)$$

$$\mathcal{M}(x \vee y) = \mathcal{M}(x) \cup \mathcal{M}(y)$$

$$\mathcal{M}(x \to y) = (\mathcal{T} - \mathcal{M}(x)) \cup \mathcal{M}(y)$$

$$\mathcal{M}(x \leftrightarrow y) = (\mathcal{M}(x) \cap \mathcal{M}(y)) \cup ((\mathcal{T} - \mathcal{M}(x)) \cup (\mathcal{T} - \mathcal{M}(y)))$$

***Exercise* 1.15** Let w be any proposition. For any interpretation i, show that $i \models w$ iff $i \in \mathcal{M}(w)$. [*Hint*: Use induction on the number of occurrences of connectives in w.]

In the third alternative view, one tries to see each model as a set of **literals**, i.e., atomic propositions or their negations. For example, all models of the proposition $p \vee q$ are the truth assignments i, j, k, where $i(p) = i(q) = 1$; $j(p) = 0, j(q) = 1$; and $k(p) = 1, k(q) = 0$. Then the models i, j, k, in this view, correspond to the sets $\{p, q\}, \{\neg p, q\}, \{p, \neg q\}$, respectively. In general, this view rests on the observation that each assignment $i : AT \to \{0, 1\}$ can be thought of as a subset of the set of literals. This association is done by the rule:

For a truth assignment i, $\mathcal{M}_i = \{p : i(p) = 1\} \cup \{\neg p : i(p) = 0\}$.

To understand this view, do the following exercise.

***Exercise* 1.16** Show that corresponding to each interpretation i of a proposition w, the set $\mathcal{M}_i$ is unique. Further, let A_w be the set of all propositional variables occurring in w, and let M be any subset of $A_w \cup \{\neg p : p \in A_w\}$ such that for no $p \in A_w$, both p and $\neg p$ are in M. Show that there is a unique interpretation i of w such that $M = \mathcal{M}_i$.

There is also a slight variation in this view, which assigns each interpretation i of a proposition to a subset of atomic propositions only, negations of them being removed. It rests on the convention that whichever atomic subproposition is absent in such a set is in fact negated. For example, suppose that w is a proposition having the only atomic subpropositions p, q. If $M_i = \{p, \neg q\}$, then write $M_i' = \{p\}$. Then the primed ones are regarded as interpretations instead of the unprimed ones; they are only subsets of the atomic propositions. The truth assignment is constructed from such a subset of atomic propositions by its characteristic function. If $w = (p \rightarrow q) \vee (r \leftrightarrow s \vee t)$, then $A_w = \{p, q, r, s, t\}$ and the interpretation i with $i(p) = i(r) = i(s) = 1$, $i(q) = i(t) = 0$ is represented by the set $\{p, r, s\}$. Note that the characteristic function of the set $\{p, r, s\}$ as a subset of A_w is simply i. Remember that our goal was to filter out the tautologies from the whole lot of propositions by using the idea of models.

Definition 1.7 Let w be a proposition. w is **valid**, written as $\models w$, if each interpretation of w is its model. w is **satisfiable** if it has a model, i.e., if there is an interpretation i of w such that $i \models w$. If w is not valid, it is called **invalid**, and we write $\not\models w$. If w is not satisfiable, it is called **unsatisfiable**. Valid propositions are also called **tautologies**, and unsatisfiable propositions are called **contradictions**. The propositions which are both invalid and satisfiable are called **contingent**.

From Table 1.2, it is clear that u is satisfiable as $i \models u$, where i is the interpretation with $i(p) = 1, i(q) = i(r) = 0$. The proposition u is invalid as $j \not\models u$, where $j(p) = j(q) = j(r) = 0$. In Exercise 1.8, v is valid ($\models v$) as each interpretation is a model (evaluates it to 1). w is unsatisfiable as no interpretation is its model (each evaluates it to 0). v is also satisfiable as it has a model, e.g., $i \models v$. Similarly, w is invalid as it has a nonmodel, e.g., $i \not\models w$. In general, each valid proposition is satisfiable and each unsatisfiable proposition is invalid. A proposition which is neither satisfiable nor invalid is called contingent. Contingent propositions have some information content. In a certain context, if you affirm that a certain contingent proposition holds, then you are really giving some information.

***Exercise* 1.17** Define a Boolean valuation of any proposition w starting from a truth assignment given as a subset of A_w and then define valid and satisfiable propositions.

Since we are concerned with consequences, we must know when a proposition follows from a given set of propositions, and when a proposition is equivalent to another.

Definition 1.8 Let Σ be a set of propositions and u, v, w be propositions. An interpretation i is called a model of Σ, written $i \models \Sigma$, if i is a model of every proposition in Σ. The set Σ is called **satisfiable** if Σ has a model. Σ **semantically entails** w, written as $\Sigma \models w$, if each model of Σ is a model of w. Propositions u and v are equivalent, written as $u \equiv v$ if every model of u is a model of v and every model of v is also a model of u. $\Sigma \models w$ is also read as "the consequence $\Sigma \models w$ is valid", or as "w follows from Σ". For a consequence $\Sigma \models w$, the propositions in Σ are called the premises or hypotheses, and w is called the conclusion.

***Exercise* 1.18** Is $p \to (q \to r) \equiv (p \to q) \to r$?

If Σ is a singleton, say, $\Sigma = \{x\}$, instead of writing $\{x\} \models w$, we simply write $x \models w$. When $\Sigma = \{w_1, \ldots, w_n\}$, we also write the consequence $\Sigma \models w$ as $w_1, \ldots, w_n \models w$. Thus, $u \equiv v$ iff $u \models v$ and $v \models u$. The above definition says that w does not follow from Σ iff there exists an interpretation which verifies each premise in Σ but it falsifies w. Moreover, if $\Sigma = \emptyset$, then $\Sigma \models w$ means that $\models w$ since, vacuously, each interpretation verifies every premise here (there is none).

Can you see the overuse of the symbol $\models$? There are three ways we are using this symbol, one for satisfaction ($i \models w$), one for validity ($\models w$), and one for entailment ($\Sigma \models w$). If you take the third view of semantics, where a truth assignment is taken as a subset of atomic propositions, then all the three uses fall in place. (Why?)

We have almost completed in describing the logic called PL, or propositional logic. It concerns with the syntactic entities as the propositions in PROP, and the semantics as described above.

***EXAMPLE* 1.1** Show that the following argument is valid: If the band performs, then the hall will be full provided that the tickets are not too costly. However, if the band performs, the tickets will not be too costly. Therefore, if the band performs, then the hall will be full.

Solution We translate the above argument into a consequence in PL. A guideline in such a translation is to identify simple declarative sentences as atomic propositions. This identification sometimes requires to determine the connectives, or the words which look like connectives. The words 'and', 'or' and the phrases 'it is not the case that', 'if $\cdots$ then', etc. can be identified quickly. The words like 'unless', 'until', and phrases like 'either $\cdots$ or' etc. can be translated to PL by suitable connectives or as truth functions. For example, due to absence of temporality in PL, 'until' and 'unless' will mean the same (in PL), and 'unless' will simply be represented by $\vee$ (why?). Similarly, 'provided that' is simply 'if', the reverse of $\to$. With this brief guideline, let us try symbolizing the above argument.

p: the band performs

q: the hall is (will be) full

r: tickets are not too costly

Then the hypotheses are the propositions $p \to (r \to q),\ p \to r$, and the conclusion is $p \to q$. Thus we are asked to see that the consequence $\{p \to (r \to q),\ p \to r\} \models p \to q$ is valid. So, how to proceed?

According to Definition 1.8, you have to consider all possible models of the set of premises, and then see whether each of them verifies the conclusion or not. And how to find all the models of the set of premises? Well, construct a truth table. Since there are only three atomic propositions, by the relevance lemma, we consider only those truth assignments that involve these propositions. That is, there are only $2^3 = 8$ truth assignments in all that we may have to consider; these are given in the first three columns of Table 1.3.

Table 1.3 Truth Table for Example 1.1

p	q	r	$p \to r$	$r \to q$	$p \to (r \to q)$	$p \to q$
0	0	0	1	1	1	1
1	0	0	0	1	1	0
0	1	0	1	1	1	1
1	1	0	0	1	1	1
0	0	1	1	0	1	1
1	0	1	1	0	0	0
0	1	1	1	1	1	1
1	1	1	1	1	1	1

For the time being do not read the column for $p \to q$ in Table 1.3. Now you must find out all (common) models of both $p \to (r \to q)$ and $p \to r$. They are in rows 1,3,5,7, and 8. In order that the argument is correct, you must check whether $p \to q$ is true (evaluated to 1) in all these rows. This is the case. Hence the argument is correct.

Note that you need not evaluate $p \to q$ in the rows 2,4,6 since it does not matter whether $p \to q$ receives the truth value 0 or 1 in these cases. But if one of the rows $1, 3, 5, 7, 8$ had 0 for $p \to q$, then the consequence would not have been valid. This gives an alternative way in showing that $\Sigma \models w$. The alternative way is to search for an interpretation where $i(w) = 0$ but $i(x) = 1$ for each $x \in \Sigma$. If the search fails, then $\Sigma \models w$. If the search succeeds in finding one such interpretation, then $\Sigma \not\models w$.

To apply this method on Example 1.1, look at Table 1.3. The interpretations which assign 0 to $p \to q$ are in rows 2 and 6. Row 2 assigns 0 to $p \to r$ and row 6 assigns 0 to $p \to (r \to q)$, that is, whenever the conclusion is falsified, at least one of the premises is also falsified. Hence the consequence is valid. This method is sometimes easier to apply, and we write it as a theorem below. (See the connection of this method to the statement, called *reductio ad absurdum* or RAA.)

Theorem 1.4 (RAA) *Let* Σ *be a set of propositions and* w *be any proposition. Then* $\Sigma \models w$ *iff* $\Sigma \cup \{\neg w\}$ *is unsatisfiable.*

Proof Let $\Sigma \models w$. Let i be any interpretation. If i is a model of Σ, then as $\Sigma \models w, i(\neg w) = 0$. If $i \not\models \Sigma$, then $i \not\models x$ for some $x \in \Sigma$;

hence $i \not\models \Sigma \cup \{\neg w\}$. On the other hand, if i is not a model of Σ, then $i \not\models \Sigma \cup \{\neg w\}$. In any case, $i \not\models \Sigma \cup \{\neg w\}$. That is, $\Sigma \cup \{\neg w\}$ is unsatisfiable.

Conversely, let $\Sigma \cup \{\neg w\}$ be unsatisfiable. Let $i \models \Sigma$. Then $i(\neg w) = 0$, i.e., $i \models w$. Therefore, $\Sigma \models w$. ∎

Another important observation is contained in the following theorem. This states a very common principle followed in almost every proof in mathematics. It says that to prove $p \to q$, it is enough to assume the truth of p and then conclude the truth of q.

Theorem 1.5 (Deduction Theorem) *Let Σ be a set of propositions and x, y be propositions. Then $\Sigma \models x \to y$ iff $\Sigma \cup \{x\} \models y$.*

Proof Suppose that $\Sigma \models x \to y$. Let i be a model of $\Sigma \cup \{x\}$. Then $i \models \Sigma$, and also $i \models x$. Since $\Sigma \models x \to y$, we have $i \models x \to y$. If $i(y) = 0$, then $i(x) = 0$ contradicting $i \models x$. Hence, $i \models y$. Thus we have shown that any model of $\Sigma \cup \{x\}$ is also a model of y. That is, $\Sigma \cup \{x\} \models y$.

Conversely, suppose that $\Sigma \cup \{x\} \models y$. Let $i \models \Sigma$. If $i \not\models x \to y$, then $i(x) = 1$ and $i(y) = 0$. But then $i \models \Sigma \cup \{x\}$ and $i \not\models y$. This contradicts the assumption $\Sigma \cup \{x\} \models y$. Hence, $i \models x \to y$. Thus we have shown that any model of Σ has to be a model of $x \to y$, proving $\Sigma \models x \to y$. ∎

***EXAMPLE* 1.2** Use Deduction Theorem to show that the argument in Example 1.1 is correct.

Solution By Deduction Theorem, it is enough to show (see the consequence in Example 1.1) that $\{p \to r, p \to (r \to q), p\} \models q$. To show the validity of this consequence, we construct a truth table where p is assigned to 1; so half of Table 1.3 is gone. The relevant ones are the rows 2,4,6,8. Among these, $p \to r$ is 1 only in the rows 6 and 8. Out of these, the only row where $p \to (r \to q)$ is 1, is the row 8. That is the only model of all the premises $p \to r, p \to (r \to q)$, and p. Here we also find that q is assigned to 1. The consequence is valid.

Alternatively, apply RAA to show that $\{p \to r, p \to (r \to q), p, \neg q\}$ is unsatisfiable. If the set is satisfiable, then there is an interpretation i such that $i(p \to r) = i(p \to (r \to q)) = i(p) = i(\neg q) = 1$. Then, $i(p) = 1$, $i(q) = 0$. Now, $i(p \to r) = 1$ forces $i(r) = 1$. So, $i(p \to (r \to q)) = 0$. This contradiction shows that $\{p \to r, p \to (r \to q), p, \neg q\}$ is unsatisfiable.

Theorem 1.6 *For any propositions u and $v, u \equiv v$ iff $\models u \leftrightarrow v$. Further, $u \equiv \top$ iff $\models u$.*

***Exercise* 1.19** Show that $u \models v$ iff $\models u \to v$. (Do not use the convention $\emptyset \models u \to v$ iff $\models u \to v$.) Using this, prove Theorems 1.6.

Before exploring equivalences, we will observe one more curious fact about PL, namely, monotonicity. The consequence relation is monotonic in the sense that if you add more premises, then an earlier conclusion still holds. Later you will discover many logics which violate this property, and as such they will be termed nonmonotonic logics.

Theorem 1.7 (Monotonicity) *Let w be a proposition and Σ, Γ be sets of propositions with $\Sigma \subseteq \Gamma$. If $\Sigma \models w$, then $\Gamma \models w$.*

Proof Suppose that $\Sigma \models w$. Let $i \models \Gamma$. Then $i(x) = 1$ for every $x \in \Gamma$. As $\Sigma \subseteq \Gamma$, we see that $i(y) = 1$ for every $y \in \Sigma$, i.e., $i \models \Sigma$. Since $\Sigma \models w, i(w) = 1$. Therefore, $\Gamma \models w$. ▮

Some of the important consequences and equivalences, referred as *laws*, are contained in the following statement.

Theorem 1.8 *Let x, y, z be propositions. The following consequences and equivalences hold:*

(a) LAW OF CONSTANTS:

$$x \wedge \top \equiv x, \quad x \wedge \bot \equiv \bot, \quad x \vee \top \equiv \top, \quad x \vee \bot \equiv x,$$

$$x \to \top \equiv \top, \quad x \to \bot \equiv \neg x, \quad \top \to x \equiv x, \quad \bot \to x \equiv \top,$$

$$x \leftrightarrow \top \equiv x, \quad x \leftrightarrow \bot \equiv \bot, \quad \neg\top \equiv \bot, \quad \neg\bot \equiv \top$$

(b) LAW OF EXCLUDED MIDDLE: $x \vee \neg x \equiv \top$

(c) LAW OF CONTRADICTION: $x \wedge \neg x \equiv \bot$, $x \leftrightarrow \neg x \equiv \bot$

(d) LAW OF DOUBLE NEGATION: $\neg\neg x \equiv x$

(e) LAW OF IDENTITY: $x \equiv x$, $x \leftrightarrow x \equiv \top$

(f) LAW OF IDEMPOTENCY: $x \wedge x \equiv x$, $x \vee x \equiv x$

(g) LAW OF ABSORPTION: $x \wedge (x \vee y) \equiv x$, $x \vee (x \wedge y) \equiv x$

(h) LAW OF COMMUTATIVITY:
$x \wedge y \equiv y \wedge x$, $x \vee y \equiv y \vee x$, $x \leftrightarrow y \equiv y \leftrightarrow x$

(i) LAW OF ASSOCIATIVITY: $x \wedge (y \wedge z) \equiv (x \wedge y) \wedge z$,
$x \vee (y \vee z) \equiv (x \vee y) \vee z$, $x \leftrightarrow (y \leftrightarrow z) \equiv (x \leftrightarrow y) \leftrightarrow z$

(j) LAW OF DISTRIBUTIVITY:

$$x \wedge (y \vee z) \equiv (x \wedge y) \vee (x \wedge z)$$

$$x \vee (y \wedge z) \equiv (x \vee y) \wedge (x \vee z)$$

$$x \vee (y \to z) \equiv (x \vee y) \to (x \vee z)$$

$$x \vee (y \leftrightarrow z) \equiv (x \vee y) \leftrightarrow (x \vee z)$$

$$x \to (y \wedge z) \equiv (x \to y) \wedge (x \to z)$$

$$x \to (y \vee z) \equiv (x \to y) \vee (x \to z)$$

$$x \to (y \to z) \equiv (x \to y) \to (x \to z)$$

$$x \to (y \leftrightarrow z) \equiv (x \to y) \leftrightarrow (x \to z)$$

(k) LAW OF DE MORGAN : $\neg(x \wedge y) \equiv \neg x \vee \neg y$, $\neg(x \vee y) \equiv \neg x \wedge \neg y$

(l) LAW OF IMPLICATION: $x \to x \equiv \top$,

$$x \to y \equiv \neg x \vee y, \qquad \neg(x \to y) \equiv x \wedge \neg y$$

$$x \to y \equiv x \leftrightarrow x \wedge y, \qquad x \to y \equiv x \vee y \leftrightarrow y,$$

(m) LAW OF CONTRAPOSITION: $x \to y \equiv \neg y \to \neg x$

(n) LAW OF HYPOTHESIS INVARIANCE: $x \to (y \to x) \equiv \top$

(o) LAW OF HYPOTHETICAL SYLLOGISM: $\{x \to y, y \to z\} \models x \to z$

(p) LAW OF EXPORTATION: $x \to (y \to z) \equiv (x \wedge y) \to z \equiv y \to (x \to z)$

(q) LAW OF BICONDITIONAL:

$$x \leftrightarrow y \equiv (x \to y) \wedge (y \to x), \qquad x \leftrightarrow y \equiv (x \wedge y) \vee (\neg x \wedge \neg y)$$
$$\neg(x \leftrightarrow y) \equiv (x \wedge \neg y) \vee (\neg x \wedge y), \qquad \neg(x \leftrightarrow y) \equiv \neg x \leftrightarrow y \equiv x \leftrightarrow \neg y$$

(r) LAW OF INTRODUCTION: $\{x, y\} \models x \wedge y, \quad x \models x \vee y, \quad y \models x \vee y$

(s) LAW OF ELIMINATION : $x \wedge y \models x, \quad x \wedge y \models y$

(t) LAW OF MODUS PONENS: $\{x, x \to y\} \models y$

(u) LAW OF MODUS TOLLENS: $\{x \to y, \neg y\} \models \neg x$

(v) LAW OF PIERCE: $(x \to y) \to x \models x$

(w) LAW OF CLAVIUS: $(\neg x \to x) \models x$

(x) LAW OF THE CASES: *If* $y \models z$, *then* $x \wedge y \models x \wedge z$.
If $x \models z$ *and* $y \models z$, *then* $x \vee y \models z$.

(y) LAW OF DISJUNCTIVE SYLLOGISM: $((x \vee y) \wedge \neg x) \models y$

(z) LAW OF UNIFORM SUBSTITUTION: *For any propositions* x, y *and any propositional variable* p *occurring in* x, *let* $x[p/y]$ *denote the proposition obtained from* x *by substituting every occurrence of* p *by* y. *If* $\models x$, *then* $\models x[p/y]$.

***Exercise* 1.20** Prove all the laws in Theorem 1.8.

There is no need to memorize all the above tautologies, equivalences, and consequences. They should be internalized by their use. However, use of these laws requires something more. For example, to show that $\models (p \to q) \vee (p \wedge \neg q)$, you may proceed as follows:

$$(p \to q) \vee (p \wedge \neg q) \;\equiv\; (p \to q) \vee (\neg(p \to q)) \;\equiv\; \top$$

since $\neg(p \to q) \equiv (p \wedge \neg q)$. In so doing, you have substituted $\neg(p \to q)$ in place of $p \wedge \neg q$ and you have apparently claimed that this substitution preserves equivalence, because $p \wedge \neg q \equiv \neg(p \to q)$. Essentially, you have used the Euclidean principle that substitution of equals yields equals. Does that principle hold in PL? For discussing that let us use a notation.

Let x, y, w be propositions. The expression $w[x := y]$ denotes any proposition obtained from w by replacing some (or none or all) occurrences of x by y. Notice the difference between this substitution and the one $w[x/y]$ introduced in the law of uniform substitution (Theorem 1.8(z)). In obtaining $w[x/y]$ from w, you must substitute *every* occurrence of x by y in w, while, in obtaining $w[x := y]$ you may substitute *some* of the occurrences of x by y in w. For example, if $w = p \wedge q \wedge \neg r \rightarrow \neg p \vee q$, then $w[p := p] = w$, while $w[p := q]$ can be any of

$$q \wedge q \wedge \neg r \rightarrow \neg p \vee q, \quad p \wedge q \wedge \neg r \rightarrow \neg q \vee q, \quad \text{or} \quad q \wedge q \wedge \neg r \rightarrow \neg q \vee q.$$

Similarly, $w[s := p] = w$ since s does not occur in w at all. The following statement approves the use of the Euclidean principle of substituting equals by equals.

Theorem 1.9 (Equivalence Substitution) *Let x, y, w be propositions and $w[x := y]$ denote any proposition obtained from w by substituting some or all or no occurrences of x by y in w. If $x \equiv y$, then $w \equiv w[x := y]$.*

Proof: Let i be an interpretation. Since $x \equiv y, i(x) = i(y)$. For computing $i(w)$, what matters is $i(x)$ and/or $i(y)$, but not the subpropositions x, y. Hence, $i(w) = i(w[x := y])$. This completes the proof. (Imagine a truth table for w and $w[x := y]$, where i is simply a row.) ∎

Note that if x does not occur in w, then $w = w[x := y]$, and then Theorem 1.9 is obvious. If x occurs in w, and w is atomic, then $w = x$, and then either $w[x := y] = w$ or $w[x := y] = y$. In any case, Theorem 1.9 is clear. You can take this observation as the basis step of an induction proof of Theorem 1.9 and then proceed to complete the proof.

***Exercise* 1.21** Give an induction proof of Theorem 1.9.

1.4 Calculations

Once we have Equivalence Substitution, we can show the validity of many propositions and consequences with ease, avoiding construction of a truth table. Suppose we want to show that $x \wedge y \equiv (x \leftrightarrow (x \vee y \leftrightarrow y))$. Our plan is to use Theorems 1.6–1.9 to show this. By Theorem 1.6, it is enough to show that $\models (x \wedge y \leftrightarrow (x \leftrightarrow (x \vee y \leftrightarrow y)))$. Now,

$$\begin{aligned}
& x \wedge y \leftrightarrow (x \leftrightarrow (x \vee y \leftrightarrow y)) && \text{[Associativity]}\\
&\quad \equiv \ (x \wedge y \leftrightarrow x) \leftrightarrow (x \vee y \leftrightarrow y) && \text{[Implication, Commutativity]}\\
&\quad \equiv \ (x \rightarrow y) \leftrightarrow (x \rightarrow y) && \text{[Identity]}\\
&\quad \equiv \ \top
\end{aligned}$$

Again, the use of Theorem 1.6 completes the job.

Let us look at the above series of equivalences closely. How is the first equivalence justified? To be specific, we know by the Law of Associativity (Theorem 1.8(i)) that $x \leftrightarrow (y \leftrightarrow z) \equiv (x \leftrightarrow y) \leftrightarrow z$. Then we uniformly substitute x as $x \wedge y$, y as x, and z as $x \vee y \leftrightarrow y$. Similarly, other equivalences hold. However, uniform substitution is expressed only for one variable, and here we may have to simultaneously substitute many variables. How to express this use of a law? Let us write any equivalence or consequence, etc. as an expression E, a metaexpression indeed. Now suppose that $x_1, x_2, x_3, \ldots, x_n$ (x, y, z above) are distinct propositions occurring in E. Let $w_1, \ldots, w_n$ be any propositions. Then an **instance** of E is the expression $E[x_1 := w_1, x_2 := w_2, \ldots, x_n := w_n]$. The rule of substitution, which we have used in the above calculation is that "if E is any law, then any of its instances can be used in a calculation". This also expresses the Euclidian ideal of substituting equals by equals, where equivalence is the notion of equality. A more precise formulation is contained in the statement below.

Theorem 1.10 (Leibniz Rule) *Let the atomic propositions $x_1, \ldots, x_n$ occur in the propositions E_1, E_2, and $w_1, \ldots, w_n$ be any propositions. Then,*

(a) $E_1 \equiv E_2$ implies

$$E_1[x_1 := w_1, \ldots, x_n := w_n] \equiv E_2[x_1 := w_1, \ldots, x_n := w_n]$$

(b) $E_1 \models E_2$ implies

$$E_1[x_1 := w_1, \ldots, x_n := w_n] \models E_2[x_1 := w_1, \ldots, x_n := w_n].$$

Proof It is enough to prove (b), as (a) follows from it (why?). For (b), suppose that $E_1 \models E_2$. By the deduction theorem, $\models E_1 \to E_2$. That is, whatever interpretation i we choose, $i(E_1 \to E_2) = 1$. Now, if j is any interpretation of $(E_1 \to E_2)[x_1 := w_1, \ldots, x_n := w_n]$, then it assigns values $j(w_1), \ldots, j(w_n)$ to $w_1, \ldots, w_n$. The same truth values are also assigned respectively to $x_1, \ldots, x_n$ by some interpretation, say, k, i.e., for the interpretation k, we have $k(x_m) = j(w_m)$, for $1 \le m \le n$. Then,

$$j((E_1 \to E_2)[x_1 := w_1, \ldots, x_n := w_n]) = k(E_1 \to E_2) = 1$$

Since j is arbitrary, we have $\models (E_1 \to E_2)[x_1 := w_1, \ldots, x_n := w_n]$. As

$$(E_1 \to E_2)[x_1 := w_1, \ldots, x_n := w_n] = E_1[x_1, \ldots, x_n] \to E_2[w_1, \ldots, w_n],$$

by the deduction Theorem, $E_1[x_1, \ldots, x_n] \models E_2[w_1, \ldots, w_n]$. ∎

We can, in fact, use our theorems and laws for devising a proof system, where equivalences and consequences can be proved as demonstrated in the above calculation. A **calculation** will be written as a sequence of propositions where successive propositions are linked by the symbol $\equiv$ or $\models$. Moreover, each step of the calculation which appears as $A \equiv B$ or as $A \models B$ must be justified by a single use of Leibniz, where $E_1 \equiv E_2$ or $E_1 \models E_2$ must be a law listed in Theorem 1.8. That is, a calculation will look like: $C_0 \oplus_1 C_1 \oplus_2 \ldots \oplus_m C_m$, where $\oplus_i \in \{\equiv, \models\}$, and every step $C_{i-1} \oplus_i C_i$ must be an instance of a law $E_1 \oplus_i E_2$. The calculation is taken as a proof of the metastatement $C_0 \otimes C_m$, where $\otimes = \equiv$ if all of $\oplus_i$ are equal to $\equiv$, and $\otimes = \models$ if at least one of $\oplus_i$ equals $\models$. Moreover, a proof of $\models u$ can be a proof of the statements $u \equiv \top$ or of $\top \models u$. And a proof of $\models u \to v$ can be any of the proofs of $u \models v$, $\top \models u \to v$. Similarly, a proof of $\models u \leftrightarrow v$ can be a proof of $\top \models u \leftrightarrow v$ or of $u \equiv v$. Moreover, a proof of unsatisfiability of a set $\Sigma = \{w_1, \ldots, w_n\}$ of propositions will be a proof of $w_1 \wedge \cdots \wedge w_n \models \bot$. For proving a consequence of the type $\{w_1, \ldots, w_n\} \models w$, we may either construct a proof of $w_1 \wedge \cdots \wedge w_n \models w$

or of $\models w_1 \wedge \cdots \wedge w_n \to w$ or of $\models w_1 \to ((\cdots w_2 \to \cdots (w_n \to w) \cdots)$. By RAA, we may also prove this consequence by constructing a proof of $w_1 \wedge \cdots \wedge w_n \wedge \neg w \models \bot$.

Another alternative is to use the hypotheses $w_1, \ldots, w_n$ as local truths. That is, we use $w_1 \equiv \top, \ldots w_n \equiv \top$ as additional (local, only for this proof) laws, and then construct a proof for $\models w$. It will be easier to use the first or the last alternatives than others, in general (why?). The following examples will illustrate the various alternatives.

***EXAMPLE* 1.3** Show by a calculation that $\{p \wedge (q \wedge r), s \wedge t\} \models q \wedge s$.

Solution We show that $(p \wedge (q \wedge r) \wedge (s \wedge t) \models (q \wedge s)$.

$$\begin{array}{ll} (p \wedge (q \wedge r) \wedge (s \wedge t) & [\text{Associativity, Commutativity}] \\ \quad \equiv\ (p \wedge (r \wedge t) \wedge (q \wedge s) & [\text{Elimination}] \\ \quad \models\ q \wedge s & \end{array}$$

***EXAMPLE* 1.4** Show that $p \to (q \to r), \neg r, p \models \neg q$.

Solution We use the hypotheses as additional (local) laws and try to have a proof of $\neg q \equiv \top$. The appropriate local laws are then $p \to (q \to r) \equiv \top$, $\neg r \equiv \top, p \equiv \top$, and these are mentioned in the proof as 'Hypothesis'. Our aim is to show that $\neg q \equiv \top$. The following is such a proof.

$$\begin{array}{ll} \neg q & [\text{Constants}] \\ \quad \equiv\ q \to \bot & [\text{Constants}] \\ \quad \equiv\ q \to \neg\top & [\text{Hypothesis}] \\ \quad \equiv\ q \to \neg\neg r & [\text{Double Negation}] \\ \quad \equiv\ q \to r & [\text{Constants}] \\ \quad \equiv\ \top \to (q \to r) & [\text{Hypothesis}] \\ \quad \equiv\ p \to (q \to r) & [\text{Hypothesis}] \\ \quad \equiv\ \top & \end{array}$$

Note that such a proof does not prove that a conclusion such as $\neg q$ above is indeed valid. It only says that if the hypotheses used in the proof are true under any interpretation, then under all those interpretations, the conclusion is also true.

***EXAMPLE* 1.5** Show that $\models (p \to r) \to ((\neg p \to \neg q) \to (q \to r))$.

Solution This can be shown in many ways. The easiest is to use the deduction theorem and construct a proof for $p \to r, \neg p \to \neg q, q \models r$ as in the following. To have a different approach from that in Example 1.4, we show that $(p \to r) \wedge (\neg p \to \neg q) \wedge q \models r$.

$$\begin{array}{ll}
(p \to r) \land (\neg p \to \neg q) \land q & \text{[Double Negation]} \\
\quad \equiv \ (p \to r) \land (\neg p \to \neg q) \land \neg\neg q & \text{[Modus Tollens]} \\
\quad \models \ (p \to r) \land (\neg\neg p) & \text{[Double Negation]} \\
\quad \equiv \ (p \to r) \land p & \text{[Modus Ponens]} \\
\quad \equiv \ r &
\end{array}$$

***EXAMPLE* 1.6** Show that $\{(p \to q) \to r, s \to \neg p, t, \neg s \land t \to q, \neg r\}$ is unsatisfiable.

Solution We show that the conjunction of all the hypotheses in the set entails $\bot$. Hereafter, we mention the names of the laws by using abbreviations such as ‘Mod Tol’ for ‘Modus Tollens’, ‘Impl’ for ‘Implication’, etc. We also use associativity of $\land$ and $\lor$ without mentioning it (find where).

$$\begin{array}{ll}
((p \to q) \to r) \land (s \to \neg p) \land t \land (\neg s \land t \to q) \land \neg r & \text{[Mod Tol]} \\
\quad \models \ \neg(p \to q) \land (s \to \neg p) \land t \land (\neg s \land t \to q) & \text{[Impl]} \\
\quad \equiv \ p \land \neg q \land (s \to \neg p) \land t \land (\neg s \land t \to q) & \text{[Mod Tol]} \\
\quad \models \ \neg q \land \neg s \land t \land (\neg s \land t \to q) & \text{[Mod Tol]} \\
\quad \models \ \neg s \land t \land \neg(\neg s \land t) & \text{[Const]} \\
\quad \equiv \ \bot &
\end{array}$$

Alternatively, you can start with a conjunction of a few suitable hypotheses (or just one hypothesis), then go on introducing others, whenever required so that a proof may be obtained. Look at the following alternate solution to Example 1.6 (give reasons at each step):

An Alternate Solution

$$\begin{array}{l}
((p \to q) \to r) \land \neg r \\
\quad \equiv \neg(p \to q) \\
\quad \equiv p \land \neg q \\
\quad \equiv p \land \neg q \land \top \\
\quad \equiv (p \land \neg q) \land (s \to \neg p) \\
\quad \models \neg q \land \neg s \\
\quad \equiv \neg q \land \neg s \land \top \\
\quad \equiv \neg q \land \neg s \land (\neg s \land t \to q) \\
\quad \models \neg s \land \neg(\neg s \land t) \\
\quad \equiv \neg s \land (t \to s) \\
\quad \models \neg t \\
\quad \equiv \neg \top \\
\quad \equiv \bot
\end{array}$$

1.5 Normal Forms

Another use of the laws is to see whether you really require all the five connectives to do propositional logic. One look at the laws will tell you that from any proposition you can eliminate the connectives $\leftrightarrow, \rightarrow, \vee$ and also the propositional constants $\top$ and $\bot$ by equivalences, since

$$x \leftrightarrow y \equiv \neg(x \wedge \neg y) \wedge \neg(y \wedge \neg x), \quad x \rightarrow y \equiv \neg(x \wedge \neg y)$$

$$x \vee y \equiv \neg(\neg x \wedge \neg y), \quad \top \equiv \neg(x \wedge \neg x), \quad \bot \equiv x \wedge \neg x$$

That is, we could have started with our only connective as $\neg$ and $\wedge$, and then introduced other connectives and the propositional constants as definitions or abbreviations. This fact is often expressed by saying that in PL, the set $\{\neg, \wedge\}$ is an **adequate set** of connectives.

***Exercise* 1.22** Show that each of the sets $\{\neg, \vee\}, \{\neg, \rightarrow\}, \{\bot, \rightarrow\}$ is also an adequate set.

Can you go further in reducing the sizes of the adequate sets? Is $\{\neg\}$ an adequate set? That is, can you define $\wedge$ in terms of $\neg$ alone? It seems impossible. But how to show this? Now in PL, we have the propositional variables $p_0, p_1, \ldots$. If $\neg$ is the only connective to be used, we would generate the formulas:

$$p_0, p_1, p_2, \cdots, \quad \neg p_0, \neg p_1, \neg p_2, \ldots, \quad \neg\neg p_0, \neg\neg p_1, \neg\neg p_2, \ldots,$$

$$\neg\neg\neg p_0, \neg\neg\neg p_1, \ldots, \quad \neg\neg\neg\neg p_0, \neg\neg\neg\neg p_1, \ldots, \ldots$$

Up to equivalence, the propositions reduce to

$$p_0, p_1, p_2, p_3, \cdots, \quad \neg p_0, \neg p_1, \neg p_2, \neg p_3, \ldots, \ldots$$

Now, is any of these propositions equivalent to $p_0 \wedge p_1$? Definitely not. Because $p_0 \wedge p_1 \not\equiv p_0$ as the interpretation i with $i(p_0) = 1$, $i(p_1) = 0$ is a model of p_0, but not a model of $p_0 \wedge p_1$. Similarly, $p_0 \wedge p_1 \not\equiv p_1 \ldots$

***Exercise* 1.23** The list above is infinite. Then how do you show that $p_0 \wedge p_1$ is not equivalent to any of the propositions $p_0, p_1, \cdots, \neg p_0, \neg p_1, \ldots,$? [*Hint*: Show that $p_0 \wedge p_1 \not\equiv p_i$ if $i \neq 0, i \neq 1$, and that $p_0 \wedge p_1 \not\equiv \neg p_i$ if $i \neq 0, i \neq 1$.]

***Exercise* 1.24** Show that the set $\{\neg\}$ is not adequate even if we use the propositional constants $\top$ and $\bot$.

The connective $\wedge$ does not form an adequate set since $\neg p$ is not equivalent to any of $p \wedge p,\ p \wedge \neg p,\ \neg p \wedge \neg p,\ \top,\ \bot,\ p \wedge \top,\ p \wedge \bot,\ \neg p \wedge \top,\ \neg p \wedge \bot$.

***Exercise* 1.25** Why does it suffice to consider the nine propositions only for showing that $\{\wedge\}$ is inadequate?

***Exercise* 1.26** Show that none of the sets $\{\vee\}, \{\rightarrow\}, \{\leftrightarrow\}$ is an adequate set. Is $\{\neg, \leftrightarrow\}$ an adequate set?
[*Hint*: Show that $\neg p$ cannot be expressed by any one of them.]

Treating the propositional constants $\top$ and $\bot$ as 0-ary connectives, you can show that $\bot, \to$ is adequate. For this, you only need to observe that $\neg p \equiv p \to \bot$. Why is it enough? However, there are two binary truth functions which we have not taken as connectives. We will see why they are special. They are given in the following truth table.

Table 1.4 NAND and NOR

p	q	$p \uparrow q$	$p \downarrow q$
0	0	1	1
1	0	1	0
0	1	1	0
1	1	0	0

It is easy to see that $p \uparrow q \equiv \neg(p \wedge q)$ and $p \downarrow q \equiv \neg(p \vee q)$; thus, $\uparrow$ is called NAND and $\downarrow$ is called NOR. Now,

$$\neg p \equiv p \uparrow p,\ \ p \wedge q \equiv \neg\neg(p \wedge q) \equiv \neg(p \uparrow q) \equiv (p \uparrow q) \uparrow (p \uparrow q)$$

Since $\{\neg, \wedge\}$ is an adequate set, so is $\{\uparrow\}$. Similarly, you can verify that

$$\neg p \equiv p \downarrow p,\ \ p \vee q \equiv \neg\neg(p \vee q) \equiv \neg(p \downarrow q) \equiv (p \downarrow q) \downarrow (p \downarrow q)$$

Therefore, $\{\downarrow\}$ is also an adequate set.

***Exercise* 1.27** Find explicit expressions of all the five connectives in terms of (a) $\uparrow$ alone, (b) $\downarrow$ alone.

If you view interpretations and models as in our second view of semantics, then each compound proposition is simply a truth function, i.e., a function which takes 0's and 1's to 0's and 1's. In that sense, $\neg$ is the unary truth function $\neg : \{0,1\} \to \{0,1\}$, $\wedge$ is the binary truth function $\wedge : \{0,1\} \times \{0,1\} \to \{0,1\}, \ldots$ Then $\uparrow$ is a function from $\{0,1\} \times \{0,1\}$ to $\{0,1\}$ and is given explicitly by $\uparrow (0,0) = \uparrow (0,1) = \uparrow (1,0) = 0$ and $\uparrow (1,1) = 0$. Since $\uparrow$ alone forms an adequate set, it follows that any binary truth function, i.e., any function $f : \{0,1\} \times \{0,1\} \to \{0,1\}$ can be expressed as a (serial) composition of $\uparrow$ alone. Since with two arguments there are $2^2 = 4$ interpretations, there are $2^4 = 16$ binary truth functions. Out of these 16 truth functions, $\uparrow$ and $\downarrow$ are the only truth functions, each of which forms an adequate set. That is, $\{\uparrow\}$ and $\{\downarrow\}$ are the only singletons which are *truth functionally complete* (another name for 'adequate'). Can you prove this?

We will take a middle path. We know that both $\{\neg, \wedge\}$ and $\{\neg, \vee\}$ are adequate sets of connectives, and so is the set $\{\neg, \wedge, \vee\}$ though a bit superfluous. However, admitting three connectives gives us nice forms of propositions. For example,

$$\begin{aligned} &(\neg p \to q) \to (p \vee r \to q \wedge s) && \text{[Impl]} \\ \equiv\ &\neg(\neg p \to q) \vee (p \vee r \to q \wedge s) && \text{[Impl]} \end{aligned}$$

$$\begin{aligned} &\equiv (p \wedge \neg q) \vee (\neg(p \vee r) \vee (q \wedge s)) && \text{[De Mor]} \\ &\equiv (p \wedge \neg q) \vee ((\neg p \wedge \neg r) \vee (q \wedge s)) && \text{[Assoc]} \\ &\equiv (p \wedge \neg q) \vee (\neg p \wedge \neg r) \vee (q \wedge s) \end{aligned}$$

The last proposition is a disjunction of conjunctions. There are certain advantages of these types of propositions. For instance, it is relatively easier to find out all the models of such a proposition at a glance. They are of three types, one where $p, \neg q$ are 1, two where $\neg p, \neg r$ are 1, and three where q, s are 1, and nothing else (Why?). We fix some terminology.

Definition 1.9 A **literal** is either a propositional variable, or negation of a propositional variable. For any propositional variable p, the literals p and $\neg p$ are called complementary (to each other) literals. A conjunctive clause is a conjunction of literals; a disjunctive clause is a disjunction of literals. A conjunctive normal form proposition (**cnf**) is a conjunction of disjunctive clauses; a disjunctive normal form (**dnf**) is a disjunction of conjunctive clauses.

The calculation above shows the equivalence

$$(\neg p \rightarrow q) \rightarrow (p \vee r \rightarrow q \wedge s) \equiv (p \wedge \neg q) \vee (\neg p \wedge \neg r) \vee (q \wedge s)$$

where the second one is in dnf. So, what do you guess?

Theorem 1.11 (Normal Form) *Every proposition is equivalent to a cnf and also to a dnf.*

Proof Let w be any proposition. Suppose the set of all propositional variables occurring in w is $A = \{q_1, \ldots, q_n\} \neq \emptyset$. [If $A = \emptyset$, then replace $\top$ by $q_1 \vee \neg q_1$ and $\bot$ by $q_1 \wedge \neg q_1$ in w to obtain an equivalent proposition with $A = \{q_1\}$.] Now any interpretation is a function from A to $\{0, 1\}$. Consider the set M of all models of w. Corresponding to each model j of w, we will construct a conjunctive clause C_j. If $j(q_k) = 1$, then let $l_{jk} = q_k$; else, if $j(q_k) = 0$, let $l_{jk} = \neg q_k$. Take $C_j = l_{j1} \wedge l_{j2} \wedge \cdots \wedge l_{jm}$, and 'or' them together to get $w' = C_1 \vee C_2 \vee \cdots \vee C_n$, where there are n number of models of w. Our construction of a dnf w' is over. It remains to show that indeed, $w \equiv w'$.

To see that $w \equiv w'$, let $i \models w$. Then as per the constructions of the literals, we see that $i(l_{i1}) = i(l_{i2}) = \cdots = i(l_{im}) = 1$. That is, $i(C_i) = 1$. Since w' is a disjunction of the C_i's, we have $i \models w'$, proving that $w \models w'$. Conversely, let $v \models w'$. Then there is at least one C_k such that $v \models C_k$. Then $v(l_{k1}) = v(l_{k2}) = \cdots = v(l_{km}) = 1$. That is, v coincides with the interpretation k, which of course is a model of w. Hence $v \models w$. Since v is an arbitrary interpretation, we have shown that $w' \models w$.

Use the law of distributivity on w' to get an equivalent cnf w''. ▮

The above proof goes like this. Suppose you want a dnf representation of $w = \neg(p \leftrightarrow q)$. First, find all possible models of w. From the truth table for w (construct it), you see that the models are i and j, where $i(p) = 1$,

$i(q) = 0$ and $j(p) = 0, j(q) = 1$. The conjunctive clause corresponding to i is $p \wedge \neg q$, and one that corresponds to j is $\neg p \wedge q$. So, the dnf representation of w is $w' = (p \wedge \neg q) \vee (\neg p \wedge q)$. The cnf is obtained by distribution of $\vee$ over $\wedge$, thereby giving $w'' = (p \vee \neg p) \wedge (p \vee q) \wedge (\neg q \vee \neg p) \wedge (\neg q \vee q)$. Of course, w'' can further be simplified to $w'' \equiv (\neg q \vee \neg p) \wedge (p \vee q)$. Why?

***Exercise* 1.28** Use the nonmodels of w, i.e., the interpretations which falsify w, to construct a cnf equivalent to w. [*Hint*: You may have to take p (not $\neg p$) when it is false (not true) in an interpretation.]

However, it looks as though the construction in the above proof is wasteful. To effect such a construction, you must have a truth table with 2^n rows if there are n propositional variables in w. Indeed, a cnf or a dnf conversion can be achieved by using equivalences. The following procedure, called *NorFor*, does just that.

```
PROCEDURE : NorFor
    Input: Any proposition $w$ of PL
    Output: A cnf and a dnf equivalent to $w$

    1. Eliminate the connectives $\rightarrow, \leftrightarrow$ by using the laws of
       Implication and Biconditional.
    2. Use De Morgan to take $\neg$ close to the propositional variables.
    3. Use Double Negation to have either one $\neg$ or none at all with
       any propositional variable.
    4. Use Distributivity to get the required cnf or dnf.
```

Try to prove that the procedure really works. Look at the laws used in the procedure. Do you feel that using only these laws, all the other laws in Theorem 1.8 can be derived by calculations? If so, why? If not, what else do you require? Note that you can use other laws (like absorption, constants, etc) for simplifying the normal forms. They can also be used anywhere in the procedure, and not only after the procedure is complete. It is a good idea to modify the procedure accordingly.

***EXAMPLE* 1.7** Convert $(p \rightarrow (\neg q \rightarrow r)) \wedge (p \rightarrow \neg q)$ to a cnf and also to a dnf.

Solution

$$\begin{aligned}
&(p \rightarrow (\neg q \rightarrow r)) \wedge (p \rightarrow \neg q)\\
&\quad \equiv (\neg p \vee (\neg q \rightarrow r)) \wedge (\neg p \vee q)\\
&\quad \equiv (\neg p \vee (\neg\neg q \vee r)) \wedge (\neg p \vee q)\\
&\quad \equiv (\neg p \vee q \vee r) \wedge (\neg p \vee q)
\end{aligned}$$

The last proposition is in cnf. Distributivity gives:

$$(\neg p \wedge \neg p) \vee (\neg p \wedge \neg q) \vee (q \wedge \neg p) \vee (q \wedge \neg q) \vee (r \wedge \neg p) \vee (r \wedge \neg q)$$

This is in dnf. Further simplification gives:

$$\neg p \vee (\neg p \wedge \neg q) \vee (q \wedge \neg p) \vee (r \wedge \neg p) \vee (r \wedge \neg q)$$

Looking at the procedure from inside, you will find that for computing a dnf or a cnf representation, you must show how to compute a cnf or dnf of the forms $x \wedge y, x \vee y$ and of $\neg x$. This is because, after eliminating the connectives $\rightarrow, \leftrightarrow$, you will be getting one of these forms. Suppose that you have already obtained a cnf of x and of y; call them $cnf(x)$ and $cnf(y)$, respectively. Then $cnf(x \wedge y)$ is simply $cnf(x) \wedge cnf(y)$. What about the $cnf(x \vee y)$? If x, y are literals, then $cnf(x \vee y) = x \vee y$. Otherwise? Well, for propositional variables p, q, you have:

$$cnf(p \wedge q) = p \wedge q, \;\; cnf(\neg p \vee q) = \neg p \vee q$$

Now, how to compute $cnf((p \wedge q) \vee (\neg p \vee q))$? Obviously, you must distribute. Distributivity of $\vee$ over $\wedge$ gives:

$$cnf((p \wedge q) \vee (\neg p \vee q)) = (p \vee \neg p \vee q) \wedge (q \vee \neg p \vee q)$$

This requires to define (as in functional programming) $dist(x, y)$. Here is such a definition of $dist(x, y)$:

Let C_i, D_j be disjunctive clauses. If $x = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ and $y = D_1 \wedge D_2 \wedge \cdots \wedge D_n$, then $dist(x, y) = \wedge_{i,j}(C_i \vee D_j)$.

Once distribution is available, you can define $cnf(w)$. There are three cases to consider, as in the following:

(a) If w is a literal, then $cnf(w) = w$.

(b) If w is $x \wedge y$, then $cnf(w) = cnf(x) \wedge cnf(y)$.

(c) If w is $x \vee y$, then $cnf(w) = dist(cnf(x), cnf(y))$.

***Exercise* 1.29** Define $dnf(w)$ and write programs in any language you know to convert a proposition to a cnf and a dnf using $cnf(w)$ and $dnf(w)$.

The normal forms can be used to decide whether a proposition is valid, invalid, satisfiable, or unsatisfiable. For example, $(p \vee \neg p \vee q) \wedge (q \vee \neg q \vee r)$ is valid. But the proposition $(p \vee q) \wedge (\neg p \vee p \vee r)$ is invalid since $i(p) = 0 = i(q)$ with $i(r) = 0$ or 1 does not satisfy it. Similarly, $(p \wedge \neg p \wedge q) \vee (q \wedge \neg q \wedge r)$ is unsatisfiable while $(p \wedge q) \vee (\neg p \wedge p \wedge r)$ is satisfiable. In general, if both $p, \neg p$ occur in a conjunctive clause, the clause is unsatisfiable. Similarly, if both $p, \neg p$ occur in a disjunctive clause, the clause is valid. A cnf being a conjunction of disjunctive clauses, it is easy to see that, if each disjunctive clause in it is valid, then the cnf is valid. Similarly, if each conjunctive clause in a dnf is unsatisfiable, then the dnf is also unsatisfiable. But to determine the validity of a cnf or unsatisfiability of a dnf, we must also see the converse statements. Let us see this for a dnf.

Suppose that $w = D_1 \wedge D_2 \wedge \cdots \wedge D_n$ is a dnf and that $\models w$. Then each interpretation evaluates each of the clauses D_j to 1. That is, each D_j is valid. Now, how can D_1 be valid? $D_1 = l_1 \vee l_2 \vee \cdots \vee l_k$, where the l's are literals. Our claim here is that for D_1 to be valid, there must be a pair of complementary literals among these. Because, otherwise, we can define an interpretation i such that $i(l_1) = \cdots i(l_k) = 0$ and this interpretation will falsify D_1. A similar argument holds for each D_j. Analogously, we have a corresponding argument for satisfiability of a cnf. Thus, we have proved the following statement.

Theorem 1.12 *A cnf is valid iff each disjunctive clause in it contains a pair of complementary literals. A dnf is unsatisfiable iff each conjunctive clause in it contains a pair of complementary literals.*

***EXAMPLE* 1.8** Categorize the following propositions into valid, invalid, satisfiable, or unsatisfiable by converting into a suitable normal form.

(a) $(p \to q) \lor (q \to \neg r) \lor (r \to q) \to \neg(\neg(q \to p) \to (q \leftrightarrow r))$

(b) $\neg((p \to q) \land (q \to r) \to (q \to r))$

(c) $(p \to (\neg q \to r)) \land (p \to \neg q) \to (p \to r)$.

Solution (a) Let us get an equivalent cnf or dnf.

$$\begin{aligned}
&(p \to q) \lor (q \to \neg r) \lor (r \to q) \to \neg(\neg(q \to p) \to (q \leftrightarrow r))\\
&\equiv (p \to q) \lor (q \to \neg r) \lor (r \to q) \to \neg((q \to p) \lor (q \leftrightarrow r))\\
&\equiv (p \to q) \lor (q \to \neg r) \lor (r \to q) \to (\neg(q \to p) \land \neg(q \leftrightarrow r))\\
&\equiv (p \to q) \lor (q \to \neg r) \lor (r \to q) \to (q \land \neg p \land ((q \land \neg r) \lor (\neg q \land r)))\\
&\equiv (\neg p \lor q \lor \neg q \lor \neg r \lor \neg r \lor q) \to ((q \land \neg p \land q \land \neg r) \lor (q \land \neg p \land \neg q \land r))\\
&\equiv \neg(\neg p \lor q \lor \neg q \lor \neg r \lor \neg r \lor q) \lor (q \land \neg p \land q \land \neg r) \lor (q \land \neg p \land \neg q \land r)\\
&\equiv (p \land \neg q \land q \land r \land r \land \neg q) \lor (q \land \neg p \land q \land \neg r) \lor (q \land \neg p \land \neg q \land r)\\
&\equiv (p \land \neg q \land q \land r) \lor (\neg p \land q \land \neg r) \lor (\neg p \land q \land \neg q \land r)
\end{aligned}$$

This is in dnf, having at least one conjunctive clause which does not have a pair of complementary literals. In the first and third clauses, $q, \neg q$ occur, while in the second, $\neg p$ occurs but not p, q occurs but not $\neg q$, and $\neg r$ occurs but not r. Thus it is satisfiable.

For validity, you have to convert it to a cnf, say, by distributing the $\lor$'s over $\land$'s in the dnf. However, there is a shorter approach here. Since both the first and the third clauses have a pair of complementary literals, they are each equivalent to $\bot$. Moreover, $\bot \lor x \equiv x$. Therefore, the above dnf is equivalent to the second clause only, i.e., it is equivalent to:

$$\neg p \land q \land \neg r$$

which is in both cnf and dnf. The cnf has now three clauses, namely, $\neg p, q, \neg r$; neither has a pair of complementary literals. Thus the proposition is invalid.

This calculation shows that it is sometimes advantageous to simplify the normal forms whenever possible. However, simplification may result in $\top$ or $\bot$ only. Of course, when you get $\top$, the proposition must be valid;

similarly, if you end up in $\bot$, it is unsatisfiable. That is, as cnf and dnf, $\top$ must be a cnf, while $\bot$ must be a dnf.

(b) $\neg(((p \to q) \wedge (q \to r)) \to (q \to r))$

$$\begin{aligned} &\equiv (p \to q) \wedge (q \to r) \wedge \neg(q \to r) \\ &\equiv (\neg p \vee q) \wedge (\neg q \vee r) \wedge q \wedge \neg r \end{aligned}$$

This is in cnf, and you may conclude that it is invalid. But is it also satisfiable? We need a dnf. So? Distribute and simplify to get

$$(\neg p \vee q) \wedge ((\neg q \wedge q \wedge \neg r) \vee (r \wedge q \wedge \neg r)) \equiv (\neg p \vee q) \wedge \bot \equiv \bot$$

Hence, it is unsatisfiable. If you do not use $\bot$, then you would end up in a bigger expression like:

$$(\neg p \wedge \neg q \wedge q \wedge \neg r) \vee (\neg p \wedge r \wedge q \wedge \neg r) \vee (q \wedge \neg q \wedge q \wedge \neg r) \vee (q \wedge r \wedge q \wedge \neg r)$$

from which the same conclusion is drawn.

(c) $((p \to (\neg q \to r)) \wedge (p \to \neg q)) \to (p \to r)$

$$\begin{aligned} &\equiv \neg((p \to (\neg q \to r)) \wedge (p \to \neg q)) \vee (p \to r) \\ &\equiv \neg(p \to (\neg q \to r)) \vee \neg(p \to \neg q) \vee (p \to r) \\ &\equiv (p \wedge \neg(\neg q \to r)) \vee (p \wedge \neg\neg q) \vee (\neg p \vee r) \\ &\equiv (p \wedge \neg q \wedge \neg r) \vee (p \wedge q) \vee \neg p \vee r \end{aligned}$$

This is in dnf having at least one clause, say, the last one, r, which does not have a pair of complementary literals. Hence the proposition is satisfiable. But is it valid? By distributing and simplifying, you find that

$$\begin{aligned} &(p \wedge \neg q \wedge \neg r) \vee (p \wedge q) \vee \neg p \vee r \\ &\quad \equiv (p \wedge \neg q \wedge \neg r) \vee ((p \vee \neg p \vee r) \wedge (q \vee \neg p \vee r)) \\ &\quad \equiv (p \wedge \neg q \wedge \neg r) \vee (\top \wedge (q \vee \neg p \vee r)) \\ &\quad \equiv (p \wedge \neg q \wedge \neg r) \vee (q \vee \neg p \vee r) \\ &\quad \equiv (p \vee q \vee \neg p \vee r) \wedge (\neg q \vee q \vee \neg p \vee r) \wedge (\neg r \vee q \vee \neg p \vee r) \end{aligned}$$

This is in cnf where each clause has a pair of complementary literals. Hence the proposition is valid.

***Exercise* 1.30** Redefine cnf and dnf so that $\bot$ is a dnf and $\top$ is a cnf.

While solving the examples above, you had encountered an awkward situation! Can you recollect? You had seen a proposition to be satisfiable, but, in fact, a stronger property holds, namely, it is valid. Similarly, you had concluded some proposition to be invalid, but later you find that it is also unsatisfiable. (Which propositions above are we referring to?) For concluding the stronger properties, you had to distribute and find out the dual normal form; i.e., if you had already a cnf, you need a dnf and vice versa. Can there be a shorter way than using distributivity? To be specific, how to determine the validity of a dnf or the satisfiability of a cnf without using distributivity? If you are impatient, then see the chapter on Resolution right now.

You have seen that by using calculations, any proposition can be converted to an equivalent cnf and also a dnf. Then from these normal forms, it can be decided whether the proposition is valid or satisfiable. Moreover, the laws are correct semantically. The correctness of the calculations may be stated as if a calculation shows that $\models w$, then, it is indeed so. This is referred to as the **soundness** of calculations. The fact that calculations can be used to prove $\models w$ is also obvious, at least through normal form conversions. Suppose that $\models w$. Then our calculation procedure tells us that we can convert w to an equivalent cnf, and then we can decide whether w is valid or not. This is referred to as the **completeness** of calculations. If a proof procedure is both sound and complete, then we say that the procedure is **adequate**. You can show that for every finite set Σ of propositions and any proposition w, $\Sigma \models w$ iff there is a calculational proof of it. If Σ is infinite, will such a statement hold?

1.6 Some Applications

The logic of propositions is so fundamental that its applications are found almost everywhere, starting from day-to-day conversation to electrical circuits and software designing. In this section we will briefly review some of these. Though we do not discuss it further, it is nonetheless very important. For, applications usually start a field of study and continuously enrich it by bringing in previously unimagined problems and peculiar features. We will not discuss all the applications of PL; see the summary at the end of the chapter. We start with a small puzzle.

***EXAMPLE* 1.9** [A Puzzle] In the Kniknaord island there are knights, knaves and possibly ordinary people. Knights always tell the truth, knaves always lie to strangers; ordinary people sometimes tell the truth, and sometimes lie to strangers. There are two uninhabited islands A and B. In at least one of these islands there is gold buried by some preposterous pirate. By good fortune you find a message by an earlier gold hunter which reads:

(a) There is no gold on Kniknaord.
(b) If there are any ordinary people in Kniknaord, then there is gold in two of the islands.

Supposing that the natives in Kniknaord know all about the buried gold, you are (a stranger, of course) allowed to ask only one question to exactly one native chosen at random. What question would you ask?

Take some time before reading the solution. Even while you read the solution, do not read it at a time. Go a step, and then try to solve the problem from that point onwards yourself. Finally, read it.

Solution From (a), it is obvious that there is gold in at least one of the islands A and B. Further, if there are ordinary people in Kniknaord, then from (b) it follows that there is gold in both the islands A and B. So, you can safely assume that there is no ordinary people in Kniknaord. Now, you must determine by your question (and then from the answer of the native) whether there is gold in island A (for then, there is no gold in island B, or, even if it is, it does not matter). Thus, you might pose a question in the form 'Is Q true?', whence, if there is gold in A, then no matter what the native is, his response must be 'yes'; and if the response is 'no', then it must lead you to conclude that there is no gold in A (so that it is in B). However, you must take care of the truth in what the native says in view of whether he is a knight or a knave. Therefore, Q will involve two simple propositions:

p: There is gold in island A.

q: You are a knight.

You want to determine the proposition Q so that the truth of p would be the same as the response of the native to the question "Is Q true?" Note that Q will be a compound proposition built from p and q. In this case, if the response to your question is 'no', then p is false, irrespective of the value of q. Similarly, if the response is 'yes', then p is true, whatever the value of q may be. Writing 1 for 'true', 0 for 'false' for the values of p, q, and 1 for a 'yes' response, 0 for a 'no' response, you get the first three columns of Table 1.5. However, the truth of Q need not be the same as the response. In the first row of the table, q is 1, i.e., the native is a knight. His response is 1 means that the truth value of Q is 1. In the second row, q is 0, so the native is a knave, and then his response and the truth value of Q must be opposites. In the third row, q is 1, so his response must be the same as the truth value of Q; and in the fourth row, his response must be opposite to the truth value of Q. Thus, we have the truth values of the unknown proposition Q corresponding to those of the atomic propositions p, q as in Table 1.5.

Table 1.5 Kniknaordian's Response

p	q	response	Q
1	1	1	1
1	0	1	0
0	1	0	0
0	0	0	1

From the truth values of p, q, Q, it is clear that $Q \equiv p \leftrightarrow q$. If it is not very clear, you can, of course, use to construct a cnf from the truth values as in the proof of the normal form theorem. Hence, your question is:

Is it true that you are a knight if and only if there is gold buried in island A?

If the answer is 'yes', then there is gold in island A; otherwise, there is gold in island B. Verify whether your question works, taking the native as a knight, knave and an ordinary person, in turn. You must verify the solution now. There is one more solution to this puzzle, though similar. Can you find it?

Next, we will briefly see how PL is used in circuit designs. The basic logic functions, 'not', 'and', 'or' are realized in electronic circuits by the so-called gates. The gates for the corresponding basic truth functions $\neg, \wedge, \vee$ are symbolized by hardware engineers in a different way. See Figure 1.7.

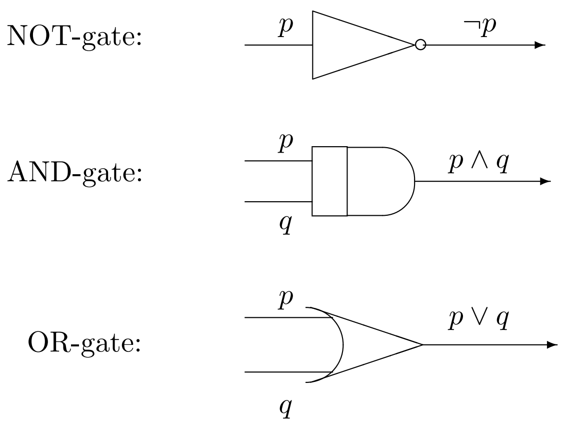


Figure 1.7 Basic logic gates.

The NOT-gate means that if a wire carries a large amount of voltage, then the gate outputs a small amount and vice versa. Similarly, the AND-gate means that if two wires carry some voltage p, q, then the gate outputs $\min(p, q)$. Similarly, the OR-gate outputs $\max(p, q)$. The voltage p, q represent either a 'small' or a 'large' amount of voltage, and are modelled by the truth values 0 and 1.

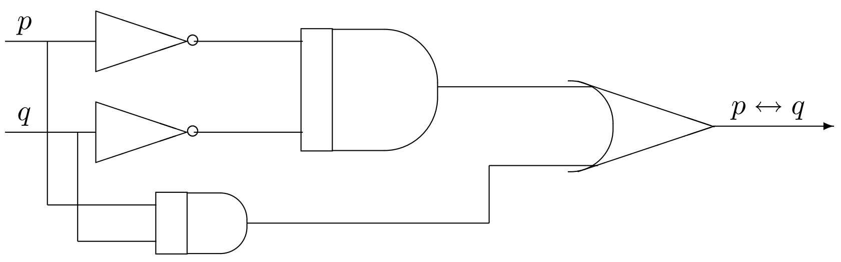


Figure 1.8 Circuit for $\leftrightarrow$.

These gates are combined together to form a bigger circuit which can perform many complicated jobs such as doing arithmetic and taking control jobs. You can, for example, construct a gate that does the job $\leftrightarrow$, by joining the basic gates together. The circuit in Figure 1.8 does this by employing the representation $p \leftrightarrow q \equiv (p \wedge q) \vee (\neg p \wedge \neg q)$. Note that the shape of a diagram component matters, but not its size.

Now, it is clear that for any intended gate, you can use one of cnf or dnf representations and then realize the gate. For example, suppose that you want to construct a circuit for realizing the truth function w as given in the following:

p	0	1	0	1	0	1	0	1
q	0	0	1	1	0	0	1	1
r	0	0	0	0	1	1	1	1
$w(p, q, r)$	0	1	1	1	1	1	1	0

The construction used in the proof of the normal form theorem gives:

$$\begin{aligned} w \equiv\ & (p \wedge \neg q \wedge \neg r) \vee (\neg p \wedge q \wedge \neg r) \vee (p \wedge q \wedge \neg r) \\ & \vee (\neg p \wedge \neg q \wedge r) \vee (p \wedge \neg q \wedge r) \vee (\neg p \wedge q \wedge r) \end{aligned}$$

***Exercise* 1.31** Draw the circuit for w as given above.

However, you can have a smaller circuit to do the same job as w. Grouping together the first and third clauses, second and sixth, and fourth and fifth, and using the laws, you get

$$w \equiv (p \vee q \vee r) \wedge (\neg p \vee \neg q \vee \neg r)$$

Since circuits will have to be implemented by actual wires (or metallic prints), it is preferable to have a circuit of smaller size which might do the same work than a crude one. This gives rise to the problem of minimization of Boolean circuits. There are many methods to do it; we will point out some bibliographic materials later.

***Exercise* 1.32** Draw the circuit diagram for w as given above and compare that with the one you have drawn in Exercise 1.31.

We will raise another problem in PL. Given a cnf, how would you determine whether it is satisfiable? You had already got a solution, where you require the law of distribution so that the cnf is converted to a dnf, and then check for complementary literals in the conjunctive clauses. Now, if the cnf has m clauses, with l_i number of literals in the i-th clause, then you need to write out, and check for complementarity in the $l_1 \times \cdots \times l_m$ clauses where each clause has m literals. In the worst case, it would require an exponential number of checks to determine satisfiability. So, it is no better than the truth table method. There have been many smart ways devised to tackle this problem, called SAT. Though they work better than exponential in most cases, the worst cases still require an exponential amount of labour. That is, we do not yet have an algorithm which would take a polynomial (in the length of the cnf) time for solving SAT. The collection of problems for which there is an algorithm to solve any instance in a polynomial time is denoted by $\mathcal{P}$.

In contrast, if you are told that here is one interpretation which is a model of the given cnf, you can check it in a polynomial time. You just have to evaluate the cnf under the interpretation. This class of problems is called $\mathcal{NP}$; it includes all those problems for which a suggested solution can be verified in a polynomial time. Intuitively, for a problem in $\mathcal{P}$, we are asked to get an algorithm to solve every instance in a polynomial time while in $\mathcal{NP}$, we are asked to check whether a suggested solution is indeed a solution, in a polynomial time. It is clear that SAT is in $\mathcal{NP}$, but it is not yet clear whether SAT is in $\mathcal{P}$. Moreover, a fundamental fact with SAT is that if it is found to be in $\mathcal{P}$, then both $\mathcal{P}$ and $\mathcal{NP}$ will coincide. Such problems are called $\mathcal{NP}$-complete problems.

A related problem is the so-called **k-SAT** problem. This is the same problem SAT but with the constraint that each of the clauses in the cnf has no more than k literals. It can be shown that corresponding to each cnf X, there exists a cnf Y where each clause in Y has no more than three literals. For example, a disjunctive clause with four literals, say, $(p \vee q \vee r \vee s)$ will be rewritten as $(p \vee q \vee x) \wedge (x \leftrightarrow r \vee s)$. Now, $(x \leftrightarrow r \vee s) \equiv (\neg x \vee r \vee s) \wedge (x \vee \neg r) \wedge (x \vee \neg s)$. Hence, $(p \vee q \vee r \vee s)$ can be rewritten as $(p \vee q \vee x) \wedge (\neg x \vee r \vee s) \wedge (x \vee \neg r) \wedge (x \vee \neg s)$. For any cnf, use this technique of rewriting repeatedly for the construction of a corresponding 3-cnf. Moreover, 3-SAT is also $\mathcal{NP}$-complete. With this result, we would then have: SAT is in $\mathcal{P}$ iff 3-SAT is in $\mathcal{P}$. Thus for deciding whether $\mathcal{P} = \mathcal{NP}$, it is enough to concentrate on **3-SAT**. If you have been mystified by this discussion, then fair enough; you need to look at the references suggested in the summary to this chapter.

There is, however, a subclass of propositions for which we have polynomial time algorithms for checking satisfiability. These are the Horn clauses, so called after the logician A. Horn. An arbitrary disjunctive clause in a cnf has some literals which appear with the symbol $\neg$, and some without. While in a **Horn clause**, there is at most one literal which is unnegated. For example, $\neg p \vee \neg q, r, \neg p \vee \neg q \vee r$ are Horn clauses, while $\neg p \vee \neg q \vee r \vee s$ and $r \vee s$ are not Horn clauses. Conventionally, Horn clauses are not written as disjunctive clauses since they can be written in another more suggestive way. For example, we might use the equivalences $\neg p \vee \neg q \equiv p \wedge q \to \bot$, $\neg p \vee \neg q \vee r \equiv p \wedge q \to r$ and $r \equiv \top \to r$ to rewrite the above Horn clauses.

Definition 1.10 A **Horn clause** is a proposition of the form

$$q_1 \wedge q_2 \wedge \cdots \wedge q_m \to q$$

where $q_1, \ldots, q_m, q$ are atomic propositions. A **Horn formula** is a conjunction of Horn clauses, written often as a set of Horn clauses.

Suppose that we have a Horn formula. If $\top \to q$ is a clause in it, then in any of its models, q must be true. If $q_1 \wedge \cdots \wedge q_m \to \bot$ is a clause in the Horn formula, and each of $q_1, \ldots, q_m$ is true in the already constructed model, then the clause cannot have a model. Thus the formula will be unsatisfiable. If there is a clause of the form $q_1 \wedge \cdots \wedge q_m \to q$, with $q \neq \bot$, and our model makes all of $q_1, \ldots, q_m$ true, then it must also make q true.

***EXAMPLE* 1.10** Is the following Horn formula satisfiable:

$$w = \{(p \wedge q \wedge r \to s), (t \to p), (\top \to t), (\top \to q), (u \to v), (\top \to u)\}$$

Solution Let us try to construct a possible model i of w. Since $\top \to q, \top \to u$ and $\top \to t$ are clauses, We must have $i(q) = i(u) = i(t) = 1$. Next, we have the clauses $t \to p$ and $u \to v$. If i is a model of these clauses, we must have $i(p) = i(v) = 1$. The clause $p \wedge q \wedge r \to s$ is satisfied by taking $i(s) = 1$. In the remaining clause $p \wedge q \wedge r \to \bot$, only p, q have been assigned some value, namely, 1. Now r can be assigned to 0 so that the Horn clause is satisfied. hence we have a model i of the Horn formula, where $i(p) = i(q) = i(s) = i(t) = i(u) = i(v) = 1$ and $i(r) = 0$. Do we have to consider the last two clauses?

However, we do not require to construct a model if only satisfiability is to be determined; see the procedure *HornSat* given below.

```
PROCEDURE : HornSat
  Input: A Horn formula $w$
  Output: $w$ is satisfiable or $w$ is not satisfiable

  1. If $\top \to \bot$ is a clause in $w$, go to Step 7.
  2. For every clause of the form $\top \to p$ in $w$, mark $p$ if it is a propositional
     variable.
  3. If $q_1 \wedge q_2 \wedge \cdots \wedge q_m \to \bot$ is a clause in $w$, where all of $q_1, q_2, \ldots, q_m$
     have been marked, then go to Step 7.
  4. If $q_1 \wedge q_2 \wedge \cdots \wedge q_m \to q$ is a clause in $w$, where all of $q_1, q_2, \ldots, q_m$
     have been marked and $q \neq \bot$, then mark $q$ and go to Step 3.
  5. If $q_1 \wedge q_2 \wedge \cdots \wedge q_m \to q$ is a clause in $w$, where $q \neq \bot$ and one or
     more of $q_1, q_2, \ldots, q_m$ have not been marked, then go to Step 6.
  6. Return '$w$ is satisfiable' and stop.
  7. Return '$w$ is not satisfiable' and stop.
```

***Exercise* 1.33** Which of the following are Horn clauses and why? For the Horn clauses, apply *HornSat* for determining their satisfiability.

(a) $(p \wedge q \wedge r \to \top) \wedge (p \wedge r \to s) \wedge (s \to \bot)$

(b) $(p \wedge q \wedge r \to \bot) \wedge (\neg p \wedge q \to r) \wedge (\bot \to s)$

(c) $\{p \wedge q \to s, p \wedge r \to p, r \to p \wedge t\}$

(d) $\{p \wedge q \wedge r \to \neg s, s \to \bot\}$

(e) $(p \wedge q \wedge r \to s) \wedge (\top \to s) \wedge (p \wedge q \to \bot)$

(f) $(\top \to p \wedge q) \wedge (\bot \to \top)$

Another application of PL is the knowledge systems. For example, in diagnosing a disease from the symptoms, we can have a data base, where information regarding a disease is stored as propositions. Then the task is to know whether a particular set of symptoms points to a disease. In such cases, the data base is called a knowledge base, a **propositional knowledge base**, and the the set of all conclusions that can be drawn from the base is called a **propositional theory**. Sometimes, the *propositional language* underlying the theory is also referred to as the propositional theory. When a case is presented to the theory, it is then required to ascertain whether a particular proposition follows from the theory. It may take a considerable amount of time to see whether such a consequence is valid. Thus the knowledge base, instead of just being stored, is first transformed to a better form so that particular consequences will be easier to decide. Such a transformation is done off-line, that is, before any suggested conclusion is presented to the theory. Such off-line transformation of the theory or the knowledge base is called **knowledge compilation**.

Methods of knowledge compilation depends upon the nature of the theory and our requirement on the on-line inference procedure. For example, a common approach to the knowledge compilation is to transform a set of propositions (the knowledge base) to the set of its **prime implicants** or the set of prime implicates. A prime implicant of a proposition is a conjunctive clause that implies the proposition with the property that no subclause of the clause implies the proposition. Similarly, a prime implicate of a proposition is defined dually, as a disjunctive clause that is implied by the proposition no subclause of which is implied by the proposition. It can be proved that the set (disjunction) of all prime implicants of a proposition is equivalent to the proposition. It can also be proved that the set of prime implicates of a proposition is equivalent to the proposition. Now, once the prime implicants of a propositional theory is obtained, any conclusion that can be drawn from the theory can equivalently be drawn from the set of prime implicants. However, drawing a conclusion from the set of prime implicants is easy, in the sense that a conclusion as a clause must have a subclause which is an element of the set of prime implicants. This activity of drawing an inference from a compiled knowledge base is an on-line activity. If we have to use some other on-line methods instead of checking for subclauses, then some other way of knowledge compilation or an off-line activity can be chosen.

SUMMARY

In this chapter, you have learnt that logic is concerned with methods of reasoning, and issues such as validity of arguments and formal aspects of truth and falsehood. You have seen that in propositional logic, the important connectives are $\neg$ (not), $\wedge$ (and), $\vee$ (or), $\rightarrow$ (implies), $\leftrightarrow$ (if and only if), $\uparrow$ (nand), and $\downarrow$ (nor). The effect of these connectives may be defined by means of truth tables. The truth tables define the meanings of the connectives unambiguously because the compound propositions are formed from the basic ones in an unambiguous manner by a formal construction of the language of propositional logic. Such a formal construction, where no concept of truth is attached to the constructs is termed as the **syntax** of the propositional logic.

In semantics, we attach truth values to the propositions by way of interpretations. A model is defined as an interpretation that evaluates a proposition to 1 (true). Propositions of which every interpretation is a model are called tautologies or valid propositions. A satisfiable proposition is one which has a model. The concept of models has been used to define a valid consequence which formalizes the notion of an argument. A consequence is valid if it is never the case that the premises are true and the conclusion is false.

Some basic tautologies and consequences have been separated out as laws. You have seen how to use these laws in proving validity of other propositions and consequences by calculations. Using calculations, you have also seen how to convert a proposition to its normal forms such as the cnf and the dnf. In this connection, you have also learnt that each of the pairs of connectives $(\neg, \wedge)$, $(\neg, \vee)$, $(\neg, \rightarrow)$ are adequate to express the other connectives. Each of the truth functions $\uparrow$ and $\downarrow$ also forms an adequate set. Finally, you have seen how to apply propositional logic to solve simple logical problems and to the construction of logical circuits. You have also been briefed about an important application of PL, namely, knowledge base systems. To pursue the topics further, the following bibliographic suggestions will be helpful.

Though propositional logic has a long history of more than 2000 years, the truth value semantics as discussed here was invented by George Boole [5] in the form of an algebra, now called Boolean algebra. It had been used by Frege [25] in a somewhat awkward fashion. The modern form of the truth table semantics was popularized by L. Wittgenstein [81] and E. Post [56], circa 1920. There had been debates about how well the *material implication* as given through the semantics of $\rightarrow$ represents the *implication* as found in natural languages. See [18, 51] for such discussions and other philosophical issues related to propositional logic. The calculational manner of proofs had been used by many in an informal way. For a formal proof procedure basing on calculations, see [35]. For a linear time algorithm for checking satisfiability of Horn formulas, see [13]. For algorithms and applications of SAT, see [14]. See [57] for hypergraph representations of satisfiability problem and also for many other algorithms for solving variants of SAT. You can also find in [57] many classes of SAT including the polynomial time decidability of 2-SAT and Horn-SAT . Consult [29] for more information on $\mathcal{NP}$-completeness and many problems including 3-SAT. As general references on propositional logic, see also the texts [6, 28, 41, 67] and the references therein. The works by R. Smullyan [72, 73] are nice collections of logical puzzles. For knowledge compilation and its application to various reasoning activities and minimization of Boolean circuits, many algorithms have been devised. The method of Karnaugh maps is one among them. The Karnaugh maps become quite involved when the number of propositional variables and the number of clauses become large. In our terminology this amounts to computing the prime implicants or prime implicates of a knowledge base. One of the oldest methods of computing prime implicants is Quine's algorithm [58]. For more information on the knowledge compilation techniques, see [12, 59, 65, 68, 77] and other net resources. Before doing all these, attempt the following problems.

PROBLEMS

1. Draw parse trees of the following expressions, and then decide which of them are propositions. Find all subpropositions of the propositions. [Treat p, q, r, s as propositional variables.]

(a) $(\neg\neg\neg r \wedge (\neg s \vee \neg(p \vee q)))$

(b) $\neg(p \vee (q \wedge (r \vee s) \wedge (q \wedge (p \wedge (r \to \neg q) \vee (r \leftrightarrow p)))))$

(c) $(((\neg r \vee (\neg p \wedge \neg q)) \to (r \to p)) \wedge \neg(p \leftrightarrow \neg(q \wedge \neg q)))$

2. Let $i(p) = i(q) = 1,\ i(r) = 0$. Draw parse trees of the following propositions, and then use the trees to evaluate the propositions under i.

(a) $\neg(p \vee \neg(p \vee \neg(q \leftrightarrow r)))$

(b) $((p \leftrightarrow \neg q) \to (r \to p \wedge q))$

(c) $((\neg p \vee \neg q) \leftrightarrow (\neg r \vee (p \to q)))$

(d) $(p \wedge (q \vee \neg p)) \vee q \vee (p \wedge (q \vee p))$

3. Give two parse trees that do not correspond to propositions, such that

(a) one of them could be extended, by adding subtrees at the leaves, to a parse tree which would correspond to a proposition.

(b) whatever way you extend the other, by adding subtrees at the leaves, it cannot correspond to a proposition; it is inherently ill formed.

4. Use induction on the number of occurrences of connectives in v to prove that for any propositions u, v, if u is a prefix of v, then $u = v$.

5. Let x, y, z be any three strings over the alphabet of PROP. Show that at most one of xy and yz is a proposition.

6. Replace the formation rules of PROP by: if u, v are propositions, then $\neg(u), (u) \wedge (v), (u) \vee (v), (u) \to (v), (u) \leftrightarrow (v)$ are propositions. Show that the unique parsing theorem holds with such a definition also.

7. Can you construct strings a, b and propositions u, v such that $a \neq u$, $b \neq v$ but $(a \wedge b) = (u \wedge v)$? What if $\wedge$ is replaced by $\rightarrow$?

8. In the proposition $(p \rightarrow q) \leftrightarrow p \wedge r$, the *main connective* is $\leftrightarrow$. The main connective in $\neg(p \rightarrow q \wedge r)$, is $\neg$. The main connective is the root of the parse tree. Define the main connective of a proposition in another way. Write a procedure to get the main connective of a proposition.

9. Suppose that we omit the right parentheses everywhere from a proposition. For example, instead of defining $(u \wedge v)$ as a proposition in our grammar, we define $(u \wedge v$ as a proposition. Similarly, for other connectives. Then will the unique parsing theorem hold under such a definition?

10. Which of the following are tautologies, contradictions, and contingent? For all the tautologies here, give also a calculational proof.

(a) $p \rightarrow (q \rightarrow p)$

(b) $(q \rightarrow p) \rightarrow p$

(c) $(\neg p \rightarrow \neg q) \rightarrow (q \rightarrow p)$

(d) $(p \rightarrow (q \rightarrow r)) \rightarrow ((p \rightarrow q) \rightarrow (q \rightarrow r))$

(e) $(((p \wedge q) \leftrightarrow p) \rightarrow q)$

(f) $((p \vee (p \wedge q)) \rightarrow (p \wedge (p \vee q)))$

(g) $(((p \vee q) \wedge (\neg q \vee r)) \rightarrow (p \vee r))$

(h) $(((p \wedge q) \wedge (\neg q \vee r)) \rightarrow (p \wedge r))$

(i) $((p \leftrightarrow q) \leftrightarrow r) \leftrightarrow ((p \leftrightarrow q) \wedge (q \leftrightarrow r))$

(j) $((p \wedge q \rightarrow r) \wedge (p \wedge \neg q \rightarrow r)) \leftrightarrow (p \rightarrow (q \leftrightarrow r))$

11. Simplify the following propositions:

(a) $(p \wedge (q \vee \neg p)) \vee q \vee (p \wedge (q \vee p))$

(b) $\neg q \rightarrow \neg(q \rightarrow \neg p)$

(c) $p \wedge (\neg p \rightarrow p)$

(d) $((p \vee (p \wedge q)) \rightarrow (p \wedge (p \vee q)))$

(e) $((p \leftrightarrow q) \leftrightarrow r) \leftrightarrow ((p \leftrightarrow q) \wedge (q \leftrightarrow r))$

(f) $\neg p \wedge q \wedge (p \rightarrow (q \vee \neg r))$

12. Prove or give a counter example; u, v are any propositions:

(a) If $u \wedge v$ is satisfiable, then u is satisfiable and v is satisfiable.

(b) If $u \vee v$ is satisfiable, then u is satisfiable or v is satisfiable.

(c) If $u \rightarrow v$ is valid and u is valid, then v is valid.

(d) If $u \rightarrow v$ is satisfiable, and u is satisfiable, then v is satisfiable.

(e) If $u \wedge v$ is valid and u is satisfiable then v is satisfiable.

(f) If $\models w$, then w must have at least two atomic propositions.

(g) If $\models w$, and w has an occurrence of a propositional variable, then w has at least two occurrences of a propositional variable.

(h) If w is a contradiction, then the main connective of w must be one of $\neg$ or $\wedge$.

13. Give models for each of the following sets of propositions:

(a) $\{p_i \vee p_{i+1}, \neg(p_i \wedge p_{i+1}) : i \in \mathbb{N}\}$

(b) $\{(p_i \vee p_{i+1}), (p_i \to \neg(p_{i+1} \wedge p_{i+2})), (p_i \leftrightarrow p_{i+3}) : i \in \mathbb{N}\}$

(c) $\{\neg p_1, p_2\} \cup \{(p_i \wedge p_j \leftrightarrow p_{i.j}) : 1 < i, j\}$

14. Are the following consequences valid? Justify.

(a) $p \vee \neg q, p \to \neg r \models q \to \neg r$

(b) $\neg(r \wedge \neg\neg q) \models (\neg q \vee \neg r)$

(c) $p \vee q \to r \wedge s, t \wedge s \to u \models p \to u$

(d) $p \vee q \to r \wedge s, s \vee t \to u, p \vee \neg u \models p \to (q \to r)$

(e) $p \to q \wedge r, q \to s, d \to t \wedge u, q \to p \wedge \neg t \models q \to t$

(f) $p,\ \neg r \to \neg p,\ (p \to q) \wedge (r \to s), (s \to u) \wedge (q \to t),\ s \to \neg t \models \bot$

15. Are the following sets of propositions satisfiable? Justify.

(a) $\{\neg p \vee \neg(q \wedge r), s \vee t \to u, u \to \neg(v \vee w), r \wedge v\}$

(b) $\{p \to q, r \to s, q \to s, \neg r \to p, t \to u, u \to \neg s, \neg t \to t\}$

(c) $\{p \vee q \to r \wedge s, s \vee t \to u, p \vee \neg u, p \to (q \to r)\}$

(d) $\{p \to q \wedge r, q \to s, d \to t \wedge u, q \to p \wedge \neg t, \neg q \to t\}$

(e) $\{p \to q \wedge r, q \to s, d \to t \wedge u, q \to p \wedge \neg t, q \to t\}$

(f) $\{p \to q \wedge r, s \to q \wedge t, u \to \neg p, (v \to w) \to u \wedge s, \neg(\neg r \to t)\}$

16. Let w be a proposition having the only connective as $\leftrightarrow$; and having no occurrence of $\top$ or $\bot$. Show that $\models w$ iff each propositional variable occurring in w occurs an even number of times. What happens if $\top$ or $\bot$ occurs in w?

17. Are the following arguments valid? Justify.

(a) If A is a knight, then B is a knight. If C is a knave, then B is a knight. Therefore, if A is a knight or C is a knave, then B is a knight.

(b) If A is a knave, then B is a knave. If C is a knight, then B is a knave. Therefore, if A is a knave and C is a knight, then B is a knave.

(c) If Sam was at the fair, then his father was negligent or his mother was not at home. If his mother was not at home, then his father was not negligent. His mother was at home. Therefore, Sam was at the fair.

(d) If Shaktiman were able and wiling to eradicate evil, he would do so. If Shaktiman were not able to eradicate evil, he would be transformed into a donkey. If Shaktiman were unwilling to eradicate evil, he would be joining the evil. As a matter of fact, Shaktiman does not eradicate evil. Therefore, if Shaktiman exists, then he is neither transformed into a donkey nor does he join the evil.

(e) Either the program does not terminate or m is eventually 0. If n is eventually 0, then m also becomes eventually 0. The program is known not to terminate. Hence m is eventually 0.

(f) All of x, y, z cannot be negative. If they were, then x would be less than both y and z. Hence x is not less than one of y or z.

(g) If the initialization is correct and the loop terminates, then the required output is obtained. The output has been obtained. Therefore, if the initialization is correct, the loop must terminate.

(h) If 2 is a prime, then it is the least prime. If 1 is a prime, then 2 is not the least prime. The number 1 is not a prime. Therefore, 2 is the least prime.

18. In outfix notation the proposition $\neg p \to (q \vee (r \wedge s))$ is written as $\to (\neg p, \vee(q, \wedge(r, s))$. If you omit the commas and parentheses, it will look like $\to \neg p \vee q \wedge rs$. Can you see how to put commas and parentheses into this expression so that you obtain the earlier outfix notation? Taking clues from this, define a language of propositional logic without parentheses. Show that unique parsing still holds. (This is called the *Polish* notation.)

19. Construct a cnf and a dnf for the truth function u, v given by the following truth table. Simplify the normal forms and then draw circuits representing the truth functions.

p	q	r	u	v
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	1
1	0	0	0	0
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

20. Let $f : \{q_1, \ldots, q_m\} \to \{0, 1\}$, where $q_1, \ldots, q_m$ are distinct atomic propositions. Show that there is a proposition w such that for any function $g : \{q_1, \ldots, q_m\} \to \{0, 1\}$, you have $g \models w$ iff $f = g$.

21. Construct a proposition w involving three atomic propositions p, q, r such that for any interpretation, changing any one of the truth values of p, q, or r changes the truth value of w.

22. Let $w = (((p \wedge q) \vee (p \vee r \vee q)) \wedge (\neg p \wedge \neg r \wedge q))$. Then show that $\neg w \equiv (((\neg p \vee \neg q) \wedge (\neg p \wedge \neg r \wedge \neg q)) \vee (p \vee r \vee \neg q))$. Generalize this exercise to compute the negation of any proposition involving the only connectives $\neg, \wedge, \vee$.

23. Express all the $\downarrow$ through $\uparrow$, and $\uparrow$ through $\downarrow$.

24. Instead of using truth tables, an arithmetic procedure can be used. The basis for this is the following representation of the connectives: $\neg w$ is represented as $1 + w$, $u \wedge v$ as $u + v + uv$, $u \vee v$ as uv, $u \to v$ as $(1 + u)v$ and $u \leftrightarrow v$ as $u + v$. The constants $\top$ and $\bot$ are taken as 0 and 1, respectively. Now in the representation, each propositional variable is treated as a variable over the set $\{0, 1\}$. The arithmetic operations of addition and multiplication are taken as usual with one exception, that is, $1 + 1 = 0$. Thus tautologies are those formulas which are identically 0 and contradictions become identically equal to 1. See that in this algebra, you have $2x = 0$ and $x^2 = x$. Justify the procedure. Use this procedure to prove all the laws in Theorem 1.8.

25. See Problem 24. Give another arithmetic representation of the connectives where $\top$ is represented as 1 and $\bot$ as 0.

26. Use Problems 24 and 25 to have representations of the truth functions $\uparrow$ and $\downarrow$. Show that every function from $\{0, 1\}^n$ to $\{0, 1\}$ can be generated from the mapping $f : \{0, 1\}^2 \to \{0, 1\}$, where $f(x, y) = (1 + x)(1 + y)$. Show that the same is true of the mapping $g : \{0, 1\}^3 \to \{0, 1\}$ given by $g(x, y, z) = 1 + x + y + xyz$. Can you find another function from $\{0, 1\}^2$ to $\{0, 1\}$ which behaves like f?

27. Assume that any politician is either honest or dishonest. An honest politician, by definition, always tells the truth, and a dishonest politician always lies. In a campaign rally, a politician declares "I am honest iff my opponent is honest".

(a) Let p be "The first politician is honest", and q be "The opponent of the first politician is honest". Explain why $p \leftrightarrow (p \leftrightarrow q)$ holds.

(b) Is the first politician honest?

(c) Is the opponent of the first politician honest?

(d) Later the opponent of the first politician declares, "If I am honest, so is my opponent". What new formulas now become true?

(e) Using the statements of both the politicians, answer (b) and (c).

28. Explain why you are trapped in the commercial given in the Preface to this book.

29. In *Merchant of Venice*, Portia takes three caskets: gold, silver, and lead. Inside one of the caskets, she puts her portrait, and on each of the locked caskets, she writes an inscription. She explains to her suitors that each inscription could either be true or false. But on the basis of the inscriptions, one has to choose the casket containing the portrait. We will have two variations of the puzzle here.

(a) Suppose that Portia writes the inscriptions as

Gold: The portrait is in here.
Silver: The portrait is in here.
Lead: At least two caskets bear false inscriptions.

Which casket should the suitor choose? First, find the answer arguing informally, and then symbolize using the following abbreviations:

G: The portrait is in the gold casket.
g: The inscription on the gold casket is true.

Similarly, use the abbreviations S, s, L, l, and argue formally. [*Hint*: The inscription on the gold casket says that G $\leftrightarrow$ g.]

(b) This time Portia says that one has to choose a casket *not* containing the portrait, basing on the following inscriptions.

Gold: The portrait is in here.
Silver: The portrait is not in here.
Lead: At most one casket bears a true inscription.

30. Three inhabitants of the island of Kniknaord (see Example 1.9), A, B, C were standing together. You asked A, "Are you a knight or a knave?" You could not hear the low voice of A. You asked B, "What did A say?" B replied, "A said that he was a knave." Then C said, "Do not believe B, he is lying." If none of A, B, C is an ordinary person, then what are they, knights or knaves?

31. Three inhabitants of the island of Kniknaord (see Example 1.9), A, B, C were standing together. You asked A, "How many knights are among you?" Again you could not listen to A. You asked B, "What did A say?" B replied, "A said, there was only one knight among us." C said, "Do not believe B, he is lying." If none of A, B, C is an ordinary person, then what are they, knights or knaves?

32. Define the *depth* of a proposition $d(w)$ recursively by:

$$d(p) = 0, \quad \text{for any atomic proposition } p$$

$$d(\neg w) = 1 + d(w), \quad \text{for any proposition } w$$

$$d((u * v)) = 1 + \max(d(u), d(v)), \quad \text{for any propositions } u, v, \text{ and any binary connective } * \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$$

Does $d(w)$ coincide with the depth of the parse tree for w?

33. The grammar in BNF for the set of literals can be written as $l ::= p \mid \neg p$, where p stands for atomic propositions. Give grammars in BNF for conjunctive clauses, disjunctive clauses, cnf, and dnf.

34. Prove: If w is a proposition of which there are exactly n number of models for some $n > 0$, then there is a dnf representation of w which is a disjunction of exactly n conjunctions of literals. Can you formulate a similar statement for a cnf representation of w?

35. A set Σ of propositions is an *independent set* if for every $w \in \Sigma$, we have $\Sigma - \{w\} \not\models w$. Prove that every finite set Σ of propositions has a subset Γ which is independent and that if $w \in \Sigma$, then $\Gamma \models w$.

36. Define a function $sp : \text{PROP} \to 2^{\text{PROP}}$, (where 2^{PROP} is the set of all subsets of PROP) by:

$$sp(p) = \{p\}, \quad \text{for any atomic proposition } p$$
$$sp(\neg w) = sp(w) \cup \{\neg w\}, \quad \text{for any proposition } w$$
$$sp(u * v) = sp(u) \cup sp(v) \cup \{u * v\}, \quad \text{for any propositions } u, v, \text{ and any binary connective } * \in \{\wedge, \vee, \to, \leftrightarrow\}$$

Prove that if a proposition w has n occurrences of connectives, then $sp(w)$ has $2n + 1$ occurrences of connectives. Is the set $sp(w)$ the set of all sub-propositions of w?

37. In this exercise, assume that all our propositions are defined in terms of the only connectives $\neg, \wedge, \vee$. Define the *dual* dl and *denial* dn of a proposition recursively by the following (let p stand for atomic propositions and others for any propositions):

$$dl(p) = p, \; dl(\neg w) = \neg dl(w),$$
$$dl(u \wedge v) = dl(u) \vee dl(v), \; dl(u \vee v) = dl(u) \wedge dl(v).$$
$$dn(p) = \neg p, \; dn(\neg p) = p, \; dn(\neg w) = \neg dn(w),$$
$$dn(u \wedge v) = dn(u) \vee dn(v), \; dn(u \vee v) = dn(u) \wedge dn(v).$$

Show that (a) $x \equiv y$ iff $dl(x) \equiv dl(y)$, (b) $dn(z) \equiv \neg z$.

38. Let $\not\leftrightarrow$, called the *XOR* or the *exclusive or*, be the truth function defined by $x \not\leftrightarrow y \equiv \neg(x \leftrightarrow y)$. Attempt to show that

(a) $\not\leftrightarrow$ is both commutative and associative.
(b) $w \not\leftrightarrow \bot \equiv w, \quad w \not\leftrightarrow \top \equiv \neg w, \quad w \not\leftrightarrow w \equiv \bot$
(c) $u \vee (v \not\leftrightarrow w) \equiv (u \vee v) \not\leftrightarrow (u \vee w), \quad u \wedge (v \not\leftrightarrow w) \equiv (u \wedge v) \not\leftrightarrow (u \wedge w)$
(d) $\{\not\leftrightarrow, \wedge, \top\}$ is an adequate set of connectives.
(e) Can you express any of $\not\leftrightarrow, \wedge, \top$ through the other two?
(f) Is $\{\not\leftrightarrow, \vee, \top\}$ an adequate set?

39. See Problem 38. Prove that every proposition is equivalent to either $\bot$ or $\top$ or $C_1 \not\leftrightarrow \cdots \not\leftrightarrow C_m$, where each C_i is either $\top$ or a conjunction of propositional variables. This is called the *exclusive or normal form*. What are the exclusive or normal forms of tautologies?

40. The *negation normal form*(nnf) is defined recursively by:

If p is a propositional variable, then both p and $\neg p$ are in nnf;

If u and v are in nnf, then $u \wedge v$ and $u \vee v$ are in nnf.

Show that every proposition is equivalent to one in nnf. Construct a grammar for nnf. Give a procedure to obtain an nnf equivalent to a proposition.

41. Let x, y be propositions having no common propositional variables. Show that if $\models x \to y$, then x is unsatisfiable or y is valid.

42. *Craig's interpolation theorem:* Let x, y be propositions having at least one propositional variable in common. A proposition z is called an *interpolant* of $x \to y$ iff $\models x \to z$, $\models z \to y$, and all propositional variables occurring in z are among the common propositional variables of x, y. Show that if $\models x \to y$, then x is unsatisfiable or y is valid or $x \to y$ has an interpolant.

43. Apply *HornSat* to the following Horn formulas:

(a) $\{p \to q, r \wedge s \wedge t \to u, \top \to t, t \wedge q \to \bot\}$

(b) $\{p \wedge q \wedge s \to p, q \wedge r \to p, p \wedge s \to s, s \wedge r \to t\}$

(c) $\{p \wedge q \wedge r \to \bot, s \to p, t \to \bot, \top \to s, \top \to q, u \to v, \top \to u\}$

44. Try to show that the procedure *HornSat* correctly checks whether a given Horn formula is satisfiable. [*Hint*: You may have to use induction on the number of times the loop in Steps 3 and 4 is executed in running the procedure.]

45. Explain why the procedure *HornSat* fails if we define a Horn clause as $q_1 \wedge q_2 \wedge \cdots \wedge q_m \to q$ admitting q_i's to be any literal. Can you specify a syntactic criterion on cnfs so that they will be equivalent to Horn formulas? If you answer 'no', why? If you answer 'yes', can you write a procedure to convert such special cnfs to Horn formulas?

46. Can you use grammars in BNF to define Horn formulas? Do you require any auxiliary category so that this can be done?

2

First Order Logic

2.1 Introduction

Consider the following argument:

Bapuji was a saint.

Since every saint is an altruist, so was Bapuji.

How do you symbolize this argument in PL? Clearly, there are three declarative sentences here and no connectives have been used. So each has to be symbolized as a different proposition. Writing

p: Bapuji was a saint

q: Every saint is an altruist

r: Bapuji was an altruist

you would symbolize the argument as the consequence: $\{p, q\} \models r$. But then this consequence is invalid, as you can have an interpretation i with $i(p) = 1, i(q) = 1, i(r) = 0$. But what do you feel about the argument? It seems to be correct. Is'nt it? So?

The answer is clear. When we say that the argument seems to be correct, we think the sentences in terms of relations. The first sentence, "Bapuji was a saint" tells something like: possibly, there could be many saints, and Bapuji was one of them. That is, the property of 'being a saint' could be applied truthfully to 'Bapuji'. So, we are going deeper into the form of a sentence rather than taking it as a single unit. The units in this sentence are a name, 'Bapuji', a predicate, a relation, or a property, 'is a saint'. In the third sentence, the same thing happens. You have two units again, a name, 'Bapuji', and a predicate, ' is an altruist'. What about the second sentence? Here, of course, there are two units, 'is a saint' and 'is an altruist'; but both are predicates. You can translate this by introducing a false name, called a variable, say, x. It would look like:

For every x, if x is a saint, then x is an altruist.

Here, the variable x is not a name, but it can be replaced by any name; it is a place holder, a gap. The name of the variable is, of course, x. That is, it is a named gap, a variable that stands for any object whatsoever in contrast to a name which is a particular object. Next, in this sentence, we have a quantifier, namely, 'every'. This is neither a name, nor a property, nor a variable. It only says how much of the scope of the variable x is involved; it quantifies over the variable. In the sentence,

Bapuji liked some Russian author

the word 'some' is also a quantifier. Can you rewrite the sentence more explicitly by using a variable? It would look like:

For some x, x was a Russian author and Bapuji liked x.

Can you identify the names, predicates, and quantifiers in the sentence:

Bapuji liked the author of *War and Peace*?

Here, 'Bapuji' is a name, 'liked' is a predicate taking two arguments (x liked y). And what about the phrase, 'the author of *War and Peace*'? Is it a name? Of course, you can take it as a name since it refers to a particular object. But we say something else rather than just regarding it as a name. In the phrase, 'the author of *War and Peace*', we seem to use the expression *War and Peace* as a name, the name of a particular book; the phrase 'the author of' is used as a different unit, something like a predicate. But it differs from a predicate in a significant way. Look, 'the author of', if used for a book, would give rise to a single object, a person here, whereas a predicate would give rise to a sentence when its argument is filled in with a name. The phrase 'the author of *War and Peace*' thus is called a definite description; it describes a person, an object in a definitive manner. Can you translate the above sentence as

Bapuji liked x and x was the author of *War and Peace*

Or as

For some x, Bapuji liked x and x was the author of *War and Peace*?

The unique sort of reference by a definite description is not captured by the predicates. We would view a definite description something like a function. If you consider the phrase 'the author of' as a function, then clearly, 'the author of *War and Peace*' is simply the value of the function 'the author of' at the object *War and Peace*. Moreover, it is a function of one argument. There can be functions of many arguments, for example, 'the eldest daughter of Mr. X and Mrs. X'. It takes two arguments, 'Mr. X' and 'Mrs. X'. The phrase 'a Russian author' is a predicate, and not a function. Similarly, 'mother of' is a function while 'brother of' is not a function, but a relation, and thus 'is a brother of' can be thought of as a predicate. One more example:

Both *War and Peace* and *Anna Karenina* were authored by the same person.

How do you rewrite it? You may take *War and Peace* as a name, *Anna Karenina* as a name, 'authored by' as a binary predicate (admitting of two arguments), and then 'are the same person' as another predicate. However, there is a difference between this predicate and other predicates. The predicate 'authored by' may be interpreted in various ways, pointing to 'the person who actually wrote', or 'the person whose name appears on the title page', etc, the phrase 'x is same as y' will always be interpreted as "both x and y refer to the same object". We will reserve a particular symbol for this special binary predicate. Similarly, we will have some special symbols for the two quantifiers (we will not have any other quantifiers in first order logic) 'for all' and 'for some'. Here are our special symbols:

$x \approx y$:	x is same as y, the identity predicate
$\forall x$:	for every x, or, for all x, or, for each x
$\exists x$:	for some x, there exists (at least one) x

with a vocabulary for names, predicates, function symbols such as

a :	*Anna Karenina*
b :	Bapuji
c :	*War and Peace*
$f(x)$:	the author of x
$L(x, y)$:	x likes y
$R(x)$:	x is a Russian author
$S(x)$:	x is a saint
$T(x)$:	x is an altruist.

The above sentences can be symbolized as (ignoring the tense):

Bapuji was a saint:	$S(b)$
Every saint is an altruist:	$\forall x(S(x) \to T(x))$
Bapuji was an altruist:	$T(b)$
Bapuji liked a Russian author:	$\exists x(L(b, x) \wedge R(x))$
Bapuji liked the author of *War and Peace*:	$L(b, f(c))$

The authors of *War and Peace*, and *Anna Karenina* were the same:

$$f(c) \approx f(a)$$

The logic thus obtained (in fact, to be obtained) is now capable of going a bit deeper into the sentences. However, the quantifiers quantify over variables which stand for objects only; we cannot quantify over the functions, nor over the predicates, and hence the name, First Order Logic (FL). The second order logic takes care of quantification over functions and predicates, which we are not defining here. First Order Logic is also called Predicate Logic with function symbols and with equality or identity. Then, Aristotelian logic is simply the monadic predicate logic without function symbols, a logic where each predicate takes only one argument and there are no function symbols. Naturally, the equality or the identity predicate is absent in the Aristotelian logic as it is a binary predicate.

2.2 Syntax of FL

So we have decided to have our alphabet to contain variables, function symbols, predicates, names, and quantifiers. We will also have the usual connectives. The set of **variables** is denoted as $\mathcal{V}$, the set of **function symbols** as $\mathcal{F}$, the set of **predicates** as $\mathcal{P}$, the set of **quantifiers** as $\mathcal{Q}$, the set of **connectives** as $\mathcal{C}$, the special propositions representing truth and falsity as $\mathcal{T}$, and the extra symbols, the set of **punctuation marks** as $\mathcal{E}$. The sets are:

$$\mathcal{V} = \{x_0, x_1, x_2, \ldots\}$$

$$\mathcal{F} = \{f_i^j : i, j \in \mathbb{N}\}$$

$$\mathcal{P} = \{P_i^j : i, j \in \mathbb{N}\} \cup \{\approx\}$$

$$\mathcal{Q} = \{\forall, \exists\}$$

$$\mathcal{C} = \{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$$

$$\mathcal{T} = \{\top, \bot\}$$

$$\mathcal{E} = \{),\ (,\ ,\}$$

Look at the subscripts and superscripts in the function symbols and predicates. The symbol f_i^j is a function symbol which could have been written as just f_i ; the superscript j says that the function symbol f_i^j has j arguments. For example, $f_1^1(x_0)$ is allowed as this function symbol has only one argument and here the only argument, the gap, is filled with a variable x_0 while $f_1^2(x_0)$ will not be allowed since this function requires two arguments so that when these are filled in, it may stand for a definite description. Thus the definite description 'author of' will be denoted by a function having a superscript 1 and not 2 or any other natural number. The superscript j in the function symbol f_i^j is referred to as its **arity**. The 0-ary function symbols do not require any variable or names to give us definite expressions, that is, they themselves are definite expressions or names. Thus **names**, also referred to as **individual constants** or just **constants**, are simply taken as 0-ary function symbols. Similarly, the superscripts in the predicates also refer to the arity of those. The relation 'brother of' is a binary predicate, as we would use it in the form 'x is a brother of y', and as such it will be denoted by a predicate P_i^2; notice the superscript here, the subscript may be any of $0, 1, 2, 3, 4, \ldots$, it does not matter. Just like 0-ary function symbols, the 0-ary predicates do not have any gaps to be filled in so that they would become sentences; they are sentences already. Thus, 0-ary predicates P_i^0's are simply the propositional variables which, in some contexts, may not be analyzed any deeper. This way, we really extend the syntax of PL to FL. We also have the special symbol $\approx$ for denoting the **equality** or **identity** predicate, assumed to be binary.

The alphabet of FL is $\mathcal{A} = \mathcal{V} \cup \mathcal{F} \cup \mathcal{P} \cup \mathcal{Q} \cup \mathcal{C} \cup \mathcal{T} \cup \mathcal{E}$.

Any string over $\mathcal{A}$ is an **expression** (an FL-expression). The function symbols allow you to express very complex definite expressions such as 'mother of mother of mother of father of Gargi', by using composition of functions. Thus definite descriptions may use any number of function symbols and their compositions. All these definite descriptions, which are only special expressions are called **terms**. The following is a recursive definition of terms.

Definition 2.1 Write t for a generic term. The grammar for **terms** is

$$t \ ::= \ x_i \mid f_i^0 \mid f_i^j(t, t, \ldots, t) \qquad (j\text{−times } t \text{ here})$$

All it says is that the variables, and the names or 0-ary function symbols are terms, and if $t_1, \ldots, t_j$ are terms, and f_i^j is a $j-$ary function symbol, then the expression $f_i^j(t_1, \ldots, t_j)$ is a term. Moreover, terms are obtained only this way, from 0-ary function symbols or other terms used as arguments in other function symbols of appropriate arity. For example, f_5^0 is a 0-ary function symbol (an individual constant or name). By definition, it is a term. We often write names as they occur in natural languages, such as 'Bapuji', 'Rajaji', 'Tolstoy' et al. Also, 'father of Bapuji' is a term, accepting 'father of' as a unary function symbol. However, in our formal language, we will not admit these expressions as terms; they have to be symbolized as some function symbols. For example, suppose we symbolize 'Bapuji' as f_0^0 and 'father of' as f_0^1, then the definite description 'father of Bapuji' will be symbolized as $f_0^1(f_0^0)$. Similarly, $f_5^1(f_3^2(f_0^0, f_1^0))$ is a term; check the arities. We will use terms for defining the (well-formed) formulas, which would represent sentences or rather sentence-like phrases.

Definition 2.2 Writing X for a generic formula and s, t, t_i for terms, the grammar for **formulas** is:

$$X \ ::= \ \top \mid \bot \mid P_i^0 \mid (s \approx t) \mid P_i^m(t_1, t_2, \ldots, t_m) \mid \neg X \mid (X \wedge X) \mid$$
$$(X \vee X) \mid (X \to X) \mid (X \leftrightarrow X) \mid \forall x_i X \mid \exists x_i X$$

It says that the special symbols $\top, \bot$, and the 0-ary predicates are (FL) formulas. Moreover, if X is any m-ary predicate, then for m terms $t_1, \ldots, t_m$, the expression $X(t_1, \ldots, t_m)$ is also a formula. The formulas might be obtained by using connectives as in PL or by prefixing any other formula to a quantifier followed by a variable. Note that the 0-ary predicates are the propositional variables, and they themselves are formulas. In fact, all propositions of PL are now treated as FL-formulas. For example, the following expressions are formulas:

$$\top$$
$$(\bot \to \top)$$
$$(f_1^0 \approx f_5^0)$$
$$(f_1^2(f_1^0, f_2^1(f_1^0)) \approx f_{11}^0)$$
$$P_2^1(f_1^1(x_5))$$
$$\neg\forall x_3(P_5^2(f_1^1(x_5), f_1^0) \to P_3^0)$$
$$\forall x_2 \exists x_5(P_5^2(x_0, f_1^1(x_1)) \leftrightarrow P_1^3(x_1, x_5, x_6))$$

whereas the following expressions are not formulas:

$$\top(x_0)$$
$$\approx (f_1^0(f_0^0))$$
$$f_1^0 \approx f_5^0$$
$$(f_1^2(f_1^0, f_2^0(f_1^0)) \approx f_{11}^2(f_{12}^1, f_1^0))$$
$$P_2^1(f_4^1(x_7), f_1^0)$$
$$\neg\forall x_1(P_5^2(f_1^0(x_2), x_3))$$
$$\forall x_2 \exists x_5(P_5^2(x_0, f_1^0(x_1)) \leftrightarrow P_1^3(x_1, x_5, x_6))$$

The first expression is not a formula since after $\top$, a variable is not allowed to occur in parentheses. The second expression is not a formula since $f_1^0(f_0^0)$ is not a term, as f_1^0 is a 0-ary function symbol and as such it cannot take an argument. The third expression is not a formula since a $\approx$ needs a pair of parentheses. In the fourth expression, f_{12}^1 needs a term as an argument. Similarly, in the fifth expression the predicate P_2^1 can have only one argument, and in the sixth and seventh, f_1^0 cannot take any term as an argument.

The language we are dealing with requires agreements, specially the superscripts of the function symbols, and predicates must match with the number of terms they take as arguments. Similarly, parentheses must match. Now, you must be able to guess the reason for such agreements. Yes, it is uniqueness of parsing; our grammar must be unambiguous. We must be able to find out the immediate subformulas of any formula, and the main connective or quantifier. If a formula X starts with $\neg$, then it is the rule $X ::= \neg X$ that has been applied last. Similarly, if it starts with (, then it is one of the binary connective rules or the equality predicate rule that had been applied last. If it starts with a P_i^j, then the last rule applied was $X ::= P_i^0$, and hence j must be equal to 0. These observations (and some more) will show that formulas are uniquely parsed. Of course, you need to construct a formal proof. The proof will also require you to show that every term is uniquely parsed.

***Exercise* 2.1** Give a formal proof of unique parsing for formulas of FL. Show that given any expression of FL, it can be determined whether the expression is a formula or not. Write procedures for determining and parsing formulas.

Once you are sure that unique parsing holds, we will follow some conventions for using less number of parentheses, and sometimes, commas. *First*, we will drop the outer parentheses from formulas. *Second*, we will drop the superscripts from the predicates and function symbols, for the information regarding them can be supplied additionally in a context; but then, we must check that the same symbol is not used with different number of arguments in any particular context. For example, in the same formula, we must not use $P_5(x, y)$ and $P_5(f_2)$ since the first one says that P_5 is a binary predicate, and the second would require P_5 to be unary. We must avoid such confusions. In the formula $\forall x_2\,\exists x_5(P_4(x_0, f_1(x_1)) \leftrightarrow P_1(x_2, x_5, x_1))$, we would take P_4 a binary predicate, f_1 as a unary function symbol, and P_1 as a ternary predicate. But we cannot possibly say that $\forall x_2\,\exists x_5(P_4(x_0, f_1(x_1)) \leftrightarrow P_4(x_2, x_5, x_1))$ is a formula since P_4 has been used once as a binary and once as a ternary predicate. With this convention, we agree to use different predicate symbols for those two instances.

Third, we will drop writing subscripts with variables, function symbols and predicates whenever possible. Instead, we will use $x, y, z, \ldots$ for variables, $f, g, h, \ldots$ for function symbols, and $P, Q, R, \ldots$ for predicates. Whenever we feel short of symbols, we may go back to writing them with subscripts. Moreover, the 0-ary function symbols (which stand for names) will be written as $a, b, c, \ldots$, from the first few small letters of the Roman alphabet. For example, the formula

$$\forall x_2\,\exists x_5(P_4(x_0, f_1(x_1)) \leftrightarrow P_1(x_2, x_5, x_1))$$

may be rewritten as

$$\forall x \exists y(P(z, f(u)) \leftrightarrow Q(x, y, u))$$

following this convention. Take care to see that each occurrence of x_2 has been replaced by x, etc.

Fourth, we will omit the parentheses and commas in writing the arguments of function symbols and predicates provided no confusion arises. If, however, some formula written this way is not easily readable, we will retain some of them. For example, $\forall x \exists y(P(z, f(u)) \leftrightarrow Q(x, y, u))$ can be rewritten as $\forall x \exists y(Pzf(u) \leftrightarrow Qxyu)$. Similarly, the term $f(t_1, \ldots, t_n)$ will sometimes be written as $f(t_1 \ldots t_n)$ or as $ft_1 \ldots t_n$. With this convention, the formula $\forall x \exists y(Pzf(u) \leftrightarrow Qxyu)$ is rewritten as $\forall x \exists y(Pzfu \leftrightarrow Qxyu)$.

Fifth, we will have precedence rules for the connectives and quantifiers to reduce parentheses. We will preserve the precedence of connectives as in PL and give the same precedence to the quantifiers as $\neg$. That is, $\neg, \forall x, \exists x$ will have the highest precedence; $\wedge, \vee$ will have the next precedence; and

$\to, \leftrightarrow$ will have the lowest precedence. For example, the formula

$$\forall x_1 \neg (\exists x_2 ((P_1^1(f_1^2(f_0^0, f_1^0)) \wedge P_1^0) \to P_2^2(x_2, f_1^0))$$
$$\leftrightarrow \forall x_3 ((P_1^1(f_5^0) \wedge P_2^1(x_1)) \to P_3^0))$$

is written as

$$\forall x \neg (\exists y (Pf(a,b) \wedge A \to Qyb) \leftrightarrow \forall z (Pc \wedge Rx \to B))$$

where the variables x_1, x_2, x_3 are rewritten as x, y, z; the function symbols $f_0^0, f_1^0, f_5^0, f_1^2$ are rewritten as a, b, c, f; and the predicates $P_1^1, P_1^0, P_2^2, P_2^1, P_3^0$ are rewritten as P, A, Q, R, B, respectively. Hereafter, we follow this convention of rewriting the variables, function symbols, and predicates.

The rewritten forms of formulas are, strictly speaking, not formulas. But we will regard them as formulas since they can be written back as formulas by following our convention. We will consider some examples of symbolizing English sentences into FL. Of course, we cannot go beyond the declarative sentences, but now we are able to break the sentences into subtler units involving names, definite descriptions, relations among them, and particularly, quantifiers such as 'for all', and 'there exists'.

***EXAMPLE* 2.1** Symbolize: Each one is older than his younger sisters.

Solution Where do we start? Identify the names, definite descriptions, and predicates. We do not have any names here. We cannot take 'his younger sister' as a definite description. (Why?) So we work with predicates only. Let us build our vocabulary for translation.

Oxy	:	x is older than y,
Sxy	:	y is a younger sister of x.

It looks that the vocabulary is enough. Now, what does the sentence say? It says that, "For every x, x is older than y if y is a younger sister of x."

But 'y is a younger sister of x' means what? The article 'a' refers to 'some' or 'all'? Is x older than all of his younger sisters or only older than some of them? It looks that the sentence asserts x to be older than all of his sisters. So, our semi-formal sentence looks like:

for every x, for every y, x is older than y if y is a younger sister of x.

We thus have the symbolization: $\forall x \forall y (Sxy \to Oxy)$.

***EXAMPLE* 2.2** Symbolize: If two persons fight over a third one's property, then the third one gains.

Solution The vocabulary would look like (You can use different symbols):

$Bxyz$	:	x and y fight over z
Gx	:	x gains
$f(x)$	:	property of x

The sentence may be symbolized as $\forall x \forall y \forall z (Bxyf(z) \to Gz)$.

***EXAMPLE* 2.3** Symbolize: For every pair of primes differing by 2, there is a pair of greater primes differing by 2.

Solution $\forall x \forall y(Px \wedge Py \wedge Dxy \to \exists z \exists w(Gzx \wedge Gwy \wedge Pz \wedge Pw \wedge Dzw))$

What do the predicates P, G, D stand for?

***Exercise* 2.2** Symbolize: There is a unique person who authored the book named *Logics for Computer Science.* [*Hint*: Use the predicate $\approx$.]

2.3 Preparing for Semantics

Look at the symbolizations of the three sentences in Examples 2.1−2.3. Can you find a variable there which is not quantified? Well, can you get a formula like $\forall xPx \to Qx$ while symbolizing an English sentence? Definitely not. It would tell:

If P holds for every one, then Q holds for x.

Now, if x had been a name, a constant, then it would refer to a person. But as a variable, it does not mean anything definite, it is only a named gap to be filled in with a name. We want, in fact, to see the difference between any arbitrary formula and a sentence. We require some terminology.

Definition 2.3 Let Y be a formula. A substring Z of Y is called a **subformula** of Y iff Z is itself a formula. The **scope** of an occurrence of a quantifier occurring in Y is the subformula of Y starting with that occurrence. An occurrence of a variable x in Y is **bound** iff this occurrence is within the scope of an occurrence of $\forall x$ or $\exists x$ (a quantifier that uses it). If an occurrence of a variable occurs within many occurrences of quantifiers using that variable, then it is said to be bound by the occurrence of the quantifier that occurs immediately to the left of it. An occurrence of a variable is called **free** iff it is not bound. A variable x is a **free variable** of Y iff there is at least one free occurrence of x in Y. A variable x is a **bound variable** of Y iff there is at least one bound occurrence of x in Y. A formula having no free variables is called a **closed formula** or a **sentence**.

If there is a free occurrence of a variable in a formula Y, we also say that x **occurs free in** Y. In the formula $\exists x(Pxy \wedge \forall yQy)$, the scope of (the only occurrence of) $\exists$ is the whole formula; scope of (again, the only occurrence of) $\forall$ is the formula $\forall yQy$. Here, all the occurrences of the variable x are bound; the first occurrence of y is free while the last two occurrences of y are bound occurrences. The variable x is a bound variable of the formula while y is both a free and a bound variable. The formula is not a closed formula (not a sentence) since it has a free variable.

In the formula $\forall x(\forall yPxyz \wedge \forall xQx \to \exists zPzzz)$, the first and second occurrences of x are bound and so are the third and fourth. The third and the fourth occurrences are within the scope of the first $\forall$; they are also within the scope of the third $\forall$. However, these occurrences of x are bound by the third occurrence of $\forall$, and not by the first $\forall$. The second occurrence of $\forall$ does not use the variable x. It is the scope of the innermost quantifier that binds a variable. All occurrences of variables x, y, z are bound occurrences. Hence this formula is a closed formula; it is a sentence. Is it now clear why we end up at closed formulas, or FL-sentences while translating English sentences?

***Exercise* 2.3** Find out the scopes of each occurrence of each quantifier, free and bound variables, and mark which quantifier binds which occurrences of variables in each of the following formulas:

(a) $\exists(Px \wedge Qx \to \neg Px \vee Qy)$

(b) $\forall x(Pyz \wedge \forall y(\neg Qxy \vee Pyz))$

(c) $\neg\exists x(\forall y Pxyz \wedge \forall z Qxyz) \to \top$

(d) $\forall x \exists y(Px \leftrightarrow Qy) \wedge \forall x \exists y(Px \to Pz \vee Qy)$

Determine which of these formulas are sentences.

Consider the sentence $\forall x(Nx \to Ox)$, which you have obtained by translating the English sentence "Every natural number is greater than or equal to 0". Now, how do you ascertain its truth? You take every possible x, say, $a, b, c, d, \ldots$. Then you are going to check whether

"If a is a natural number, then a is greater than or equal to 0."

"If b is a natural number, then b is greater than or equal to 0."

"If c is a natural number, then c is greater than or equal to 0." ...

That is, you will be interested in the truth of all the sentences

$$Na \to Oa, \quad Nb \to Ob, \quad Nc \to Oc, \ldots$$

Syntactically, you have substituted $a, b, c, \ldots$ in place of x in the formula $Nx \to Ox$, and then you want to check whether the formulas so obtained are true or not. In general, you may substitute terms in place of variables in any formula. But if you substitute x by a in $\forall x(Nx \to Ox)$, you are not getting anything; it only becomes $\forall a(Na \to Oa)$, and is not a formula as a is not a variable. That is, substitution of terms are only used meaningfully for free variables. We will give a notation.

We will write $[x/t]$ for the substitution of the variable x by the term t, and then for a formula X, we will write $X[x/t]$ for the formula obtained from X by effecting the substitution $[x/t]$ on it, i.e., for the formula obtained by substituting all the free occurrences of the variable x in the formula X by the term t. Sometimes, we will also write $(X)[x/t]$ for better readability. For example,

$$(\forall x \exists y(Px \wedge Qyx) \to Rzy)[y/t] = \forall x \exists y(Px \wedge Qyx) \to Rzt$$

$$(\forall x \exists y(Px \wedge Qyx) \to Rzy)[x/t] = \forall x \exists y(Px \wedge Qyx) \to Rzy$$

Note that the free occurrences are replaced, and not every occurrence. However, all the free occurrences must be replaced when a substitution is applied on the formula.

Consider the formula: $\forall x \exists y(Hx \to Fxy)$, which you might have got by symbolizing the English sentence, "Each human being has a father". Now to verify its truth, you are going to see whether for every object x, the formula $\exists y(Hx \to Fxy)$ is satisfied or not. Now this 'every object x' can also stand for terms (definite expressions, etc). For example, $f(a)$, the eldest brother of a, is an object anyway. After replacement, the formula

$$(\exists y(Hx \to Fxy))[x/f(a)] = \exists y(Hf(a) \to Ff(a)y)$$

says that " a's eldest brother, who is a human being, has a father". Instead of $f(a)$, suppose you substitute $f(y)$, then you would obtain

$$(\exists y(Hx \to Fxy))[x/f(y)] = \exists y(Hf(y) \to Ff(y)y)$$

What does this formula say? It says that "y's eldest brother, who is a human being, is his own father". This is absurd, whereas the original sentence with our common understanding sounds plausible. So, what is wrong if that object (for x) was y's eldest brother?

You can see what is happening after you substitute. The occurrences of x was free in $\exists y(Hx \to Fxy)$. By substituting x as $f(y)$, all these new occurrences of y become bound. This is not a faithful substitution. Syntactically, such a substitution is said to *capture* the variable. Our substitution should not capture a variable; otherwise, we will end up at a wrong and unintended situation. That is, the only admissible substitutions will be those which do not capture a variable, which do not make a free occurrence bound. The following definition formalizes this notion.

Definition 2.4 Let Z be a formula, x, y be variables and t be a term. The variable y is **free for** x in Z iff x does not occur free within the scope of any $\forall y$ or $\exists y$ in Z. The term t is free for x in Z iff each variable occurring in t is free for x in Z.

In the formula $\exists y(Hx \to Fxy)$, y is not free for x since x occurs free within the scope of $\exists y$. And this is the reason that after replacing this occurrence of x by $f(y)$, the new occurrence of y becomes bound, that y is getting captured by the substitution. The variable x is free for x in $\exists y(Hx \to Fxy)$; so is any variable z that does not occur in the formula. Similarly, z is free for x in $\exists y(Hx \to Fxy)$ as z does not at all occur in this formula. Analogously, if t is any term that does not have an occurrence of any variable, called a **closed term**, then t is free for x in any formula. If no variable of t is bound in a formula Y, then t is free for every variable occurring in Y. For example, the term $t = f(x, y, f(a, b, c))$ is free for y in the formula $\forall x \exists z(Pxy \leftrightarrow Qyz)$.

Once t is free for x in Y, all free occurrences of variables in Y remains free in $Y[x/t]$, i.e., no variable of t is captured by such a substitution. We will use substitutions for terms also. For example, $f(x)[x/t]$ will be taken equal to the term $f(t)$. We will write this equality using the same symbol '=' for terms as well as formulas. For example,

$$f(x, y, z, g(a))[x/t] = f(t, y, z, g(a))$$

$$\forall x(Pxy \leftrightarrow Qyz)[y/s] = \forall x(Pxs \leftrightarrow Qsz)$$

assuming, of course, that t is free for x in a formula where the term $f(x, y, z, g(a))$ has been used and, similarly, the term s must also be free for y in the formula $\forall x(Pxy \leftrightarrow Qyz)$.

Let us focus on how to interpret formulas. We go back to our old example of translating the sentence:

All natural numbers are greater than or equal to 0.

We had the translation: $\forall x(Nx \to Ox)$.

Now, as a formula (in FL), we do not, in fact, know what the predicates N and O stand for; they are simply some formal syntactic entities. All that the formula says is that N and O are unary predicates, and they can be combined in such a manner that we may regard the expression

$$Y = \forall x(Nx \to Ox)$$

as a formula. Now, by using a vocabulary for N, O, you may read the formula as different English sentences:

Every natural number is greater than 0.

(Nx: x is a natural number, Ox: x is greater than 0)

Authors of logic books are good teachers.

(Nx: x is an author of a logic book, Ox: x is a good teacher)

All men are females.

(Nx: x is a man, Ox: x is a female)

You can read it in many ways as you like, having the logical structure of the sentence as 'all so and so are such and such'. In the above readings, you have a model of the formula Y, say, of all numbers, where the first reading is probably meaningful. You also have models in the set of human beings, where the second and third reading of Y can probably be interpreted. In the first model of numbers, the predicate N stands for the unary relation of 'being a natural number'; in the second reading, N stands for the unary relation of 'being an author of a logic book', and so on. This is how you get infinite number of different interpretations of the same formula Y; once by changing the domain (a nonempty set) where you want to interpret and, next, by reading the predicates as different relations over the same domain.

For the formula $\forall x Pxf(x)$, you can have, similarly, an infinite number of models. For example, taking f as the function 'father of', and P as the relation 'younger than', you have a model where the closed formula represents the English sentence:

Everyone is younger than his father.

You can also read the sentence $\forall x Pxf(x)$ as

Every natural number is less than its square.

Or as

Every natural number is bigger than its cube.

All these English sentences are different readings or interpretations of the same formula $\forall x Pxf(x)$. What happens if we drop the quantifier? How to interpret the formula $Pxf(x)$? What does the variable x stand for? Of course, you can read it as

x is younger than his father ...

But then how do you determine the truth of such an unfinished sentence? This is unfinished because variables are just gaps, and only after substitution of a constant or a name there, you can get a sentence from it (else, use a quantifier). You cannot possibly take this sentence to be read as $\forall x Pxf(x)$ nor as $\exists x Pxf(x)$. Moreover, which are the ones to be taken and why? One way out is to think of a state where x is assigned to an object. Then the formula becomes a sentence, speaking about that object in that state. For example, in a program, you have a variable x. When this variable is assigned a value, say, 0, then you know the state of the program. The truth of a statement in that state now can be found out provided, of course, all the other variables have been either quantified or assigned some values. The truth of this kind of non-closed formulas depends upon a state.

Suppose that you admit of states for deciding the truth of such formulas as $Pxf(x)$ having a free variable. Now, can you connect the truth of such a formula in a state to the truth of a sentence (like $\forall x Pxf(x)$ or $\exists x Pxf(x)$)? Clearly, if $a, b, c, \ldots$ are all possible values for x, then we must have all of $Paf(a), Pbf(b), Pcf(c), \ldots$ to hold so that $\forall x Pxf(x)$ may be true. Similarly, if in some state $Pxf(x)$ is true, i.e., for one of these constants, say, a, if $Paf(a)$ holds, then $\exists x Pxf(x)$ also holds. Moreover, a state can be thought of as assigning values to the free variables of a formula to some objects in a model or interpretation. When we interpret $Pxf(x)$ in the set of human beings, with f as 'father of', P as 'younger than', and if x is assigned to Priyanka, then in this state, the formula $Pxf(x)$ would be interpreted as the sentence:

Priyanka is younger than her father.

Such an assignment function ℓ, which associates variables to elements of the universe or domain of an interpretation is called a **valuation** (or a **variable assignment function**). In the interpretation of x as 'Priyanka', you had taken $\ell(x) =$ Priyanka and $\ell(f(x)) =$ Priyanka's father. In so doing, you have taken ℓ as a mapping that associates not only variables but also terms to elements of the domain. If ℓ is extended to terms, we must see that it agrees with the mapping that associates predicates to relations and function symbols to actual functions over the domain. Writing as ϕ, the mapping that associates predicates and function symbols to relations and functions over the domain, we have here, $\phi(f) =$ 'father of'. But then 'father of Priyanka' is written in symbols, as $\phi(f)(\ell(x))$. With the valuation ℓ extended for terms, we must have then $\phi(f)(\ell(x)) = \ell(f(x))$. That is how a valuation must agree with the mapping ϕ of the interpretation.

Suppose we fix $\ell(x) = a$ and $\ell(y) = b$ while interpreting the formula $\forall x Pxy$. We must then consider all possible assignments to the variable x, even when y is fixed to b. This is so because the truth of $\forall x Pxy$ should not depend upon an assignment of x to some particular object, though it may depend upon the particular value y is assigned to. How to achieve this? We require some mechanism to consider all possible assignments which would differ from ℓ in the only variable x. That is, $\forall x Pxy$ will be true in each state fixed by ℓ if Pxy holds in every state fixed by ℓ', where ℓ' may differ from ℓ only in assigning x. One way is to use the notation $\ell[x \rightarrowtail a]$ denoting a valuation which equals the valuation ℓ except fixing x to a. Then we would require Pxy to hold in every state fixed by each valuation $\ell[x \rightarrowtail a]$ when a is allowed to be any arbitrary object. The other way is to define all **valuations ℓ' equivalent to ℓ along** x, meaning that $\ell(y) = \ell'(y)$ for every variable $y \neq x$. And then, we would require Pxy to hold in every state fixed by ℓ' so that $\forall x Pxy$ would hold. We will follow this latter approach as it simplifies notation.

One more point, when we interpret a binary predicate P as 'less than or equal to' in $\mathbb{N}$, the set of natural numbers, and a constant c as 0, then the formula $\forall x Pcx$ is interpreted as the sentence: "0 is less than or equal to every natural number". To know whether this formula is true under this interpretation, we must ascertain whether the interpreted sentence, "0 is less than or equal to every natural number" holds in $\mathbb{N}$ or not. So, we *assume* that there is an inner mechanism of 'truth' in-built in every interpretation for deciding whether a sentence in the domain holds or not.

2.4 Semantics of FL

You must remember that with all these formalisms, we are trying to capture a simple idea. If you encounter a formula $\exists x Px$, then you try to find an instance of x, a concrete object, where the property is satisfied. If you succeed in finding one such instance, then the formula is satisfied; otherwise not. Similarly, for the formula $\forall x Px$, if the property P holds for all possible instances in your domain, then, in that domain, the formula is satisfied.

Now that we have had an elaborate preparation for giving meanings to the building blocks of formulas, we may start with the formal definitions of the required notions.

An **interpretation** is a pair $I = (D, \phi)$, where D is a nonempty set, called the **domain** or **universe** of I, ϕ is a mapping associating function symbols with concrete functions on D and predicates with concrete relations on D, i.e., $\phi : \mathcal{P} \cup \mathcal{F} \to \mathcal{R}$, where $\mathcal{R}$ is the collection of all relations and functions on D. Further, the mapping ϕ preserves arity, i.e.,

(a) If P is any 0-ary predicate (a proposition), then $\phi(P)$ is a sentence about objects in D, which may either be true or false.

(b) If P is an n-ary predicate, $n \geq 1$, then $\phi(P) \subseteq D^n$, an n-ary relation on D.

(c) If f is a 0-ary function symbol (a constant, a name), then $\phi(f) \in D$, an object in D.

(d) If f is an n-ary function symbol, $n \geq 1$, then $\phi(f) : D^n \to D$, is a function of n arguments on D.

A **valuation** under the interpretation $I = (D, \phi)$ is a mapping ℓ that assigns each term to an element of D, which is first defined for variables and then extended to terms satisfying:

(i) If f is a constant, then $\ell(f) = \phi(f)$.

(ii) If f is an n-ary function symbol and $t_1, \ldots, t_n$ are terms, then $\ell(f(t_1, \ldots, t_n)) = \phi(f)(\ell(t_1), \ldots, \ell(t_n))$.

A valuation ℓ' is called **equivalent** to a valuation ℓ **along the variable** x iff $\ell(y) = l'(y)$ for all variables $y \neq x$. A **state** I_ℓ is a triple (D, ϕ, ℓ), where $I = (D, \phi)$ is an interpretation and ℓ is a valuation under I. Two states I_ℓ and $I_{\ell'}$ are said to be **equivalent along the variable** x iff the valuations ℓ and ℓ' are equivalent along the variable x.

Let A, B, C be formulas and $I_\ell = (D, \phi, \ell)$ be a state. Satisfaction of a formula in the state I_ℓ is defined recursively by the following:

1. $I_\ell \models \top$.
2. $I_\ell \not\models \bot$.
3. If A is a proposition (0-ary predicate), then $I_\ell \models A$ iff $\phi(A) = 1$ (i.e., iff $\phi(A)$ holds in D).
4. If $A = P(t_1, \ldots, t_n)$ for an n-ary predicate P other than $\approx$, and terms $t_1, \ldots, t_n$, then $I_\ell \models A$ iff $(\ell(t_1), \ldots, \ell(t_n)) \in \phi(P)$ (i.e., iff the objects $\ell(t_1), \ldots, \ell(t_n)$ in D are related by the relation $\phi(P)$ which is defined in D).
5. If $A = (s \approx t)$ for terms s, t, then $I_\ell \models A$ iff $\ell(s) = \ell(t)$ holds in D.
6. If $A = \neg B$, then $I_\ell \models A$ iff $I_\ell \not\models B$.
7. If $A = B \wedge C$, then $I_\ell \models A$ iff both $I_\ell \models B$ and $I_\ell \models C$.
8. If $A = B \vee C$, then $I_\ell \models A$ iff at least one of $I_\ell \models B$ or $I_\ell \models C$ holds.
9. If $A = B \to C$, then $I_\ell \models A$ iff $I_\ell \not\models B$ holds or $I_\ell \models C$ holds.

10. If $A = B \leftrightarrow C$, then $I_\ell \models A$ iff either both $I_\ell \models B$ and $I_\ell \models C$ hold, or both $I_\ell \not\models B$ and $I_\ell \not\models C$ hold.
11. If $A = \forall xB$, then $I_\ell \models A$ iff $I_{\ell'} \models B$ holds for every valuation ℓ' equivalent to ℓ along x.
12. If $A = \exists xB$, then $I_\ell \models A$ iff $I_{\ell'} \models B$ holds for some (at least one) valuation ℓ' equivalent to ℓ along x.

As in PL, there is also a notion of a first order language. In this context, the connectives, the punctuation marks, the quantifiers, the symbols $\top, \bot$, and the variables are taken as the *logical constants* and the predicates (including propositions) and function symbols (including names) are called the *nonlogical constants*. A **first order language** is then formed from an alphabet containing all the logical constants and some nonlogical constants following the same formation rules as FL. Then an interpretation is only defined for a first order language. In our terminology of an interpretation of a formula, the first order language contains all the nonlogical constants occurring in it, and possibly, some more. We do not use this terminology here, though you should be able to follow other texts which use this.

The satisfaction relation between a state I_ℓ and a formula A, i.e., $I_\ell \models A$, is also read as " the interpretation I is a **model** of the formula A in the state ℓ " or as " the state I_ℓ is a **state-model** of A ", or as " I_ℓ *satisfies* A ", or as " I_ℓ *verifies* A ". A formula A is said to be *satisfiable* iff for some interpretation I and some valuation ℓ under I, the state $I_\ell \models A$.

An interpretation I is said to **satisfy** a formula A, written as $I \models A$ iff $I_\ell \models A$, for every valuation ℓ (under I). $I \models A$ is read as I is a *model* of A. To distinguish between the metaphrases: 'I is a model of', and 'I_ℓ is a model of', we will refer to I_ℓ as a state-model. To say that the state I_ℓ is a model of a formula A, where ℓ is a valuation under the interpretation I, we would rather stick to saying that I_ℓ is a state-model of A. A formula A is called **valid**, written as $\models A$ iff for every interpretation I, $I \models A$, i.e., for every interpretation I, and for every valuation ℓ under I, the state $I_\ell \models A$.

A set Σ of formulas is called *satisfiable* iff there is an interpretation I, a valuation ℓ under I such that $I_\ell \models A$ for every $A \in \Sigma$. The interpretation I is a *model* of Σ, written as $I \models \Sigma$ iff $I \models A$ for every $A \in \Sigma$. For a set Σ of formulas and a formula A, Σ **semantically entails** A (or A is a semantic consequence of Σ), written as $\Sigma \models A$, iff every state-model of Σ is a state-model of A (iff for every interpretation I and for every valuation ℓ under I, if $I_\ell \models B$ for every $B \in \Sigma$, then $I_\ell \models A$). If $\Sigma = \{X_1, \ldots, X_n\}$, the consequence $\Sigma \models X$ is also written as $X_1, \ldots, X_n \models X$. If the domain of a model has m elements, we will say that the model has m elements. Thus **finite** or *infinite models* are the models whose domain is finite or infinite, respectively.

Two formulas A, B are **equivalent**, written as $A \equiv B$ iff every state-model of A is also a state-model of B and vice versa. That is, $A \equiv B$ iff for every interpretation I and every valuation ℓ under I, we have either $I_\ell \models A$ and $I_\ell \models B$, or $I_\ell \not\models A$ and $I_\ell \not\models B$.

Compare the complicated semantics of FL with that of PL. Here, satisfaction is defined in a state which has an interpretation and a valuation. So, we have the concepts of a model and of a state-model. If I is an interpretation and ℓ is a valuation under I, we have the state I_ℓ. Now, a formula A is satisfiable if it has a state-model. In contrast, 'A has a model' says something stronger. It says that for every valuation ℓ under I, the state I_ℓ is a state-model of A. Thus there can be satisfiable formulas having no models; since there may be a state-model of it. Similarly, for validity, a formula A is valid if, for every interpretation and for every valuation under that interpretation, the state-model $I_\ell \models A$, i.e., the corresponding sentence of A holds in the domain D.

***EXAMPLE* 2.4** Consider the formulas Pxy, $Pcf(c)$ and interpretations $I = (\mathbb{N}, \phi)$, $J = (H, \phi, \psi)$, where $\mathbb{N}$ is the set of all natural numbers, H is the set of all human beings, $\phi(P) =$'less than', $\phi(c) = 0$, $\phi(f) =$ 'successor function', $\psi(P) =$ 'is a brother of', $\psi(c) =$ Rajiv, and $\psi(f) =$ 'mother of'. Determine whether $I \models Pxy, I \models Pcf(c), J \models Pxy, J \models Pcf(c)$.

Solution Note that f is a unary function symbol; thus it can be associated to the functions 'successor of' and 'mother of', which take only one argument. Similarly, P is a binary predicate and is associated with binary relations 'less than' and 'is a brother of'. It is immaterial what ϕ, ψ associate with to the other predicates and functions. Let ℓ be a valuation under I, where $\ell(x) = 0, \ell(y) = 1$ and m be a valuation under J with $m(x) =$ Rajiv and $m(y) =$ Sanjay.

The state $I_\ell = (\mathbb{N}, \phi, \ell)$ is a state-model of Pxy. That is, $I_\ell \models Pxy$ iff $(\ell(x), \ell(y)) \in \phi(P)$ iff $(0, 1)$ is a member of the relation, 'less than' iff $0 < 1$, which holds in $\mathbb{N}$. Therefore, $I_\ell \models Pxy$.

Now, $I_\ell \models Pcf(c)$ iff $(\ell(c), \ell(f(c))) \in \phi(P)$ iff $(\phi(c), \phi(f)(\phi(c))) \in \phi(P)$ iff (0, successor of 0) $\in \phi(P)$ iff $0 < 1$. Since this is true in $\mathbb{N}$, $I_\ell \models Pcf(c)$.

For the state J_m, we have $J_m \models Pxy$ iff $(m(x), m(y)) \in \psi(P)$ iff (Rajiv, Sanjay) $\in$ 'is a brother of' iff Rajiv is a brother of Sanjay. This latter sentence holds (as both are sons of Mrs. Indira Gandhi). Hence $J_m \models Pxy$. Note that, if you have some other Rajiv and Sanjay in mind, then in that world, you may see that Rajiv is not a brother of Sanjay. Then $J_m \not\models Pxy$. However, it will be unequivocal due to our assumption that in every domain, we would like to consider, there must already exist a mechanism to ascertain the truth of sentences (about objects in the domain).

Now, $J_m \models Pcf(c)$ iff $(m(c), m(f(c))) \in \psi(P)$ iff $(\psi(c), \psi(f)(\psi(c))) \in \psi(P)$ iff (Rajiv, mother of Rajiv) $\in$ 'is a brother of' iff Rajiv is a brother of his own mother, which as we know, does not hold. Thus, $J_m \not\models Pcf(c)$.

What about I, J? $I \models Pxy$ iff $I_\ell \models Pxy$ for every valuation ℓ. We have already seen this for a particular valuation ℓ above which takes x to 0 and

y to 1. Consider another valuation j with $j(x) = 1, j(y) = 0$. Is it that $I_j \models Pxy$? Well, by definition, $I_j \models Pxy$ iff $(j(x), j(y)) \in \phi(P)$ iff $1 < 0$, which does not hold. Hence $I_j \not\models Pxy$. Consequently, $I \not\models Pxy$.

For $I \models Pcf(c)$, we may take another valuation j and try checking whether $I_j \models Pcf(c)$. But, $Pcf(c)$ has no variables, thus we will essentially repeat the earlier argument:

$$I_j \models Pcf(c) \text{ iff } (\phi(c), \phi(f)(\phi(c))) \in \phi(P) \text{ iff } n_1 < n_1 + 1$$

when $\phi(c) = n_1$ and successor of n_1 is $n_1 + 1$. This holds, whatever $n_1 \in \mathbb{N}$ we choose. Hence $I_j \models Pcf(c)$. Consequently, $I \models Pcf(c)$.

It is instructive to see that we cannot try all natural numbers one by one, which is asked when we change valuations. It is then easier to check for **unsatisfiability** (A is unsatisfiable if A is not satisfiable.) by a suitable choice rather.

Now, for $J \models Pxy$, we have seen that $J_m \models Pxy$. We take one more valuation, say, i under J with $i(x) =$ Sonia , $i(y) =$ Maneka. $J_m \models Pxy$ iff Sonia is a brother of Maneka, which does not hold. (Both Sonia and Maneka are ladies.) Consequently, $J_m \not\models Pxy$. Similarly, since $J_m \not\models Pcf(c)$, we have $J \not\models Pcf(c)$.

The formula Pxy is satisfiable since it has a state-model I_ℓ. The formula $Pcf(c)$ is also satisfiable since it too has a state-model I_ℓ. The formula Pxy is **invalid** (not valid) as there is a state, say, I_j which does not satisfy (which **falsifies**) Pxy. Similarly, since $J_m \not\models Pcf(c)$, the formula $Pcf(c)$ is also invalid.

In the above example, you had seen that it does not matter what an interpretation associates with to predicates or function symbols that do not occur in a formula. Using this intuition, we may sometimes write the corresponding relations and functions explicitly in an interpretation instead of defining formally what ϕ is.

For example, if we are to interpret the formula $Pxf(yz)$, it is enough to write an interpretation $I = (\mathbb{N}, \phi)$, with $\phi(P) =$ ' $<$', $\phi(f) =$ 'sum of so and so', as $I = (\mathbb{N}, P', f')$, where P' is '$<$' and f' is 'sum of so and so'. We follow this convention of writing interpretations in the next example.

***EXAMPLE* 2.5** Is the formula $A = \forall x Pxf(x)$ satisfiable? Is it valid?

Solution Take an interpretation $I = (\mathbb{N}, P', f')$, where P' is the 'less than' relation and f' is the function 'plus 5'. Since a variable x is occurring in A, we must also start with a valuation, say, ℓ with $\ell(x) = 2$.

The state $I_\ell \models \forall x Pxf(x)$ iff $I_{\ell'} \models Pxf(x)$ for every valuation ℓ' equivalent to ℓ along x. One such ℓ', for example, is given by $\ell'(x) = 3$ and $\ell'(y) = \ell(y)$ for $y \neq x$. Here, however, $\ell'(y)$ or $\ell(y)$ need not be defined as the variable y does not appear in the formula. Now, $I_{\ell'} \models Pxf(x)$ iff $\ell'(x) < \ell'(f(x))$ iff $\ell'(x) < f'(\ell'(x))$ iff $2 < 2 + 5$ which holds in $\mathbb{N}$. Hence, $I_{\ell'} \models Pxf(x)$.

The same argument applies even when you take $\ell'(x)$ as 0 or 1 or 3 or 4, etc. Hence, for each valuation ℓ' equivalent to ℓ along x, we see that $I_{\ell'} \models Pxf(x)$. This shows that $I \models \forall x Pxf(x)$.

Thus the formula $\forall x Pxf(x)$ is satisfiable as it has a state-model such as $I_{\ell'}$; it also has a model, such as I. Is it also valid? It does not seem likely since there is too much of arbitrariness in this formula. To see that it is invalid, you must find a state that may falsify it. For example, consider the interpretation $J = (\mathbb{N}, \bar{P}, f')$, where $\bar{P}$ is the 'greater than' relation. Take ℓ' and f' as above. We then have $J_{\ell'} \models Pxf(x)$ iff $(\ell'(x), \ell'(f(x))) \in \bar{P}$ iff $2 > 2 + 5$, which does not hold. Hence $J_{\ell'} \not\models Pxf(x)$. Consequently, $J_{\ell'} \not\models \forall x Pxf(x)$, since ℓ' is also equivalent to itself along x. That is, the formula $\forall x Pxf(x)$ is invalid.

***EXAMPLE* 2.6** Consider the formula $A = \forall x \forall y (Pxa \land Pyx \to \neg Pya)$. Let $D = \{a', b', c'\}$ and $P' = \{(a', a'), (b', a'), (c', a')\}$. Define the interpretation $I = (D, P', a')$. Here we are not using ϕ; instead, we implicitly write $\phi(P) = P'$ and $\phi(a) = a'$. Now, is it true that $I \models A$? (Does $I \models A$ hold?)

Solution We start with a valuation ℓ mapping the relevant variables to D, say, $\ell(x) = a', \ell(y) = b'$. We must first decide whether the state I_ℓ satisfies A or not. By definition, $I_\ell \models A$ iff for every valuation ℓ' equivalent to ℓ along x, $I_{\ell'} \models \forall y (Pxa \land Pyx \to \neg Pya)$. How do the ℓ' s look like? $\ell'(y) = \ell(y) = b'$ but $\ell'(x)$ is now free to have any of the values a', b', c'. Thus, we have three choices for ℓ':

$$\ell_1'(x) = a', \ell_1'(y) = b';\ \ \ell_2'(x) = b', \ell_2'(y) = b';\ \ \ell_3'(x) = c', \ell_3'(y) = b'.$$

We must check whether each of these satisfies the formula, i.e., whether

$$I_{\ell_i'} \models \forall y (Pxa \land Pyx \to \neg Pya) \text{ for } i = 1, 2, 3$$

Let us check for ℓ_1'. Now, $I_{\ell_1'} \models \forall y (Pxa \land Pyx \to \neg Pya)$ iff for every valuation $j1$ equivalent to ℓ_1' along y, $I_{j1} \models Pxa \land Pyx \to \neg Pya$. There will again be three cases for $j1$. This time, y varies. We have the valuations (equivalent to $j1$ along y):

$$j11(x) = j12(x) = j13(x) = a', j11(y) = a', j12(y) = b', j13(y) = c'.$$

We must check that $I_{j1k} \models Pxa \land Pyx \to \neg Pya$ for $k = 1, 2, 3$. Now,

$$\begin{aligned}
&I_{j11} \models Pxa \land Pyx \to \neg Pya\\
&\quad \text{iff} \ \ I_{j11} \not\models Pxa \land Pyx \text{ or } I_{j11} \models \neg Pya\\
&\quad \text{iff} \ \ I_{j11} \not\models Pxa \text{ or } I_{j11} \not\models Pyx \text{ or } I_{j11} \not\models Pya\\
&\quad \text{iff} \ \ (j11(x), a') \notin P' \text{ or } (j11(y), j11(x)) \notin P' \text{ or } (j11(y), a') \notin P'\\
&\quad \text{iff} \ \ (a', a') \notin P' \text{ or } (a', a') \notin P' \text{ or } (a', a') \notin P'\\
&\quad \text{iff} \ \ (a', a') \notin P', \text{ which is not the case as } (a', a') \in P'.
\end{aligned}$$

Then it follows, in succession, that

$$I_{j11} \not\models Pxa \wedge Pyx \to \neg Pya$$

$$I_{\ell_1'} \not\models \forall y(Pxa \wedge Pyx \to \neg Pya)$$

$$I_\ell \not\models \forall y(Pxa \wedge Pyx \to \neg Pya)$$

$$I \not\models \forall x \forall y(Pxa \wedge Pyx \to \neg Pya)$$

However, it would be easier to see how the interpretation works, informally. The sentence $\forall x \forall y(Pxa \wedge Pyx \to \neg Pya)$ would be satisfied under the interpretation I provided we see that for each possibility for x and y as elements of the domain D, the sentence holds. Since $D = \{a', b', c'\}$ has three elements, and there are two variables x, y to be instantiated to these elements, we would have to consider the following nine sentences:

1. If $(a', a') \in P'$ and $(a', a') \in P'$, then $(a', a') \notin P'$.
2. If $(a', a'), \in P'$ and $(b', a') \in P'$, then $(b', a') \notin P'$.

$$\vdots$$

9. If $(c', a') \in P'$ and $(c', c') \in P'$, then $(c', a') \notin P'$.

Moreover, all the sentences must hold on D. Now, we see that the first sentence itself does not hold. Therefore, $I \not\models A$. This is essentially the semantics which we have defined recursively.

EXAMPLE 2.7 Is $\forall x \exists y Pxy \models \exists y \forall x Pxy$?

Solution Let us try to see whether there is a state I_ℓ which is a state-model of $\forall x \exists y Pxy$ but not of $\exists y \forall x Pxy$.

Let $I = (D, P')$, where $D = \{2, 3\}$, $P' = \{(2, 3), (3, 2)\}$. Since there are only two variables x and y and two objects in the domain D of I, the relevant valuations are:

$$\ell 1(x) = 2, \ell 1(y) = 2;\ \ \ell 2(x) = 2, \ell 2(y) = 3$$

$$\ell 3(x) = 3, \ell 3(y) = 2;\ \ \ell 4(x) = 3, \ell 4(y) = 3$$

Now, $I_{\ell 1} \models \forall x \exists y Pxy$ iff for every valuation $\ell 1'$ equivalent to $\ell 1$ along x, we have $\ell 1' \models \exists y Pxy$. what are the valuations equivalent to $\ell 1$ along x? These valuations assign the same value to y but may assign different values to x. As $\ell 1(y) = 2$, we have only two such valuations. They are $\ell 1$ and $\ell 3$. Hence,

$$I_{\ell 1} \models \forall x \exists y Pxy \text{ iff } I_{\ell 1} \models \exists y Pxy \text{ and } I_{\ell 3} \models \exists y Pxy.$$

Again, $I_{\ell 1} \models \exists y Pxy$ iff for every valuation $m1$ equivalent to $\ell 1$ along y, we have $I_{m1} \models Pxy$. What are such valuations? They would possibly differ in assigning values to y, but x must be fixed as in $\ell 1$, to 2 as $\ell 1(x) = 2$. These are the valuations $\ell 1$ and $\ell 2$ above. Thus,

$$I_{\ell 1} \models \exists y Pxy \text{ iff at least one of } I_{\ell 1} \models Pxy \text{ or } I_{\ell 2} \models Pxy \text{ holds.}$$

Similarly, when can we assert that $I_{\ell 3} \models \exists y Pxy$? First, we see that the valuations equivalent to $\ell 3$ along y (keeping x fixed to $\ell 3(x) = 3$) are $\ell 3$ and $\ell 4$. Thus,

$$I_{\ell 3} \models \exists y Pxy \text{ iff at least one of } I_{\ell 3} \models Pxy \text{ or } I_{\ell 4} \models Pxy \text{ holds.}$$

Continuing, we find that

$$\begin{aligned}
& I_{\ell 1} \models \forall x \exists y Pxy \\
& \quad \text{iff } [I_{\ell 1} \models Pxy \text{ or } I_{\ell 2} \models Pxy] \text{ and } [I_{\ell 3} \models Pxy \text{ or } I_{\ell 4} \models Pxy] \\
& \quad \text{iff } [(\ell 1(x), \ell 1(y)) \in P' \text{ or } (\ell 2(x), \ell 2(y)) \in P'] \text{ and} \\
& \qquad [(\ell 3(x), \ell 3(y)) \in P' \text{ or } (\ell 4(x), \ell 4(y)) \in P'] \\
& \quad \text{iff } [(2,3) \in P' \text{ or } (2,3) \in P'] \text{ and } [(3,2) \in P' \text{ or } (3,3) \in P'].
\end{aligned}$$

This is so, since both $(2,3), (3,2) \in P'$. Therefore, $I_{\ell 1} \models \forall x \exists y Pxy$. What about the satisfaction $I_{\ell 1} \models \exists y \forall x Pxy$? Now,

$$\begin{aligned}
& I_{\ell 1} \models \exists y \forall x Pxy \\
& \quad \text{iff for some valuation } m1 \text{ equivalent to } \ell 1 \text{ along } y,\ I_{m1} \models \forall x Pxy \\
& \quad \text{iff } I_{\ell 1} \models \forall x Pxy \text{ or } I_{\ell 2} \models \forall x Pxy \qquad (\text{Since } m1 \text{ is either } \ell 1 \text{ or } \ell 2.) \\
& \quad \text{iff } (I_{\ell 1} \models Pxy \text{ and } I_{\ell 3} \models Pxy) \text{ or } (I_{\ell 2} \models Pxy \text{ and } I_{\ell 4} \models Pxy) \\
& \qquad (\text{Since } \ell 1 \text{ and } \ell 3 \text{ are the only valuations equivalent to } \ell 1 \text{ along } x, \\
& \qquad \text{and } \ell 2 \text{ and } \ell 4 \text{ are the only valuations equivalent to } \ell 2 \text{ along } x.) \\
& \quad \text{iff } [(\ell 1(x), \ell 1(y)) \in P' \text{ and } (\ell 3(x), \ell 3(y)) \in P'] \text{ or} \\
& \qquad [(\ell 2(x), \ell 2(y)) \in P' \text{ and } (\ell 4(x), \ell 4(y)) \in P'] \\
& \quad \text{iff } [(2,2) \in P' \text{ and } (3,2) \in P'] \text{ or } [(2,3) \in P' \text{ and } (3,3) \in P']
\end{aligned}$$

Since this does not hold, $I_{\ell 1} \not\models \exists y Pxy$, i.e., $\forall x \exists y Pxy \not\models \exists y \forall x Pxy$.

***Exercise* 2.4** In the solution to Example 2.7, see that $I_{\ell k} \models \forall x \exists y Pxy$, for $k = 2, 3, 4$. Further, show that $\exists y \forall x Pxy \models \forall x \exists y Pxy$.

We will once again see Example 2.7 informally. Interpret P as the 'less than' relation over the set $\mathbb{N}$ of natural numbers. Then, the formula $\forall x \exists y Pxy$ simply means that "for each natural number, there is a bigger natural number". Whereas the formula $\exists y \forall x Pxy$ means that "there is a biggest natural number". Clearly, the first one holds, while the second does not. Therefore, the consequence $\forall x \exists y Pxy \models \exists y \forall x Pxy$ is not valid.

This argument is simple. But there is a minor technical problem. Here, we have simply stated that there is a model of $\forall x \exists y Pxy$ which is not a model of $\exists y \forall x Pxy$. However, for showing the invalidity of the consequence, by definition, we must have a state-model of $\forall x \exists y Pxy$ which must falsify $\exists y \forall x Pxy$. In general, the two concepts of having a model and having a state-model are not equivalent. Consider the consequence $A \models B$. One critical case is when a formula A has no models but has only a state-model. In such a case, each model of A is a model of B, but the state-model of A

need not satisfy B. Thus, the above simpler argument does not show that $\forall x \exists y Pxy \not\models \exists y \forall x Pxy$, is it? However, for a sentence, a closed formula, we will see that the concepts of a model and a state-model coincide, and then this argument will indeed be correct.

In fact, for satisfaction of sentences, the interpretation I or the state I_ℓ should not matter since our definition, in a way, gives more importance to quantifiers than to the particular valuations; quantifiers override the valuations. To this end, we start with the following: (Why is it so named?)

Theorem 2.1 (Relevance Lemma) *Let A be a formula and V_A be the set of all free variables of A. Let $I = (D, \phi)$ be an interpretation and ℓ, ℓ' be valuations under I such that $\ell(x) = \ell'(x)$ for every $x \in V_A$. Then $I_\ell \models A$ iff $I'_\ell \models A$.*

Proof We prove it by induction on the number of occurrences of connectives and quantifiers in A. As the basis of induction, A may be in one of the following forms:

(i) $\top$, (ii) $\bot$, (iii) a proposition B, (iv) $(s \approx t)$, (v) $P(t_1, \ldots, t_n)$,

for an n-ary predicate $P \neq \approx$ and terms $s, t, t_1, \ldots, t_n$.

In case (i), both $I_\ell \models \top$ and $I_{\ell'} \models \top$. In case (ii), both $I_\ell \not\models \bot$ and $I_{\ell'} \not\models \bot$. In case (iii), $I_\ell \models B$ iff $\phi(B) = 1$ iff $I_{\ell'} \models B$.

In case (iv), since all variables appearing in $(s \approx t)$ are free variables, we have $\ell(x) = \ell'(x)$ for all variables occurring in $(s \approx t)$. Moreover, $\ell(c) = \phi(c) = \ell'(c)$ for every constant c occurring in $(s \approx t)$. It is thus straightforward (by induction on the number of variables occurring in any term) that $\ell(s) = \ell'(s)$. Similarly, $\ell(t) = \ell'(t)$. Hence, $I_\ell \models (s \approx t)$ iff $\ell(s) = \ell(t)$ iff $\ell'(s) = \ell'(t)$ iff $I_{\ell'} \models (s \approx t)$.

In case (v), we use the result that $\ell(s) = \ell'(s)$ for every term s, where $\ell(x) = \ell'(x)$ for every variable and constant x occurring in s. (Exercise 2.5.) Now,

$$\begin{aligned}
&I_\ell \models P(t_1, \ldots, t_n) \\
&\quad \text{iff} \;\; (\ell(t_1), \ldots, \ell(t_n)) \in \phi(P) \\
&\quad \text{iff} \;\; (\ell'(t_1), \ldots, \ell'(t_n)) \in \phi(P) \text{ as } \ell(t_i) = \ell'(t_i) \\
&\quad \text{iff} \;\; I_{\ell'} \models P(t_1, \ldots, t_n).
\end{aligned}$$

For the induction step, lay out the induction hypothesis that for any formula having number of occurrences of connectives and quantifiers less than k, the statement holds. Let A be a formula having this number as k. Then we have the following cases:

(a) $A = \neg B$ for some formula B.

(b) $A = (B * C)$ for some formulas B, C and $* \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$.

(c) $A = \exists x B$ for some formula B and some variable x.

(d) $A = \forall x B$ for some formula B and some variable x.

In case (a), $I_\ell \models \neg B$ iff $I_\ell \not\models B$ iff $I_{\ell'} \not\models B$ (by induction hypothesis) iff $I_{\ell'} \models \neg B$. Case (b) is similar.

For cases (c) and (d), suppose that m is a valuation equivalent to ℓ along x, i.e., $m(y) = \ell(y)$ for all variables $y \neq x$. Let $m'(y) = m(y)$ for all free variables of B. With the fact that $\ell'(y) = \ell(y)$ for all free variables of $\exists x B$, what can be said about ℓ' and m'? Well, these conditions imply that for all free variables z of A with $z \neq x, m'(z) = m(z) = \ell(z) = \ell'(z)$ (as $V_A \subseteq V_B$). That is, m' is a valuation equivalent to ℓ' along x.

For case (c), suppose that $I_\ell \models \exists x B$. Then there is at least one valuation m equivalent to ℓ along x such that $I_m \models B$. If m' is a valuation that agrees with m on V_B, then we have $I_{m'} \models B$ as well. By induction hypothesis, there exists such a valuation m'. Using the result that m' is equivalent to ℓ' along x, we conclude that there exists a valuation m' equivalent to ℓ' along x such that $I_{m'} \models B$. This gives $I_{\ell'} \models \exists x B$. Thus, we have proved that if $I_\ell \models \exists x B$ then $I_{\ell'} \models \exists x B$. Converse implication is obtained by reversing the roles of ℓ and ℓ'. Case (d) is proved similarly. ∎

***Exercise* 2.5** Show that if ℓ and ℓ' are two valuations that agree on all variables occurring in a term t, then $\ell(t) = \ell'(t)$ by induction on the number of occurrences of variables in t or on the number of occurrences of function symbols in t. [*Hint*: Both ℓ and ℓ' are valuations under the same interpretation I.]

***Exercise* 2.6** Complete (b) and (d) in the proof of Theorem 2.1.

As a consequence of Relevance Lemma, we see that for a sentence, a valuation does not contribute anything towards its satisfaction. For, when A is a sentence (a closed formula), $V_A = \emptyset$ and, vacuously, ℓ and ℓ' agree on all the free variables of A. By definition then, $I_\ell \models A$ iff $I_{\ell'} \models A$. That is, for a sentence, a state model and a model have the same meaning. Thus we have the following theorem:

Theorem 2.2 (Relevance Lemma for Sentences) *If A is a sentence, I is an interpretation, and ℓ is a valuation under I, then $I_\ell \models A$ iff $I \models A$.*

As a corollary, we obtain the following theorem (prove it):

Theorem 2.3 *Let A be a sentence and I be an interpretation. Then either $I \models A$ or $I \models \neg A$.*

***Exercise* 2.7** Construct formulas X and Y which are not closed, and interpretations I and J such that

(a) either $I \models X$ or $I \models \neg X$ (b) $J \not\models Y$ and $J \not\models \neg Y$.

What about constructing a formula Z and an interpretation K such that $K \models Z$ and $K \models \neg Z$?

Theorems 2.1–2.3 confirm our intuition that a sentence is satisfiable iff it has a model. In other words, any interpretation I can either be a model of a sentence or is a model of its negation. However, for an arbitrary formula, this may not hold. Such a dichotomy is possible only for states. To be explicit, for an arbitrary formula, a state is either a state-model of it or is a state-model of its negation. Sentences are special in this sense; and thus allow translation into a 'fact statement' in any interpretation. For example, the sentence $\forall x \exists y Pxy$ can be translated into the fact statement: "every object in D is related to some object by the relation $\phi(P)$ " in the interpretation (D, ϕ). Similarly, $\exists x \forall x Pxy$ can be translated as "every object in D is related to the same object in D by the relation $\phi(P)$ ". In so doing, one may just regard each variable occurring in a sentence as a variable in the domain D and every predicate as a relation on D, etc.

These considerations quite naturally lead to different presentations of semantics, while tackling the quantifiers. For example, when we say that $\forall x Px$ is true in natural numbers, we would first prescribe how the unary predicate is interpreted in $\mathbb{N}$. Once we fix such a relation $P' \subseteq \mathbb{N}$, which is associated with P, the sentence $\forall x Px$ is translated (to within $\mathbb{N}$) as "for every natural number x, P' holds for x ", that is, "all of $P'(0), P'(1), P'(2), \ldots$ hold".

Here we are considering valuations that fix x to 0, to 1, to 2, ... one after another, and then the relation P' must hold correspondingly for each of them. We cannot, of course, use substitutions $[x/0], [x/1], \ldots$ and then consider the formulas $P(0), P(1), \ldots$ since $[x/0], [x/1], \ldots$ are not substitutions at all. Why? Because, $0, 1, 2, \ldots$ are not terms; they are the objects from the domain of the interpretation. The same effect is produced by using valuations. If ℓ is a valuation, we write $\ell[x \rightarrowtail a]$ as a new valuation obtained from ℓ which assigns every variable other than x to what ℓ assigns, but fixing x to a. That is,

$$\ell[x \rightarrowtail a](y) = a \text{ if } y = x, \text{ else it equals } \ell(y).$$

Then $\forall x Px$ would be true under the state, now called an **environment**, I_ℓ, provided Px holds in all the environments $I_{\ell[x \rightarrowtail a]}$ where a varies over the set of natural numbers, $\mathbb{N}$.

In general, the difference between the valuations equivalent to ℓ along x and $\ell[x \rightarrowtail a]$ is that $\ell[x \rightarrowtail a]$ fixes x to a. Formally then, we would like to define satisfaction of a formula in an environment I_ℓ by modifying only the quantifier cases (9) and (10) in the definition of satisfaction. These two cases of satisfaction are replaced by

(9') If $A = \forall x B$, then $I_\ell \models A$ iff $I_{\ell[x \rightarrowtail d]} \models B$ for every $d \in D$.

(10') If $A = \exists x B$, then $I_\ell \models A$ iff $I_{\ell[x \rightarrowtail d]} \models B$ for some $d \in D$.

Now, the assignment ℓ is simply a look-up table giving values of the variables (and also to terms, being faithful to ϕ). The fact that this change in semantics is no change is expressed by the following statement:

Theorem 2.4 *Let A be any formula and $I = (D, \phi)$ be an interpretation. Let ℓ be a valuation. Then the state $I_\ell \models A$ iff the environment $I_\ell \models A$.*

Proof A formal proof is obtained by induction on the number of occurrences of quantifiers in A. Since satisfaction in an environment differs from that in a state only for the quantifiers, the crucial step is to prove the statement when A is of the form $\forall xB$ or $\exists xB$ assuming that the theorem holds for the formula B. That is, one shows:

(a) For every valuation m equivalent to ℓ along x,
$I_m \models B$ iff for every $d \in D$, $I_{\ell[x \mapsto d]} \models B$.

(b) There is at least one valuation m equivalent to ℓ along x such that
$I_m \models B$ iff for some $d \in D$, $I_{\ell[x \mapsto d]} \models B$.

Assume that for every valuation m equivalent to ℓ along x, $I_m \models B$. Now, $\ell[x \mapsto d]$ is one such valuation equivalent to ℓ along x. Hence, $I_{\ell[x \mapsto d]} \models B$. This holds irrespective of whatever $d \in D$ we choose. Therefore, for every $d \in D$, $I_{\ell[x \mapsto d]} \models B$.

Conversely, assume that for every $d \in D$, $I_{\ell[x \mapsto d]} \models B$. Let m be a valuation equivalent to ℓ along x. Then what is m? It may only differ from ℓ at x. But $m(x) \in D$. Hence $m = \ell[x \mapsto m(x)]$. Then from our assumption it follows that $I_{\ell[x \mapsto m(x)]} \models B$, i.e., $I_m \models B$. This is the case for every valuation m equivalent to ℓ along x. This proves (a). Similar argument proves (b). ∎

***Exercise* 2.8** Show the following:

(a) $\ell[x \mapsto d][x \mapsto e] = \ell[x \mapsto e]$.

(b) If $x \neq y$, then $\ell[x \mapsto d][y \mapsto e] = \ell[y \mapsto e][x \mapsto d]$.

(c) If $x \neq y$, then $\forall y(B[x/\bar{d}]) = (\forall yB)[x/\bar{d}]$.

Henceforward, we will not distinguish between states, environments, and structures; we will use whichever we feel is convenient. Before going further, we will have an alternative semantics.

Note that we have introduced environments to circumvent the difficulty in substituting objects from a domain for variables. We could not possibly substitute a variable by an element from the domain since in a substitution of the form $[x/t]$, t must be a term. If we represent each element from the domain, by a new constant, then such a replacement would be syntactically correct. In this alternate view of semantics, we do just this. Corresponding to each element in the domain, we introduce a new constant extending the syntax of our language. Then as entities in our extended language, these new constants can be used in the substitutions. For example, consider interpreting the formula $\forall xPx$ in $\mathbb{N}$. First, we adjoin to our alphabet (of FL) new constants, say, n' corresponding to each $n \in \mathbb{N}$. Then, $I = (\mathbb{N}, P')$ will be a model of $\forall xPx$ iff I is a model of $P_{n'}$ for each n'. This is more direct than earlier semantics.

However, in our language, we will have at least as many more constants as the elements in the domain D. That is, if we interpret a formula in a finite set having m elements, we will have m constants in our language. If we interpret a formula in $\mathbb{R}$, of real numbers, we will have an uncountable number of symbols in our alphabet. Of course, this is no problem since in any formula, we would have only a finite number of symbols used; proof by induction will still work. Since new symbols are imported to the language, we will refer to this semantics by the name, **import semantics**. A formal description of import semantics follows.

Given a formula and an interpretation $I = (D, \phi)$, the alphabet is extended by adding new (individual) constants, the symbols in the set $\mathrm{NC} = \{\bar{d} : d \in D\}$. The function ϕ is extended to include in its domain the new constants by $\phi(\bar{d}) = d$. Any valuation ℓ under I is similarly extended by being faithful to ϕ, i.e., by defining $\ell(\bar{d}) = \phi(\bar{d}) = d$ for each $\bar{d} \in \mathrm{NC}$. Definitions of states and environments are kept as they were. We will refer to the states or environments as **structures** in import semantics. For satisfaction, i.e., $I_\ell \models A$, of a formula A in a structure I_ℓ, the conditions (1–8) are kept as they were. The conditions (9–10) are replaced by the following:

(9″) $I_\ell \models \forall x B$ iff for every $d \in D$, $I_\ell \models B[x/\bar{d}]$,

(10″) $I_\ell \models \exists x B$ iff for some $d \in D$, $I_\ell \models B[x/\bar{d}]$.

Is the import semantics any different from the earlier semantics of states and environments in satisfying formulas?

Theorem 2.5 *Let x be a variable, t be a term, A be a formula, ℓ be a valuation under an interpretation $I = (D, \phi)$, and d be an element of D. Then, (a) $\ell[x \rightarrowtail d](t) = \ell(t[x/\bar{d}])$, (b) $I_{\ell[x \rightarrowtail d]} \models A$ iff $I_\ell \models A[x/\bar{d}]$.*

Proof For (a), use induction on the number of occurrences of function symbols in t. For (b), use induction on the number of occurrences of connectives and quantifiers in A. The crucial steps are when A is in either of the forms $\forall y B$ or $\exists y B$. The second case is similar to the first. In the first case, we have two subcases: (i) $x = y$, (ii) $x \neq y$. In case (i),

$$\begin{aligned}
& I_{\ell[x \rightarrowtail d]} \models \forall x B \\
& \quad \text{iff } I_{\ell[x \rightarrowtail d][x \rightarrowtail e]} \models B \text{ for every } e \in D \\
& \quad \text{iff for every } e \in D,\ I_{\ell[x \rightarrowtail e]} \models B \quad (\text{as } \ell[x \rightarrowtail d][x \rightarrowtail e] = \ell[x \rightarrowtail e]) \\
& \quad \text{iff for every } e \in D,\ I_\ell \models B[x/\bar{e}\,] \quad (\text{by induction hypothesis}) \\
& \quad \text{iff } I_\ell \models \forall x B \\
& \quad \text{iff } I_\ell \models (\forall x B)[x/\bar{d}\,] \ (\text{as } x \text{ is no more a free variable}) \\
& \quad \text{iff } I_\ell \models A[x/\bar{d}\,].
\end{aligned}$$

In case (ii),

$$\begin{aligned}
& I_{\ell[x \rightarrowtail d]} \models \forall y B \\
& \quad \text{iff for every } e \in D,\ I_{\ell[x \rightarrowtail d][y \rightarrowtail e]} \models B \\
& \quad \text{iff for every } e \in D,\ I_{\ell[y \rightarrowtail e][x \rightarrowtail d]} \models B \\
& \qquad (\text{as } x \neq y \text{ gives } \ell[x \rightarrowtail d][y \rightarrowtail e] = \ell[y \rightarrowtail e][x \rightarrowtail d])
\end{aligned}$$

iff for every $e \in D$, $I_\ell[y \rightarrowtail e] \models B[x/\bar{d}]$ (by induction hypothesis)
iff for every $e \in D$, $I_\ell \models B[x/\bar{d}][y/\bar{e}]$ (by induction hypothesis)
iff $I_\ell \models \forall y(B[x/\bar{d}])$
iff $I_\ell \models (\forall yB)[x/\bar{d}]$ (as $x \neq y$ gives $\forall y(B[x/\bar{d}]) = (\forall yB)[x/\bar{d}]$)
iff $I_\ell \models A[x/\bar{d}]$.

Use Exercise 2.8 to complete the proof. ∎

***Exercise* 2.9** Using Theorem 2.5, show that the import semantics is no different from the earlier semantics, that is, for any formula A, the structure $I_\ell \models A$ iff the environment or the state $I_\ell \models A$.

***EXAMPLE* 2.8** Show that $\neg\forall xPx \equiv \exists x\neg Px$ in each of the three semantics of states, environments, and structures.

Solution Here, we briefly describe all the three approaches.

State: Let $I = (D, \phi)$ be an interpretation where D is a nonempty set and $P' \subseteq D$, a unary relation corresponding to the predicate P. Take a valuation ℓ which assigns $\ell(x) \in D$ to the variable x. The state $I_\ell \models \neg\forall xPx$ iff $I_\ell \not\models \forall xPx$ iff for some valuation m equivalent to ℓ along x, we have $I_m \not\models Px$ iff for some valuation m (equivalent to ℓ along x), $I_m \models \neg Px$ iff $I_\ell \models \exists x\neg Px$.

Environment: With the same I, ℓ as in the case of a state, the environment $I_\ell \models \neg\forall xPx$ iff $I_\ell \not\models \forall xPx$ iff for some $d \in D$, $I_{\ell[x\rightarrowtail d]} \not\models Px$ iff for some $d \in D$, $I_{\ell[x\rightarrowtail d]} \models \neg Px$ iff $I_\ell \models \exists x\neg Px$.

Structure: Take I and ℓ as above. The structure $I_\ell \models \neg\forall xPx$ iff $I_\ell \not\models \forall xPx$ iff for some $d \in D$, $I_\ell \not\models P\bar{d}$ iff for some $d \in D$, $I_\ell \models \neg P\bar{d}$ iff $I_\ell \models \exists x\neg Px$ as $\neg P\bar{d} = (\neg Px)[x/\bar{d}]$.

Henceforth, we will not distinguish between the states, environments, and structures.

2.5 Some Useful Consequences

You have seen various approaches to the semantics of first order logic. What then? You must be able to determine valid formulas, valid consequences, equivalences, and then check whether common intuitions such as deduction theorem, RAA, etc. hold in this logic. We start with a simple result.

Theorem 2.6 (Monotonicity) *Let Σ and Γ be sets of formulas with $\Sigma \subseteq \Gamma$ and A be a formula. If $\Sigma \models A$, then $\Gamma \models A$.*

Proof Let $\Sigma \models A$. If I_ℓ is a state satisfying Γ, then $I_\ell \models X$ for each $X \in \Gamma$. As $\Sigma \subseteq \Gamma$, $I_\ell \models X$ for each $X \in \Sigma$. Since $\Sigma \models A$, $I_\ell \models A$. We have shown that each state-model of Γ is also a state-model of A, proving $\Gamma \models A$. ∎

Theorem 2.7 (RAA) *Let Σ be a set of formulas and A be a formula. Then $\Sigma \models A$ iff $\Sigma \cup \{\neg A\}$ is unsatisfiable.*

Proof Assume that $\Sigma \models A$. Let $I_\ell \models \Sigma \cup \{\neg A\}$. Then $I_\ell \models \Sigma$ and $I_\ell \models \neg A$. This says that $I_\ell \not\models A$, contradicting $\Sigma \models A$. Hence there is no state I_ℓ such that $I_\ell \models \Sigma \cup \{\neg A\}$. That is, $\Sigma \cup \{\neg A\}$ is unsatisfiable.

Conversely, suppose that $\Sigma \cup \{\neg A\}$ is unsatisfiable. If $\Sigma \not\models A$, then there is a state I_ℓ such that $I_\ell \models \Sigma$ but $I_\ell \not\models A$. Then $I_\ell \models \neg A$, consequently, $I_\ell \models \Sigma \cup \{\neg A\}$, contradicting the fact that $\Sigma \cup \{\neg A\}$ is unsatisfiable. ▮

Theorem 2.8 (Deduction) *Let Σ be a set of formulas and A, B be formulas. Then, $\Sigma \models A \to B$ iff $\Sigma \cup \{A\} \models B$.*

Proof Using RAA, we can see that $\Sigma \cup \{\neg(A \to B)\}$ is unsatisfiable iff $\Sigma \cup \{A, \neg B\}$ is unsatisfiable. Equivalently, $\Sigma \cup \{\neg(A \to B)\}$ is satisfiable iff $\Sigma \cup \{A, \neg B\}$ is satisfiable. We show the stronger (Is it?) assertion that for any state I_ℓ, $I_\ell \models \Sigma \cup \{\neg(A \to B)\}$ iff $I_\ell \models \Sigma \cup \{A, \neg B\}$.

So, let $I_\ell \models \Sigma \cup \{\neg(A \to B)\}$. Then $I_\ell \models \Sigma$ and $I_\ell \models \neg(A \to B)$. The latter is the same, as asserting that $I_\ell \models \{A, \neg B\}$. Conversely, if $I_\ell \models \Sigma \cup \{A, \neg B\}$, then $I_\ell \models \Sigma$ and $I_\ell \models \{A, \neg B\}$, which then implies that $I_\ell \models \Sigma \cup \{\neg(A \to B)\}$. ▮

Theorem 2.9 (Equivalence Substitution) *Let A, B, C be formulas. Denote by $A[B := C]$ a formula obtained from A by replacing some (or no or all) occurrences of B, as a subformula of A, by C. If $B \equiv C$, then $A \equiv A[B := C]$.*

Proof Here is only an outline; develop a proof from it. If B does not occur in A, or it has not been replaced by C at all, then $A = A[B := C]$, and there is nothing to prove. Otherwise, let I_ℓ be a state (ℓ being a valuation under an interpretation $I = (D, \phi)$). Now, to see whether $I_\ell \models A$, how do you proceed? Depending upon the form of A, you may write down a sentence: $I_\ell \models A$ iff $\ldots S \ldots$.

Proceeding step by step, the sentence S will involve somewhere $I_\ell \models B$ or $I_\ell \not\models B$, as the case demands. And these satisfaction relations will be coming as many times as B occurs in A. Due to equivalences of B and C, we have $I_\ell \models B$ iff $I_\ell \models C$. And then, these satisfactions can be replaced by $I_\ell \models C$ or $I_\ell \not\models C$. Replace only those occurrences of $I_\ell \models B$ or of $I_\ell \not\models B$ which correspond to the replacements of C while you obtained $A[B := C]$. Clearly, retracing the steps in S, you reach at $I_\ell \models A[B := C]$. ▮

All the above theorems are propositional in nature; their proofs simply follow the lines of the proofs you have seen in PL. There are, however, some results which are special to FL. These results basically hold due to the difference between an interpretation and a state under an interpretation. For example, consider the formula Px. With this formula, we can associate two sentences by the use of quantifiers, such as $\forall x Px$ and $\exists x Px$. Now, if Px is satisfiable, it would mean that there is a state I_ℓ which satisfies it. If $\ell(x) = a$, then it would guarantee that $I \models \exists x Px$. On the other hand, if Px is valid, then every state I_ℓ satisfies Px. Then every interpretation I would be a model of $\forall x Px$. Moreover, for sentences, we need not consider states;

interpretations suffice due to the Relevance Lemma. This observation can be generalized to any formula rather than the special one, Px.

Let X be a formula having the only free variables $x_1, \ldots, x_m$. We will encapsulate this information by writing X as $X[x_1, \ldots, x_m]$. The **existential closure** of X, denoted by $\exists^* X$, is the formula obtained from X by existentially quantifying over all free variables in it, i.e.,

$$\exists^* X = \exists x_1 \exists x_2 \cdots \exists x_m X[x_1, x_2, \ldots, x_m]$$

The **universal closure** of X, denoted by $\forall^* X$, is the formula obtained from X by universally quantifying over all free variables in it, i.e.,

$$\forall^* X = \forall x_1 \forall x_2 \cdots \forall x_m X[x_1, x_2, \ldots, x_m]$$

For $m = 0$, the formula is a sentence, and by definition, we take, in this case, $X = \exists^* X = \forall^* X$. The above observation amounts to the following statement.

Theorem 2.10 *Let X be any formula and $\exists^* X$, $\forall^* X$ be its existential and universal closures, respectively. Then*

(a) X is satisfiable iff $\exists^* X$ is satisfiable.

(b) X is valid iff $\forall^* X$ is valid.

Proof Use induction on m, the number of free variables of X. In the basis step, when $m = 0$, we have $X = \exists^* X = \forall^* X$ and the statement follows. For induction, assume that for $m = k$, both the conclusions (a) and (b) above hold. Let X be a formula having $k + 1$ free variables, i.e., let $X = X[x_1, \ldots, x_{k+1}]$. Consider two formulas:

$$X_e = \exists x_{k+1} X[x_1, \ldots, x_{k+1}] \quad \text{and} \quad X_a = \forall x_{k+1} X[x_1, \ldots, x_{k+1}]$$

Let I be an interpretation and ℓ be a valuation under I. Now, $I_\ell \models X_e$ iff for some valuation ℓ' equivalent to ℓ along x_{k+1}, $I_{\ell'} \models X$. That is, if X_e is satisfiable, then with I_ℓ as one of its state models, we have a state-model of X showing that X is satisfiable. Conversely, if $I_\ell \models X$, then take ℓ' as ℓ (since ℓ is equivalent to ℓ along x_{k+1}), and then we see that $I_{\ell'} \models X$. Hence, X is satisfiable iff X_e is satisfiable. Using induction hypothesis, we have: X_e is satisfiable iff $\exists x_1 \cdots \exists x_k X_e$ is satisfiable. Now, $\exists^* X = \exists x_1 \cdots \exists x_k X_e$. This proves (a). For (b), similarly, the statement:

$I_\ell \models X_a$ iff for every valuation ℓ' equivalent to ℓ along $x_{k+1}, I_{\ell'} \models X$

does the job. ∎

***Exercise* 2.10** Complete case (b) in the proof of Theorem 2.10.

Is the formula $Px \lor \neg Px$ valid? By Theorem 2.10, it would be so provided $\forall x(Px \lor \neg Px)$ is valid, which you can show now. Alternatively, you check directly whether $I_\ell \models Px \lor \neg Px$. This is so since one of $\ell(x) \in P'$ or $\ell(x) \notin P'$ holds. Can you apply the same argument to decide whether the formula $\neg Pxy \land Qz \leftrightarrow \neg(Qz \to Pxy)$ is valid? Well, observe that you can get this formula from the proposition $\neg q \wedge p \leftrightarrow \neg(p \rightarrow q)$, if you replace p by Qz, q by Pxy. The proposition is a valid one. Thus the same argument as above is still applicable with a state I_ℓ. The next theorem shows that this can be done in a quite general setting. But we will have to talk about valid formulas and also valid propositions.

To minimize confusion, let us use the word **tautology** for the valid propositions, that is, a tautology is a valid formula where the only symbols used are propositional constants ($\top, \bot$ and 0-ary predicates), connectives and, possibly, punctuation marks. As PL is a fragment of FL, all PL-valid propositions are tautologies. For example, $\neg q \wedge p \leftrightarrow \neg(p \rightarrow q)$ is a tautology.

We have one more notational problem. We want to replace the propositional constants by formulas. How to write this replacements? Of course, you can use the symbol $[p := X]$ for substituting p by X. However, we had already used this notation in equivalence substitution, where all occurrences of p need not be replaced by X. We require here to replace every occurrence of p by X. We will have to use a different notation.

Let σ : PROP $\rightarrow$ FORM be a function from the set of propositions, PROP, to the set of formulas, FORM. For any proposition A, write:

$\sigma'(A)$ = the proposition obtained from A by replacing all occurrences of each propositional variable p by the formula $\sigma(p)$.

For example, if $\sigma(p) = Px$, $\sigma(q) = Qxy$, $A = p \vee \neg p \rightarrow q$, then, $\sigma'(A) = Px \vee \neg Px \rightarrow Qxy$. In contrast to any substitution, we will refer this substitution as the **uniform substitution**. Note that, here it does not matter whatever σ assigns to other propositional variables. Now, if $i(p) = 1, i(q) = 0$, for a PL-interpretation i, then correspondingly, for a state I_ℓ, we have $I_\ell \models Px$ and $I_\ell \not\models Qxy$; consequently, $i(A) = 1$ and also $I_\ell \models \sigma'(A)$. This would happen only when i and I_ℓ are so related.

Theorem 2.11 *Let A be a proposition, i be any PL-interpretation (i.e., either $i(A)$ is 0 or 1) and I_ℓ be a state such that for every propositional variable p, $i \models p$ iff $I_\ell \models \sigma(p)$. Then $i \models A$ iff $I_\ell \models \sigma'(A)$.*

Proof Use induction on $n(A)$, the number of occurrences of connectives in A. In the basis step, when $n(A) = 0$, you have $A = \top, \bot$, or p, for some propositional variable p. Then $\bar{\sigma}(A) = \top, \bot$, or $\sigma(p)$, respectively. Here, the assumption coincides with the conclusion. Lay out the induction hypothesis that, if $n(A) < m$, then $i \models A$ iff $I_\ell \models \sigma'(A)$.

Let A be a proposition with $n(A) = m \geq 1$. Then, A is in one of the following forms:

(i) $\neg B$, (ii) $(B \wedge C)$, (iii) $(B \vee C)$, (iv) $(B \rightarrow C)$, (v) $(B \leftrightarrow C)$.

In case (i), $\sigma'(A) = \neg\sigma'(B)$. Since $n(B) < m$, by induction hypothesis, $i \models B$ iff $I_\ell \models \sigma'(B)$. Now, $i \models A$ iff $i \not\models B$ iff $I_\ell \not\models \sigma'(B)$ iff $I_\ell \models \sigma'(A)$. Other cases are proved analogously. (Prove them.) ∎

Now we can go back to the result that when propositional variables are substituted by arbitrary formulas uniformly in a valid proposition, a valid formula results.

Theorem 2.12 (Uniform Substitution in a Tautology) *Let A be a tautology. Let σ be a mapping that associates propositional variables to formulas. Let $\sigma'(A)$ be the result of uniformly substituting each occurrence of each propositional variable p by $\sigma(p)$ in A. Then $\models \sigma'(A)$.*

Proof Let I be any interpretation and ℓ be any valuation under I. Let p be any propositional variable. Then $\sigma(p)$ is a formula. Now, either $I_\ell \models \sigma(p)$ or $I_\ell \not\models \sigma(p)$. Define a PL-interpretation i such that $i(p) = 1$ if $I_\ell \models \sigma(p)$, and $i(p) = 0$ if $I_\ell \not\models \sigma(p)$. Since A is a valid proposition, $i \models A$. By Theorem 2.11, $I_\ell \models \sigma'(A)$. We have shown that any arbitrary state satisfies $\sigma'(A)$. hence $\sigma'(A)$ is valid. ∎

Uniform substitution in a tautology allows you to conclude that the formula $Px \vee \neg Px$ is valid and that $Pxy \to (Qxyz \to Pxy)$ is valid, and so on. You can get many valid formulas from the laws in Theorem 1.8. In addition, we have some more laws involving quantifiers as given in the following theorem. They will be helpful in making calculations and showing some other formulas and consequences to be valid.

Theorem 2.13 *The following statements hold for any formulas X, Y, variables x, y, and terms r, s, t. (Whenever $X[x/t]$ appears, it is assumed that t is free for x in X; new constant means a constant that has not appeared in the context, etc.)*

(a) LAW OF CONSTANTS: $\models X$ iff $X \equiv \top$, $\models \forall x(\bot \to X)$, $\exists x(\bot \wedge X) \equiv \bot$.

(b) LAW OF EQUALITY: $\models (t \approx t)$, $(s \approx t) \equiv (t \approx s)$,

$\{r \approx s, s \approx t\} \models (r \approx t)$, $\{s \approx t, X[x/s]\} \models X[x/t]$,

If x does not occur in t, then

$\forall x((x \approx t) \to X) \equiv X[x/t]$, $\exists x((x \approx t) \wedge X) \equiv X[x/t]$.

If x does not occur in X, then $\forall x X \equiv X$, $\exists x X \equiv X$.

(c) LAW OF COMMUTATIVITY:

$\forall x \forall y X \equiv \forall y \forall x X$, $\exists x \exists y X \equiv \exists y \exists x X$, $\models \exists x \forall y X \to \forall y \exists x X$.

(d) LAW OF DISTRIBUTIVITY:

$\forall x(X \wedge Y) \equiv \forall x X \wedge \forall x Y$, $\exists x(X \vee Y) \equiv \exists x X \vee \exists x Y$,

$\models \forall x X \vee \forall x Y \to \forall x(X \vee Y)$, $\models \exists x(X \wedge Y) \to \exists x X \wedge \exists x Y$.

If x does not occur in X, then

$\forall x(X \vee Y) \equiv X \vee \forall x Y$, $\exists x(X \wedge Y) \equiv X \wedge \exists x Y$,

$\forall x(X \to Y) \equiv X \to \forall x Y$, $\exists x(X \to Y) \equiv X \to \exists x Y$,

$\forall x(Y \to X) \equiv \exists x Y \to X$, $\exists x(Y \to X) \equiv \forall x Y \to X$.

(e) LAW OF RENAMING: $\forall xX \equiv \forall yX[x/y]$, $\exists xX \equiv \exists yX[x/y]$

(f) LAW OF EMPTY QUANTIFICATION:

If x does not occur in X, then $\forall xX \equiv X$, $\exists xX \equiv X$.

(g) LAW OF DE MORGAN:

$\neg\forall xX \equiv \exists x\neg X$, $\neg\exists xX \equiv \forall x\neg X$, $\forall xX \equiv \neg\exists x\neg X$, $\exists xX \equiv \neg\forall x\neg X$.

(h) LAW OF GENERALIZATION: *If* $\models X$, *then* $\models \forall xX$, $\models X[x/t] \to \exists xX$.

(i) LAW OF SPECIFICATION:

$\models \forall xX \to X[x/t]$, $\models \exists xX \to X[x/c]$ *for a new constant c.*

(j) LAW OF UNIFORM SUBSTITUTION: *Let σ be a function that associates positional variables to formulas. Let $\sigma'(X)$ be the result of uniformly substituting each occurrence of each propositional variable p by $\sigma(p)$ in X. If $\models X$, then $\models \sigma'(X)$, where $\sigma(p)$ is such a formula that no free variables of it becomes bound in $\sigma'(X)$.* ∎

***Exercise* 2.11** Check the correctness of the laws in (a) to (i) above.

The laws of constants and equality are straightforward. But the last two equivalences in the law of equality need some attention; they are:

$$\forall x(x \approx t \to X) \equiv X[x/t] \text{ and } \exists x(x \approx t \wedge X) \equiv X[x/t]$$

Here, you require a condition that x must not occur in t. What happens if x occurs in t? With $t = x, (x \approx t) \equiv \top$ and $(x \approx t \to X) \equiv X$, so that you have $\forall xX \equiv X$ and $\exists xX \equiv X$, which are obviously wrong. Check this with the law of empty quantification.

Next, look at the law of commutativity, where you have only an implication, and not an equivalence. This is because $\forall y\exists xX \to \exists x\forall yX$ is invalid. Why? Let $X = Pxy$, and revisit Example 2.7. Next, you find something interesting in the law of distributivity. In general, the formulas $\forall(X \vee Y) \to \forall xX \vee \forall xY$ and $\exists xX \wedge \exists xY \to \exists x(X \wedge Y)$ are invalid. For example, with $X = Px$ and $Y = \neg Px$, you have $\forall x(X \vee Y) \equiv \top$. The formula $\forall xX \vee \forall xY$ is, now, $\forall xPx \vee \forall x\neg Px$. Is this formula valid? Take an interpretation $I = (\{2,3\}, P')$ with $P' = \{2\}$. Then,

$$I \models \forall xPx \vee \forall x\neg Px \text{ iff } (2 \in P' \text{ and } 3 \in P') \text{ or } (2 \notin P' \text{ and } 3 \notin P'),$$

which clearly does not hold. This shows that $\forall(X \vee Y) \to \forall xX \vee \forall xY$ is invalid. Why is $\exists xX \wedge \exists xY \to \exists x(X \wedge Y)$ invalid? You can try to show it the same way as we did for $\forall(X \vee Y) \to \forall xX \vee \forall xY$. But, consider the following exercise.

***Exercise* 2.12** Using the laws of constants and De Morgan show that, if $X = Px$ and $Y = \neg Px$, then $\exists xX \wedge \exists xY \to \exists x(X \wedge Y) \equiv \forall xPx \vee \forall x\neg Px$.

Now you are able to appreciate the condition "x does not occur in X" in the next two laws of distributivity. Do you? In the laws of specification, think about the 'import semantics' where, for each object in a domain of an interpretation, we had introduced a new constant to our language. This would explain the phrase 'a new constant c' appearing in $\exists xX \to X[x/c]$. We may not have, in fact, sufficient terms in our language to name any arbitrary object; thus new names are invented. Similarly, in the law of generalization, we have a weird looking expression: 'if $\models X$, then $\models \forall xX$'. It is not the same as $X \models \forall xX$ (nor as $\models X \to \forall xX$; recollect deduction theorem). Why? The former says that

"If each state I_ℓ satisfies X, then each state I_ℓ satisfies $\forall xX$."

While the latter says that

"For each state I_ℓ, whenever I_ℓ satisfies X, I_ℓ also satisfies $\forall xX$."

The point is that the former can vacuously hold whereas the latter may not. For example, with $X = Px$, 'if $\models Px$ then $\models \forall xX$' holds since $\forall xPx$ is the universal closure of Px. But, $Px \models \forall xPx$ does not hold as there can be a state I_ℓ which satisfies Px and, at the same time, falsifies $\forall xPx$. To see this, take the domain of I as the natural numbers and let ℓ assign x to 2. Suppose that P is interpreted as the set of prime numbers. Then Px would mean the sentence "2 is a prime number", which holds; while $\forall xPx$ would mean, "all natural numbers are prime", which is clearly wrong.

2.6 Calculations

With the use of the laws listed in Theorem 2.13, you can extend the use of calculations for first order logic. As in PL, a **calculation** will look like:

$$C_0 \oplus_1 C_1 \oplus_2 \ldots \oplus_m C_m$$

where $\oplus_i \in \{\equiv, \models\}$, and every step $C_{i-1} \oplus_i C_i$ must be an instance of a law $E_1 \oplus_i E_2$. The calculation is taken as a proof of the metastatement $C_0 \odot C_m$, where $\odot = \equiv$ if all $\oplus_i = \equiv$, else, $\odot = \models$. Moreover, a proof of $\models Z$ can be a proof of one of the statements $Z \equiv \top$ or $\top \models Z$. A proof of $\models Y \to Z$ can also be any of the proofs of $Y \models Z$, due to deduction theorem. Similarly, a proof of $\models Y \leftrightarrow Z$ can also be a proof of $Y \equiv Z$. A proof of the consequence $\{A_1, \ldots, A_n\} \models A$ is a proof of $A_1 \wedge \cdots \wedge A_n \models A$. Another alternative for a proof of a consequence is to use $A_i \equiv \top$ as additional *local laws* and then prove $\models A$. A calculation is justified provided that each step is justified. Remember that each step is justified in PL due to Leibniz Rule (Theorem 1.10). Here also we give a version of this metastatement for FL.

Theorem 2.14 (Leibniz Rule) *Let the propositional variables $p_1, \ldots, p_n$ occur in the formulas E_1, E_2, and $B_1, \ldots, B_n$ be any formulas. Then,*

(a) $E_1 \equiv E_2$, implies

$$E_1[p_1 := B_1, \ldots, p_n := B_n] \equiv E_2[p_1 := B_1, \ldots, p_n := B_n].$$

(b) $E_1 \models E_2$, implies

$$E_1[p_1 := B_1, \ldots, p_n := B_n] \models E_2[p_1 := B_1, \ldots, p_n := B_n].$$

Proof Modify the proof of Theorem 1.10 suitably. Alternatively, you may use uniform substitution in tautologies to get this. ∎

In defining a step of a calculation, we have used the phrase 'an instance of', which is so in the sense of Leibniz. That is,

$$E_1[p_1 := B_1, \ldots, p_n := B_n] \equiv E_2[p_1 := B_1, \ldots, p_n := B_n]$$

is an instance of $E_1 \equiv E_2$. Similarly,

$$E_1[p_1 := B_1, \ldots, p_n := B_n] \models E_2[p_1 := B_1, \ldots, p_n := B_n]$$

is also an instance of $E_1 \models E_2$. This rule of Leibniz will be used in every step of a calculation implicitly.

Now, how to use a law such as 'generalization' in a calculation? For example, we have a tautology $p \to p$. In place of p, you may substitute Px to get $Px \to Px$. Then, you generalize to conclude that $\forall x(Px \to Px)$ is valid. Can these steps be written as a calculation? Our strategy here is to start from the special sentence $\top$, which stands for truth and proceed towards deriving $\forall x(Px \to Px)$. Here is an attempt:

$\top$	[Implication]
$\equiv\ p \to p$	[Theorem 2.14]
$?\ \ Px \to Px$	[Generalization]
$!\ \ \forall x(Px \to Px)$	

What symbol you must use in place of '?' ? Can you use $\equiv$? You cannot, since $p \to p \not\equiv Px \to Px$. What Theorem 2.14 says is that, "if $\models p \to p$, then $\models Px \to Px$." The same argument tells you that you cannot use $\models$ in place of '!'. So? The calculation is, after all, correct. Let us write the symbol '$\Rightarrow$' whenever we use Theorem 2.14. Thus, in our definition of a calculation, we can have $\oplus = \equiv$ or $\models$ or $\Rightarrow$. The symbol $\Rightarrow$ will be used whenever a metastatement is used in the sense that:

$X \Rightarrow Y$ will abbreviate the statement "if $\models X$, then $\models Y$".

Remember that whenever we have a calculation $C_0 \oplus_1 C_1 \oplus_2 \ldots \oplus_m C_m$, which uses the new symbol $\Rightarrow$, and C_0 is valid, we would then conclude that C_m is valid. The above calculation would be written as

$\top$	[Implication]
$\equiv\ p \to p$	[Theorem 2.14]
$\Rightarrow\ Px \to Px$	[Generalization]
$\Rightarrow\ \forall x(Px \to Px)$	

Since $\top$ is valid, the calculation shows that the formula $\forall x(Px \to Px)$ is valid. However, in such a case, where the symbol $\Rightarrow$ is used in a calculation, with premises in a set Σ and with a conclusion X, we have a proof of the metastatement:

If all the premises in Σ are valid, then the conclusion X is valid.

This is weaker than $\Sigma \models X$, as the latter demands that for every state I_ℓ, if $I_\ell \models Y$ for each $Y \in \Sigma$, then $I_\ell \models X$. Semantically, we will write such a consequence as $\Sigma \Rightarrow X$ instead of $\Sigma \models X$. If all the premises in Σ are sentences, then, of course, $\Sigma \Rightarrow X$ and $\Sigma \models X$ will coincide. In case of proving validity of a formula X, a calculation showing $\top \Rightarrow X$ suffices.

At this point, redefine a *calculation* formally incorporating the new symbol $\Rightarrow$. Let us illustrate this with some more examples. We use abbreviations, such as 'Prop' for 'a (propositional) tautology', 'Subs Taut' for 'Uniform Substitution in a Tautology', 'Gene' for 'Generalization', 'Spec' for 'Specification', 'Hyp' for 'Hypothesis', 'Mod Pon' for Modus Ponens.

***EXAMPLE* 2.9** Show that $\models \forall x((\neg Px \to \neg Qx) \to (Qx \to Px))$.
Solution

$$\begin{array}{ll}
\top & \text{[Prop]} \\
\equiv\ (\neg p \to \neg q) \to (q \to p) & \text{[Subs Taut]} \\
\Rightarrow\ (\neg Px \to \neg Qx) \to (Qx \to Px) & \text{[Gene]} \\
\Rightarrow\ \forall x((\neg Px \to \neg Qx) \to (Qx \to Px)) &
\end{array}$$

***EXAMPLE* 2.10** Show that $\models (\forall x(Px \to Qx) \to (\exists xPx \to \exists xQx))$.
Solution It looks easier to apply deduction theorem. So, we show that $\{\forall x(Px \to Qx), \exists xPx\} \models \exists xQx$.

$$\begin{array}{ll}
\exists xPx & \text{[Spec]} \\
\quad\models\ Pc & \text{[Prop]} \\
\quad\equiv\ Pc \wedge \top & \text{[Hyp]} \\
\quad\equiv\ Pc \wedge \forall x(Px \to Qx) & \text{[Spec]} \\
\quad\equiv\ Pc \wedge (Pc \to Qc) & \text{[Mod Pon]} \\
\quad\models\ Qc & \text{[Gene]} \\
\quad\models\ \exists xQx &
\end{array}$$

***EXAMPLE* 2.11** Show that $\{\exists xPx, \forall x\forall y(Px \to Qy)\} \models \forall yQy$.
Solution

$$\begin{array}{ll}
\exists xPx & \text{[Spec]} \\
\quad\models\ Pc & \text{[Hyp \& Const]} \\
\quad\models\ Pc \wedge \forall x\forall y(Px \to Qy) & \text{[Spec]} \\
\quad\models\ Pc \wedge \forall y(Pc \to Qy) & \text{[Dist]} \\
\quad\models\ Pc \wedge (Pc \to \forall yQy) & \text{[Mod Pon]} \\
\quad\equiv\ \forall yQy &
\end{array}$$

EXAMPLE **2.12** Show the validity of the argument:
All lions are animals. Therefore, all heads of lions are heads of animals.

Solution To translate the argument into an FL-consequence, let

Lx : x is a lion, Ax : x is an animal, Hxy : x is a head of y.

Then, "x is a head of a lion" is symbolized as $\exists y(Ly \wedge Hxy)$, etc. You get the consequence as $\forall x(Lx \to Ax) \models \forall x(\exists y(Ly \wedge Hxy) \to \exists y(Ay \wedge Hxy))$. Here, it seems easier to use RAA, since the conclusion has predicate H which is absent in the premise. So we show that the set

$$\{\forall x(Lx \to Ax), \neg\forall x(\exists y(Ly \wedge Hxy) \to \exists y(Ay \wedge Hxy))\}$$

is unsatisfiable. That is, we will have a calculation which uses the premises in this set and ending at $\bot$. We start with one of the premises.

$$\begin{array}{ll}
\neg\forall x(\exists y(Ly \wedge Hxy) \to \exists y(Ay \wedge Hxy)) & \text{[De Mor \& Prop]} \\
\quad\equiv\ \exists x(\exists y(Ly \wedge Hxy) \wedge \neg\exists y(Ay \wedge Hxy)) & \text{[Spec, } a \text{ is new]} \\
\quad\models\ \exists y(Ly \wedge Hay) \wedge \neg\exists y(Ay \wedge Hay) & \text{[Spec, } b \text{ is new]} \\
\quad\models\ Lb \wedge Hab \wedge \neg\exists y(Ay \wedge Hay) & \text{[De Mor]} \\
\quad\equiv\ Lb \wedge Hab \wedge \forall y(\neg Ay \vee \neg Hay & \text{[Spec]} \\
\quad\models\ Lb \wedge Hab \wedge (\neg Ab \vee \neg Hab) & \text{[Prop]} \\
\quad\equiv\ (Lb \wedge Hab \wedge \neg Ab) \vee (Lb \wedge Hab \wedge \neg Hab) & \text{[Prop]} \\
\quad\equiv\ (Lb \wedge Hab \wedge \neg Ab) \vee \bot & \text{[Prop]} \\
\quad\equiv\ (Lb \wedge Hab \wedge \neg Ab) & \text{[Hyp]} \\
\quad\equiv\ (Lb \wedge Hab \wedge \neg Ab) \wedge \forall x(Lx \to Ax) & \text{[Spec]} \\
\quad\models\ (Lb \wedge Hab \wedge \neg Ab) \wedge (Lb \to Ab) & \text{[Mod Pon]} \\
\quad\models\ (Hab \wedge \neg Ab \wedge Ab) & \text{[Prop]} \\
\quad\equiv\ \bot &
\end{array}$$

EXAMPLE **2.13** The following set of formulas is unsatisfiable:

$$\begin{array}{l}
\{\forall x\forall y\forall z((Px \wedge Qy \wedge Rzy \wedge Syx) \to Rzx), \\
\ \neg\exists x\exists y\exists z(Px \wedge Py \wedge \neg(x \approx y) \wedge Rzx \wedge Rzy), \\
\ \ Qa \wedge Rba \wedge Sac \wedge Pc \wedge Pd \wedge \neg(c \approx d), Red \wedge (e \approx b)\}.
\end{array}$$

Solution

$$\begin{array}{lr}
\forall x\forall y\forall z((Px \wedge Qy \wedge Rzy \wedge Syx) \to Rzx) & \text{[Spec,}[x/c, y/a, z/b]] \\
\quad\equiv Pc \wedge Qa \wedge Rba \wedge Sac \to Rbc & \text{[Hyp, Prop]} \\
\quad\equiv (Pc \wedge Qa \wedge Rba \wedge Sac \to Rbc) & \\
\qquad\wedge \neg\exists x\exists y\exists z(Px \wedge Py \wedge \neg(x \approx y) \wedge Rzx \wedge Rzy) & \text{[De Mor, Prop]} \\
\quad\models (Pc \wedge Qa \wedge Rba \wedge Sac \to Rbc) & \\
\qquad\wedge \forall x\forall y\forall z(Px \wedge Py \wedge Rzx \wedge Rzy \to (x \approx y)) & \text{[Spec,}[x/d, y/c, z/e]]
\end{array}$$

$$\begin{aligned}
&\equiv (Pc \wedge Qa \wedge Rba \wedge Sac \to Rbc) \\
&\quad \wedge (Pd \wedge Pc \wedge Red \wedge Rec \to (d \approx c)) && \text{[Hyp]} \\
&\models (Pc \wedge Qa \wedge Rba \wedge Sac \to Rbc) \\
&\quad \wedge (Pd \wedge Pc \wedge Red \wedge Rec \to (d \approx c)) \wedge Qa \wedge Rba \\
&\quad \wedge Sac \wedge Pc \wedge Pd \wedge \neg(c \approx d) \wedge Red \wedge (e \approx b) && \text{[Mod Pon]} \\
&\models Rbc \wedge (Pd \wedge Pc \wedge Red \wedge Rec \to (d \approx c)) \wedge Qa \wedge Rba \\
&\quad \wedge Sac \wedge Pc \wedge Pd \wedge \neg(c \approx d) \wedge Red \wedge (e \approx b) && \text{[Equality]} \\
&\models Rbc \wedge (Pd \wedge Pc \wedge Rbd \wedge Rbc \to (d \approx c)) \wedge Qa \wedge Rba \\
&\quad \wedge Sac \wedge Pc \wedge Pd \wedge \neg(c \approx d) \wedge Rbd \wedge (e \approx b) && \text{[Mod Pon, Prop]} \\
&\models (d \approx c) \wedge \neg(c \approx d) && \text{[Equality]} \\
&\models (d \approx c) \wedge \neg(d \approx c) && \text{[Prop]} \\
&\models \bot
\end{aligned}$$

In the last example, the specification choices were very important. Had you chosen something else instead of the chosen ones, you would not have ended at $\bot$ (but of course, you could use them later). The choices were motivated by the occurrences of the constants in the predicates. Only experience will help you make the right choice. Now, you should try Problems 31−33 at the end of the chapter. Next, we proceed towards discovering possible normal forms in FL as we had cnfs and dnfs in PL.

2.7 Normal Forms

Consider the sentence $\forall x(Px \to Qx)$. Since $Px \to Qx$ is equivalent to $\neg Px \vee Qx$, we can have the equivalent sentence $\forall x(\neg Px \vee Qx)$. Then, by specification, this would entail $\neg Pa \vee Qa$. Similarly, this also entails the other specification instances: $\neg Pb \vee Qb$, $\neg Pc \vee Qc$, and so on. Finally, you can have a conjunction of all these instances, i.e.,

$$(\neg Pa \vee Qa) \wedge (\neg Pb \vee Qb) \wedge (\neg Pc \vee Qc) \wedge \cdots$$

But what does this $\cdots$ mean? Is it terminating somewhere or is it like "$\mathbb{N}$ has the elements $0, 1, 2, \ldots$"? If it is like the latter, then it is not a formula. Then, how to have a normal form for arbitrary formulas? How to get rid of the quantifiers in a meaningful way? One way is just to drop a quantifier. But when is it permitted and in what sense?

You know that $\models \forall x(Px \to Qx)$ iff $\models (Px \to Qx)$, though such a thing does not hold for existential quantifier. In this case, at least when all the quantifiers are universal, you can drop them, and this will still preserve validity. Then, what can you do for an existential quantifier, say, for the formula $\exists x(Px \wedge Qx)$? Well, it is simply $Pc \wedge Qc$, where c is a new constant. This seems to be all right. What about the formula $\forall x \exists y(Pxy \to Qxy)$?

This formula, a sentence, says that 'for each x, there is a y $\cdots$', which means, so to speak, that y may depend in some way on x so that the relation $Pxy \to Qxy$ holds. How to express dependence? Just like new constants, use a new function symbol! The sentence will be represented as $Pxf(x) \to Qxf(x)$. What about $\forall x\exists y(Px \wedge Qxy \to \forall z\exists yRyz)$? There are two occurrences of $\exists y$. The variable y of this first occurrence, which includes Qxy in its scope, must depend upon x while the second one should depend upon z, and not on x. In fact, by renaming, you can see that the latter y is not within the scope of the first $\exists y$ as the formula is equivalent to $\forall x\exists y(Px \wedge Qxy \to \forall z\exists uRuz)$. This says that the y in the subformula $\forall z\exists yRyz$ is not a free variable, and hence is not bound by the first occurrence of $\exists y$. While going for normal forms, we must take care of all these eventualities.

Moreover, in the formula $\exists y\forall xPxy$, the variable y does not depend upon x. So, it is not clear as in the renamed formula above whether the variable u depends upon x or not. We need to express it in the form $\forall x\exists y\forall z\exists u(\cdots)$, where all the quantifiers should occur only in the beginning. Such a form is called a **prenex form** of a formula. For example, $\forall xPx \to \forall yQy$ is not in prenex form, but it can be put in a prenex form such as $\forall x\forall y(Px \to Qy)$ or as $\forall y\forall x(Px \to Qy)$, using the laws of distributivity. Note that the prenex forms are equivalent to the original formula.

What will be the prenex form of the formula $\forall xPx \to \forall xQx$? You cannot, of course, write $\forall xPx \to \forall xQx \equiv \forall x(Px \to Qx)$ as x occurs in $\forall xQx$. (The law of distributivity says that $\forall x(X \to Y) \equiv \forall xX \to Y$ if x does not occur in Y.) However, you can use renaming, and then, equivalence substitution will allow you to write the formula equivalently as $\forall xPx \to \forall yQy$, consequently, distributivity will give you $\forall x(Px \to \forall yQy)$, and then, $\forall x\exists y(Px \to Qy)$, or $\exists y\forall x(Px \to Qy)$. Such renaming of bound variables which allows you to use the laws towards conversion to a prenex form is known as *rectification.* The following definition formalizes these notions.

Definition 2.5 A formula is called **rectified** iff no variable in it is both free and bound, and each occurrence of a quantifier uses a different variable. A formula is said to be in **prenex form** iff all occurrences of quantifiers in it are in the beginning, i.e., if it is in the form $Q_1x_1Q_2x_2\cdots Q_nx_nX$, where each $Q_i \in \{\forall, \exists\}$, and X contains no quantifiers. The string of quantifiers and variables, $Q_1x_1Q_2x_2\cdots Q_nx_n$, is called the **prefix** and X is called the **matrix** of the prenex form.

For example, $\forall x(Px \to Qy \wedge \exists xQx)$ is not a rectified formula as there are two occurrences of quantifiers using the same variable x. In the formula $\forall x(Px \to Qy \wedge \exists yQy)$, different occurrences of quantifiers use different variables; but this is also not a rectified formula because y is both a free and bound variable of the formula. (Note that free variables cannot be renamed if you want to preserve equivalence.) However, the last formula is equivalent to $\forall x(Px \to Qy \wedge \exists zQz)$, which is rectified. Again, this is not in prenex form, because there are occurrences of other symbols to the left of an occurrence of a quantifier, that is, to the left of $\exists z$. However, $\forall x \exists z(Px \to Qy \wedge Qz)$ is in prenex form, where the prefix is $\forall x \exists z$ and the matrix is $(Px \to Qy \wedge Qz)$. So, what do you expect? Can you bring any formula to a prenex form, say by a series of equivalences, by a calculation?

Theorem 2.15 (Prenex Form) *For any formula X, there is a formula Y in prenex form such that $X \equiv Y$.*

Proof Use induction on the number of occurrences of quantifiers in X. You can do it yourself after so many induction proofs! ∎

***Exercise* 2.13** Prove Theorem 2.15. Can you talk about *the* prenex form of a formula?

It will be more useful to have a constructive procedure for prenex form conversion. The procedure *PrenForm* below is one such.

PROCEDURE : *PrenForm*

Input: A formula X

Output: A prenex form formula equivalent to X

1. Eliminate $\leftrightarrow$ using the law $A \leftrightarrow B \equiv (A \to B) \wedge (B \to A)$ on all subformulas of X.
2. Rename the bound variables to rectify X (After this step, X is assumed to be a rectified formula).
3. Move $\neg$ to precede the predicates by using the equivalences:

$$\neg\neg A \equiv A, \quad \neg(A \vee B) \equiv \neg A \wedge \neg B, \quad \neg(A \wedge B) \equiv \neg A \vee \neg B$$
$$\neg\exists x A \equiv \forall X \neg A, \quad \neg\forall x A \equiv \exists x \neg A, \quad \neg(A \to B) \equiv A \wedge \neg B$$

4. Pull out the quantifiers using commutativity of $\wedge$ and $\vee$, and the equivalences (x does not occur in B as the formula is rectified):

$$\forall x A \to B \equiv \exists x(A \to B), \quad \exists x A \to B \equiv \forall x(A \to B)$$
$$B \to \forall x A \equiv \forall x(B \to A), \quad B \to \exists x A \equiv \exists x(B \to A)$$
$$\forall x A \wedge B \equiv \forall x(A \wedge B), \quad \exists x A \wedge B \equiv \exists x(A \wedge B)$$
$$\forall x A \vee B \equiv \forall x(A \vee B), \quad \exists x(A \vee B) \equiv \exists x(A \vee B)$$

***EXAMPLE* 2.14** Construct a prenex form formula equivalent to

$$A = \exists z(Pxy \to \neg\forall y(Qy \wedge Ryz)) \wedge (Qx \to \forall x Sx)$$

Solution The connective $\leftrightarrow$ does not occur in A; so, we rectify A by renaming the bound variables if needed. Both y and x occur free and also bound in A. Rename the bound variables: y as v and x as u. The formula so obtained is

$$B = \exists z(Pxy \to \neg\forall v(Qv \wedge Rvz)) \wedge (Qx \to \forall u Su)$$

Now, B is rectified. Start moving $\neg$ near to the predicates, by using the equivalences in Step 3 in *PrenForm*. You obtain an equivalent formula

$$C = \exists z(Pxy \to \exists v(\neg Qv \lor \neg Rvz)) \land (Qx \to \forall u Su)$$

Next, pull the quantifiers to the left using the equivalences in Step 4. You get the formula

$$G = \forall u \exists z \exists v((Pxy \to \neg Qv \lor \neg Rvz) \land (Qu \to Su))$$

This is in prenex form with prefix $\forall u \exists z \exists v$ and the matrix as

$$(Pxy \to \neg Qv \lor \neg Rvz) \land (Qu \to Su)$$

Note that you could also have brought C to the formula

$$H = \exists z \exists v \forall u((Qx \to Su) \land (Pxy \to \neg Qv \lor \neg Rvz))$$

Are they really equivalent? Argue with the semantics of FL.

***Exercise* 2.14** Can you see why the procedure *PrenForm* eliminates $\leftrightarrow$ before rectifying the formula? Can you write a procedure for prenex form conversion without first rectifying the formula?

***EXAMPLE* 2.15** Find a prenex form for the formula

$$\exists x(Px \to \neg\exists y(Py \to (Qx \to Qy)) \land \forall x(Px \leftrightarrow \forall y Qz)).$$

Solution

$$\begin{aligned}
&\exists x(Px \to \neg\exists y(Py \to (Qx \to Qy)) \land \forall x(Px \leftrightarrow \forall y Qz))\\
&\quad\equiv\ \exists x(Px \to \neg\exists y(Py \to (Qx \to Qy))\\
&\qquad\quad \land \forall x((Px \to \forall y Qz) \land (\forall y Qz \to Px)))\\
&\quad\equiv\ \exists x(Px \to \forall y(Py \land Qx \land \neg Qy)\\
&\qquad\quad \land \forall u(\forall v(Pu \to Qz) \land \exists w(Qz \to Pu)))\\
&\quad\equiv\ \exists x(\forall y(Px \to (Py \land Qx \land \neg Qy))\\
&\qquad\quad \land \forall u \forall v \exists w((Pu \to Qz) \land (Qz \to Pu)))\\
&\quad\equiv\ \exists x \forall y \forall u \forall v \exists w((Px \to Py \land Qx \land \neg Qy) \land (Pu \to Qz) \land (Qz \to Pu))
\end{aligned}$$

which is in prenex form with prefix $\exists x \forall y \forall u \forall v \exists w$ and the matrix as

$$((Px \to Py \land Qx \land \neg Qy) \land (Pu \to Qz) \land (Qz \to Pu))$$

***Exercise* 2.15** Try other ways of bringing to prenex form the formula in Example 2.15, where the prefix may be having a different order of occurrences of $\exists$ and $\forall$, proceeding in a different way after the third line in the solution. Give semantic arguments as to why your answer is equivalent to the one above.

Look at the matrix of a prenex form. You can further use the propositional tautologies to convert it into either a conjunction of disjunctions or a disjunction of conjunctions, just like the conversion of propositions into cnf or dnf. For this purpose, you have to redefine (or extend to FL) the notions of literals and clauses. In FL, you have the atomic formulas as propositional constants $\top$ and $\bot$, and also the formulas of the form $P(t_1, \ldots, t_n)$

for n-ary predicates P and terms $t_1, \ldots, t_n$. A **literal** is again either an atomic formula or negation of an atomic formula. A **conjunctive clause** is a conjunction of literals, and a **disjunctive clause** is a disjunction of literals. A **cnf** is a conjunction of disjunctive clauses, and a **dnf** is a disjunction of conjunctive literals. Using propositional tautologies, you can convert the matrix of a prenex form to both a cnf and a dnf. Then, by equivalence substitution, the prenex form will be equivalent to one, with its matrix in cnf or dnf. A formula in prenex form with its matrix as a cnf is said to be in **pcnf** or in prenex conjunctive normal form, and one whose matrix is in dnf is in **pdnf** or in prenex disjunctive normal form. Both pcnf and pdnf are commonly called **prenex normal forms** or **pnf**. With your experience with prenex forms and the propositional normal forms, you can easily prove the following theorem.

Theorem 2.16 (Prenex Normal Form) *Every formula is equivalent to one in pcnf and one in pdnf.*

What we want next is to get rid of the quantifiers. That is, of course, a familiar notion. When you want to prove a statement of the form $\forall x Px$, you usually start as "Let x be arbitrary. ... We see that Px holds. Hence $\forall x Px$ ". What we want now is to keep only Px so that such a mechanism would be in-built. What about $\exists x Px$? In fact, in mathematical proofs, we produce Pc for an ambiguous name c, and then conclude that $\exists x Px$. It is ambiguous in the sense that we might not be giving a constant to satisfy the relation corresponding to the predicate P. This c might have been produced ambiguously by another existential quantification. For example, consider the consequence $\exists x(Px \wedge Qx) \models \exists x Px$. Let $I = (D, \phi)$ be an interpretation of $\exists x(Px \wedge Qx)$. Then we argue that there is at least one $d \in D$ such that $d \in P'$ and also that $d \in Q'$. Now, since $d \in P'$, we have $I \models \exists x Px$. Here, is d a definite object in D? Not necessarily. It is ambiguous since there might be many such d's in D. The point is that we can use such ambiguous constants in the case of existential quantifiers.

Agreed that $\exists x Px$ can be written as Pc for some (ambiguous) constant c, representing an object in the domain, how to rewrite $\exists x \exists y Qxy$? Is it OK to rewrite it as Qcc? Definitely not since Q' may not hold for the pair (c', c'). But it is safe to get rid of $\exists x \exists y$ by representing the formula as Qbc, where b, c are (possibly) different constants. Similarly, $\exists x \forall y Pxy$ may be rewritten as Pcy, noting that the free variable in Pcy is, in fact, universally quantified. Now, what about $\forall y \exists x Pxy$? It is different from $\exists x \forall y Pxy$ since it seems that x is dependent upon y in order that P may hold for them, and the same x may not work for every y ("each person has a father" does not mean the same as "each person has the same father"). This dependence may be written by using a new function symbol, say, $x = f(y)$. The function symbol f should not have occurred at all in the context. Then the formula $\forall y \exists x Pxy$ may be rewritten as $Pf(x)y$. One more problem: we are planning to omit all the universal quantifiers and interpret all free variables later as if universally quantified. Then, if there were already free variables in the formula, they would also become universally quantified later. For example, $\forall x Pxy$ and $\forall x \forall y Pxy$ will be rewritten as the same formula Pxy. Is this viable? What is the sense of 'viability' here?

In what sense do the meanings of the formulas $\forall y \exists x Pxy$ and of $Pf(y)y$ remain the same? Certainly, they are not equivalent. Can we say that $\models \forall y \exists x Pxy$ iff $\models \forall y Pf(y)y$? The latter formula contains a new function symbol f which may be interpreted in any way we like without affecting the former formula. And for validity, we must see that for every such interpretation of f the truth of the formula must hold, which is, so to speak, expecting too much. If we require this statement to hold for some f, then it may hold. For example, if P is the equality predicate, you require to see the truth of the statement: $\models \forall y \exists x(x \approx y)$ iff $\models \forall y(f(y) \approx y)$. It is all right that $\models \forall y \exists x(x \approx y)$ since $\models (x \approx x)$. But, $\models \forall y(f(y) \approx y)$ is not OK for all f, since in $\mathbb{N}$, with f as the successor function we see that $\forall y(f(y) \approx y)$ is falsified. So, we switch over to another weaker statement for giving sense to the rewriting of existential quantifiers by new function symbols. We try: $\forall y \exists x Pxy$ is satisfiable iff $\forall y Pf(y)y$ is satisfiable.

This trial looks to be successful; for satisfiability, you only require some function f so that $\forall y Pf(y)y$ will hold. Moreover, in this case, as f is a new function, you can choose its interpretation in such a way that $\forall y Pf(y)y$ may hold. All the more, in $\forall y \exists x Pxy$, you had to choose one such x for the corresponding y! We take this sense and continue with getting rid of the quantifiers. Again, we start with a sentence since a formula is valid iff its universal closure is valid. The quantifier elimination process discussed above is referred to as **skolemization** after the logician T. Skolem. Since sentences can be brought to prenex form, we assume that the given formula is already in prenex form. The procedure *QuaEli* given below uses skolemization to eliminate quantifiers from a prenex form formula.

QuaEli finds the first occurrence of $\exists$ in the prefix of the prenex form formula X. Then it finds out the dependence of the variable say, x, used by this $\exists$ by taking those variables which precede this occurrence of $\exists$ (and are universally quantified). It does not take all of them, but only chooses those which are used along with x in some predicate occurring in the formula. These are the variables on which x may depend. To see why is it so, consider the formula $\forall x \forall y \exists z Pxz$. You can see very well that the choice of z does not depend upon y, but only on x though both y and x are universally quantified and precede z. Now, if there are k such variables, say, $x_{i1}, \ldots, x_{ik}$ on which x depends, then *QuaEli* introduces a new function symbol f of arity k and replaces every occurrence of x in the matrix of the formula by the term $f(x_{i1}, \ldots, x_{ik})$. Such new function symbols are referred to as the **indical functions**. The procedure then removes the occurrence $\exists x$ from the prefix. These steps are repeated for all occurrences of $\exists$ in X. After the repeated use of these steps, the formula is reduced to one having no occurrence of a $\exists$. Finally, all the universal quantifiers are simply ignored to output the quantifier free formula $\overline{X}$ corresponding to X. Note that all variables in $\overline{X}$ are now universally quantified. The quantifier-free form $\overline{X}$ of the formula X is referred to as the **Skolem form** of the formula X.

```
PROCEDURE : QuaEli
  Input: A prenex form formula $X = Q_1y_1 \cdots Q_ny_nA$
  Output: A quantifier free formula $\overline{X}$
   Let the free variables of $X$ be $y_{n+1}, \ldots, y_m$
   while $1 \le j \le n$
      do
      if $Q_i = \exists$ then let $S_i = \{y_1, \ldots, y_{i-1}, y_{n+1}, \ldots, y_m\}$
        while $1 \le j \le i-1$ or $n+1 \le j \le m$
        do
        if no atomic subformula of $X$ contains both $y_i$ and $y_j$ then
            $S_i := S_i - \{y_j\}$
        od
         let $S_i := \{y_{i1}, \ldots, y_{ik}\}$
         let $f_i$ be a new function symbol of arity $k$
         $Y := Q_1y_1 \cdots Q_{i-1}y_{i-1}Q_{i+1}y_{i+1} \cdots Q_ny_n(A[y_i/f_i(y_{i1}, \ldots, y_{ik})])$
      od
   $\overline{X}$ := matrix of $Y$
```

***Exercise* 2.16** Write *QuaEli* as we wrote previous procedures in steps.

***EXAMPLE* 2.16** Find the Skolem form of the prenex form formula

$$X = \forall y \exists z \exists v \forall u((Qx \to Su) \wedge (Pxy \to \neg Qv \vee \neg Rvz))$$

Solution First, *QuaEli* concentrates on how $\exists z$ can be eliminated. The variables preceding it, which are universally quantified, along with the free variables, are in the set $S_z = \{x, y\}$. However, no atomic subformula of X has both z and x, neither do both z and y occur in any atomic subformula. Hence, $S_z := S_z - \{x\}; S_z := S_z - \{y\}$ gives $S_z = \emptyset$. That is, z does not depend upon any other variable; so, z is to be replaced by a new 0-ary function symbol, a constant. Writing c for this constant (the procedure uses the symbol f_1), we have the updated formula:

$$A = \forall y \exists v \forall u((Qx \to Su) \wedge (Pxy \to \neg Qv \vee \neg Rvc))$$

The above procedure is repeated for $\exists v$. You see that v may depend upon the variable y preceding it and the free variable x; but no atomic subformula contains both v and y, nor both v and x. Hence, again, a new constant, say, b, is introduced to replace v. The updated formula so obtained is

$$\forall y \forall u((Qx \to Su) \wedge (Pxy \to \neg Qb \vee \neg Rbc))$$

Finally, drop all $\forall$'s to obtain the (quantifier free) Skolem form formula

$$\overline{X} = (Qx \to Su) \wedge (Pxy \to \neg Qb \vee \neg Rbc)$$

What we now expect is that X is satisfiable iff $\overline{X}$ is satisfiable; show it.

***EXAMPLE* 2.17** Use *QuaEli* to get a quantifier free formula for $A = \exists x \forall y \forall u \forall v \exists w((Px \to Py \wedge Qx \wedge \neg Qy) \wedge (Pu \to Qz) \wedge (Qz \to Pu))$.

Solution For x, w, we choose the constants a, b respectively. Then,

$$\overline{A} = (Pa \to Py \wedge Qa \wedge \neg Qy) \wedge (Pu \to Qz) \wedge (Qz \to Pu)$$

Note that b does not occur in $\overline{A}$. Is $\overline{A}$ satisfiable iff A is satisfiable?

***Exercise* 2.17** What does *QuaEli* give as output if inputs are

(a) $\exists z \exists v \forall u \forall x \forall y((Qx \to Su) \wedge (Pxy \to \neg Qv \vee \neg Rvz))$

(b) $\forall x \forall y \forall u \exists v \exists z((Qx \to Su) \wedge (Pxy \to \neg Qv \vee \neg Rvz))$?

The quantifier free formula $\overline{A}$ in Example 2.16 can be converted to one of the normal forms by using the PL-mechanism. For example, a cnf for $\overline{A}$ in Example 2.16 is

$$\begin{aligned}\overline{A}_1 &= (Qx \to Su) \wedge (Pxy \to \neg Qb \vee \neg Rbc)\\ &\equiv (\neg Qx \vee Su) \wedge (\neg Pxy \vee \neg Qb \vee \neg Rbc)\end{aligned}$$

And the dnf of $\overline{A}$ in Example 2.17 is

$$\begin{aligned}\overline{A}_2 &= (Pa \to Py \wedge Qa \wedge \neg Qy) \wedge (Pu \to Qz) \wedge (Qz \to Pu)\\ &\equiv (\neg Pa \wedge \neg Pu \wedge \neg Qz) \vee (\neg Pa \wedge Qz \wedge Pu)\\ &\quad \vee (Py \wedge Qa \wedge \neg Qy \wedge \neg Pu \wedge \neg Qz) \vee (Py \wedge Qa \wedge \neg Qy \wedge Qz \wedge Pu)\end{aligned}$$

Such forms are called *Skolem normal forms* or *Skolem standard forms*.

Definition 2.6 A formula is in **scnf** or Skolem conjunctive normal form if it is quantifier free and is in cnf. A formula is in **sdnf** or Skolem disjunctive normal form if it is quantifier free and is in dnf.

As expected, we have the following result.

Theorem 2.17 (Standard Form) *For any formula X, there exist an scnf formula $\overline{X}$ and an sdnf formula $\hat{X}$ such that X is satisfiable iff $\overline{X}$ is satisfiable iff $\hat{X}$ is satisfiable.*

Proof Let A be any rectified formula in prenex form and A' be obtained from A by skolemization. Let $I = (D, \phi)$ be any interpretation and ℓ be any valuation under I. We show that

(a) if $I_\ell \models A'$ then $I_\ell \models A$, and

(b) if $I_\ell \models A$, then ϕ can be extended to include the indical functions in its domain in such a way that the new interpretation J with a new valuation m will have $I_m \models A'$. (Why is it enough to show these?)

Both (a) and (b) are proved by induction on $n(A)$, the number of occurrences of $\exists$ in A. In the basis case, $n(A) = 0$ gives $A = A'$; thus, both (a) and (b) hold. Assume that (induction hypothesis) both (a) and (b) hold for any formula A whenever $n(A) < n$.

Let A be a formula, with $n(A) = n$ and $y_1, \ldots, y_j$ as the free variables of A. As A is in prenex form, it looks like $A = \forall x_1 \ldots \forall x_k \exists x B$, where $n(B) = n - 1$. (Look at the first occurrence of $\exists$ in A; the formula to the right of it is B.) Then, $A' = \forall x_1 \ldots \forall x_k (B[x/t])'$, where $t = f(x_1, \ldots, x_k, y_1, \ldots, y_j)$ for an indical function f.

Assume, without loss of generality (Why?), that each of these variables $x_1, \ldots, x_k$ occurs along with x in some predicate. We write $(B[x/t])'$ for the formula obtained from $B[x/t]$ by skolemization. Let $I = (D, \phi)$ be an interpretation and ℓ be a valuation under I.

For (a), let $I_\ell \models A'$. Then for all $d_1, \ldots, d_k \in D$, $I_m \models (B[x/t])'$, where the valuation $m = \ell[x_1 \rightarrowtail d_1] \cdots [x_k \rightarrowtail d_k]$. Write $m(t) = d$. B' has been obtained from B by skolemization. Since $(B[x/t])' = B'[x/t]$, for all $d_1, \ldots, d_k \in D, I_{m[x \rightarrowtail d]} \models B'$. By induction hypothesis, for all $d_1, \ldots, d_k \in D, I_{m[x \rightarrowtail d]} \models B$. Then, $I_\ell \models \forall x_1 \ldots \forall x_k \exists x_k B$, i.e., $I_\ell \models A$.

For (b), suppose $I_\ell \models \forall x_1 \ldots \forall x_k \exists x B$. Then, for every $d_1, \ldots, d_k \in D$, $I_{m[x \rightarrowtail d]} \models B$. Since f is a new (indical) function symbol, I does not interpret it. Define $\phi(f)$ to be the function from D^{k+j} to D so that $\phi(f)(m(x_1, \ldots, m(x_k), m(y_1), \ldots, m(y_j)) = d$ i.e., $m(t) = d$. Then, for every $d_1, \ldots, d_k \in D, I_{m[x \rightarrowtail d]} \models B[x/t]$. As $m(x) = d = m(t)$, we have $I_m \models B[x/t]$. By induction hypothesis, I can be extended to some J such that $J_m \models (B[x/t])'$ whenever $I_m \models B[x/t]$. Thus, for every $d_1, \ldots, d_k \in D$, $J_m \models (B[x/t])'$. That is, $J \models \forall x_1 \ldots \forall x_k (B[x/t])'$, or that, $J \models A'$. ∎

Note that we have assumed A to be in prenex form and also it was rectified. Of course, our procedure *PrenForm* always rectifies a formula. If it is not rectified, then the above proof will not work. Why? See where exactly it goes wrong.

Next, you must also have observed that skolemization treats both free and universally quantified variables the same way. This says that we always keep in mind the metastatement, "$\models X$ iff $\models \forall^* X$". Thus, you can simply start from a sentence by taking the universal closure of a given formula. Now, if you start with a sentence, skolemization can be done in a simpler way. Modify the procedure *QuaEli* to do it.

You have seen that skolemization helps in transforming a formula into a quantifier-free form keeping satisfiability invariant. Can you get a similar formula which is validity invariant? The question is, can you transform a sentence into a quantifier-free form so that the sentence is valid iff its quantifier free form is valid? The hint is that satisfiability and validity are dual concepts just as $\wedge, \vee$ and as $\forall, \exists$ are. Let us see an example. If $\forall x \exists y X$ is a sentence, it will be valid iff $\neg \forall x \exists y X$ is unsatisfiable iff $\exists x \forall y \neg X$ is unsatisfiable iff X' is unsatisfiable, where X' is the Skolem standard form of $\exists x \forall y \neg X$. Now, while getting the Skolem standard form of $\exists x \forall y \neg X$, you remove the first $\exists x$. Look at the original sentence. This corresponds to dropping the first $\forall x$ from the original sentence, by introducing an indical function. And finally, all $\exists$'s would be dropped without any replacements.

This suggests the following procedure. Let A be a given rectified formula in prenex form. Take its universal closure, call it B. Now remove all $\forall$'s from B by using *QuaEli*–$\forall$, where this is a modified form of *QuaEli*. The modification treats $\exists$ as $\forall$ and $\forall$ as $\exists$ in *QuaEli*. Then, you get a quantifier free form of A, called the **functional form** of A. The corresponding cnf and dnf forms are named as **fcnf** or the functional cnf, and **fdnf** or the functional dnf, respectively.

***EXAMPLE* 2.18** Find an fcnf for the formula

$$A = \exists z \exists v \forall u((Qx \to Su) \wedge (Pxy \to \neg Qv \vee \neg Rvz))$$

Solution The free variables in A are x and y. First, we take the universal closure of the formula to get

$$\forall x \forall y \exists z \exists v \forall u((Qx \to Su) \wedge (Pxy \to \neg Qv \vee \neg Rvz))$$

It is a rectified formula in prenex form. We want to eliminate the universal quantifiers by introducing new indical functions. Since no $\exists$ precedes $\forall x \forall y$, x, y are replaced by constants, say, a, b, respectively. Note that this amounts to replacing all the free variables first by some new (different) constants. We thus get the formula

$$\exists z \exists v \forall u((Qa \to Su) \wedge (Pab \to \neg Qv \vee \neg Rvz))$$

$\exists z$ and $\exists v$ precede $\forall u$, but there is no atomic subformula containing both z, u or both v, u. Hence, we use a new constant, say, c, to eliminate $\forall u$. This gives us the formula

$$\exists z \exists v((Qa \to Sc) \wedge (Pab \to \neg Qv \vee \neg Rvz))$$

Then we simply drop the existential quantifiers to get the functional form

$$(Qa \to Sc) \wedge (Pab \to \neg Qv \vee \neg Rvz)$$

The fcnf is obtained by replacing $\to$ with $\neg$ and $\vee$. We have the fcnf as

$$A' = (\neg Qa \vee Sc) \wedge (Pab \to \neg Qv \vee \neg Rvz))$$

The theorem 2.18 following the example below claims that $\models A$ iff $\models A'$. Can you show this for the formula A in Example 2.18 directly from the semantics?

***EXAMPLE* 2.19** Construct an fdnf for the formula

$$\exists x \forall y \forall u \forall v \exists w \forall z((Rux \to Py \wedge Qx \wedge \neg Qy) \wedge (Pu \to Qz) \wedge (Qz \to Pu))$$

Solution This is already rectified and in prenex form having no free variables. We start removing universal quantifiers. The first $\forall$ from the left is $\forall y$, which occurs after $\exists x$. But both x, y do not occur in any predicate. Hence y is replaced by a constant, say, a. Next, $\forall u$ occurs after $\exists x$, and both u, x occur in Rux. Thus, a new indical function, say, f of arity 1, is introduced and $f(x)$ replaces u. Similarly, v is replaced by a constant, say, b, and z is replaced by a constant c. Thus, we have the functional form

$$(Rf(x)x \to Pa \wedge Qx \wedge \neg Qa) \wedge (Pf(x) \to Qc) \wedge (Qc \to Pf(x))$$

Now, you can complete the steps to get an fdnf.

Analogous to *QuaEli*, you can write the procedure *QuaEli*–$\forall$ for eliminating the quantifier $\forall$ and prove the following theorem.

Theorem 2.18 (Functional Form) *For any formula X, there exist an fcnf formula $\tilde{X}$ and an sdnf formula X' such that X is valid iff $\tilde{X}$ is valid iff X' is valid.*

Once you get a prenex form of a formula, you can skolemize and then obtain an scnf and also an sdnf. In the scnf, the variables are all universally quantified. Moreover, all the universal quantifiers must be in the beginning of the formula. But then, $\forall$ distributes over $\wedge$, and thus, you may visualize an scnf clause by clause, and think that in each of the clauses, the variables are universally quantified. In an sdnf, however, this will not happen, since $\forall$ does not distribute over $\vee$. That is, the conjunctive clauses in an sdnf do share their variables, and you cannot think of free variables in the individual clauses as universally quantified; they are universally quantified for the whole sdnf. This is why scnf goes well with skolemization on the $\exists$'s and fdnf goes well with the skolemization applied on $\forall$'s.

You must remember that in the Skolem normal form, all the free variables are universally quantified, whereas in the functional form, all the free variables are existentially quantified. We will refer to both the Skolem form and the functional form as the **sentential forms** of a formula. Theorems 2.17 and 2.18 are summarized in the following statement.

Theorem 2.19 (Sentential Form) *Let X be any formula. Then there exist sentences $X_S = \forall y_1 \ldots \forall y_m Y$ and $X_F = \exists z_1 \ldots \exists z_n Z$, for some quantifier-free formulas Y and Z such that*

(a) X is satisfiable iff X_S is satisfiable.

(b) X is valid iff X_F is valid.

The validity of X_F can now be tested by searching for closed terms $t_1, \ldots, t_n$ such that $\overline{Z} = Z[z_1/t_1] \cdots [z_n/t_n]$ holds. Similarly, by duality, satisfiability of X_S can be tested. But where to search for these terms which may make $\overline{Z}$ hold?

2.8 Herbrand Interpretation

We reiterate the last question. You have a quantifier free formula corresponding to any given formula for determining satisfiability. Since satisfiability means that you have a model for the formula, and a model may start with a nonempty set, the problem is: how to choose such a set from among seemingly infinite possibilities? Is there any way to construct a sort of canonical model taking help from the formula itself?

For simplicity, suppose the quantifier free formula is Pab. It is quite natural to start with a set having two elements, say, $\{a', b'\}$ and take the relation $P' = \{(a', b')\}$. Let us simplify a bit. Instead of using primes on the syntactic entities, what happens if we take our domain as $\{a, b\}$ and the relation as P itself, asserting that P holds for the ordered pair (a, b)? That is, why don't we look at the syntactic entities as the concrete ones? It would obviously simplify notation.

It looks that it is viable, but we must remember that we are, in fact, dropping the primes, as a convention. Well, what happens if the formula has a free variable (which is now universally quantified)? For example, suppose that the formula is Pxa. Then we can start with a singleton, say, $\{a\}$ and assert that Paa holds. What happens for $Pf(a)a$? The same way again: start with a singleton $\{a\}$. How to tackle the function symbol, now, a function f? It must be a function from $\{a\}$ to $\{a\}$; so define $f(a) = a$. Then, of course, you have a model of $Pf(a)a$.

However, syntactically, $f(a) \neq a$ and, semantically, f may not represent an identity function. As a syntactic entity, f is just a function symbol of arity 1, and it potentially generates a countable set $D = \{a, f(a), f(f(a)), \ldots\}$. On this domain D, we may regard f to represent a function

$$f' : \{a, f(a), f(f(a)), \ldots\} \to \{a, f(a), f(f(a)), \ldots\}$$

defined by $f'(a) = f(a), f'(f(a)) = f(f(a)), \ldots$. Following this, we rather write f' as f itself. Fine. What about the equality predicate? We cannot, in fact, regard $f(a)$ as a if we are given with a formula $a \approx f(a)$. For a predicate P, if we have the formula $Paf(a)$ to be satisfied, we agree to assert $Paf(a)$, but if P is the predicate $\approx$, then we must assert $a \approx f(a)$, which would then correspond to asserting $a = f(a)$. However, as syntactic entities, a and $f(a)$ are distinct; so we cannot possibly assert $a = f(a)$!

Let us see why this problem does not show up when we do semantics. Suppose that we start with a domain D. We have $\phi(a) = d$ for some $d \in D$. Then $\phi(f(a))$ can also be the same element d. This creates no ambiguity. But in the syntactic domain $\{a, f(a), f(f(a)), \ldots\}$, $f(a)$ is never equal to a. Then, how to bring this notion of equality (or identity) of a with $f(a)$ to our syntactic domain? Since this is a problem of identification, naturally, equivalence relations play a role. If we can define some equivalence relation on this syntactic domain, where a and $f(a)$ may be equivalent, then the equivalence classes of that relation would form the domain instead of the set $\{a, f(a), f(f(a)), \ldots\}$. Essentially, our domain will consist of representatives of each of the equivalence classes. Note that the equivalence relation need not be (must not be, in general) the equality relation '='. However, we have to capture as many properties of '=' as possible by this equivalence relation.

The properties of '$=$' that are useful to us are that it is an equivalence relation, and that we can substitute equals for equals keeping meanings intact. The substitutivity property tells (in FL) that if $s = t$, then $f(\ldots, s, \ldots) = f(\ldots, t, \ldots)$ for any function (symbol) f; besides, for predicates P, we must have "if $P(\ldots, s, \ldots)$ holds, then $P(\ldots, t, \ldots)$ holds". We can now have a relation, say, E defined on the set $\{a, f(a), f(f(a)), \ldots\}$ satisfying these properties, and then interpret the predicate $\approx$ as this E. A formal description incorporating these ideas follows.

Let X be a formula in Skolem form; so, all free variables are assumed to be universally quantified. Let D_0 be the set of all constants occurring in X. If $D_0 = \emptyset$, then $D_1 = \{\eta\}$; else, $D_1 = D_0$. Then the domain, D_X for the formula X, called the **Herbrand universe**, named after the logician J. Herbrand, is defined recursively:

1. $D_X \supseteq D_1$.
2. If f is an n-ary function symbol occurring in X, and $t_1, t_2, \ldots, t_n \in D_X$, then $f(t_1, t_2, \ldots, t_n) \in D_X$.
3. Each element of D_X satisfies 1 or 2 or both.

We will write D_X as D, whenever the formula X is clear in a context. Given a formula X, you are supposed to generate the set D_X step by step starting from D_0 as the set of all constants occurring in X as shown below:

$$D_0 = \{c : c \text{ is a constant occurring in } X\}$$

$$D_1 = D_0 \text{ if } D_0 \neq \emptyset, \text{ else, } D_1 = \{\eta\}$$

$$D_2 = D_1 \cup \{f(t_1, \ldots, t_n) : t_1, \ldots, t_n \in D_1 \text{ and } f \text{ is an } n\text{-ary function symbol occurring in } X\}$$

In general,

$$D_{i+1} = D_i \cup \{f(t_1, \ldots, t_n) : t_1, \ldots, t_n \in D_i \text{ and } f \text{ is an } n\text{-ary function symbol occurring in } X\}$$

Finally,

$$D = D_X = D_0 \cup D_1 \cup D_2 \cup \cdots = \cup_{i \in \mathbb{N}} D_i$$

For example, let $X = \neg Pxf(x) \wedge Pya$, where a is a constant. Then

$$D_0 = \{a\}$$

$$D_1 = D_0 = \{a\}$$

$$D_2 = \{a, f(a)\}, \ldots$$

$$D = D_X = \{a, f(a), f(f(a)), \ldots\}$$

The elements of the Herbrand universe are also called the **ground terms**. Note that ground terms are simply the closed terms obtained from the function symbols occurring in a formula or from the special symbol η if no constant occurs in it. We want to use the Herbrand universe as a domain of an interpretation. For this purpose, we require a function ϕ which would be defined for each function symbol and each predicate occurring in X. Moreover, ϕ must assign function symbols to functions over D and predicates to relations over D. This is defined as follows:

(a) If f is an n-ary function symbol occurring in X, then $\phi(f) = f$, the latter f is taken as a function from D^n to D defined by: the functional value of the n-tuple of terms $t_1, t_2, \ldots, t_n$ under f is $f(t_1, t_2 \ldots, t_n)$ for objects $t_1, t_2, \ldots, t_n \in D$.

(b) If $P \not\approx$ is an m-ary predicate, then $\phi(P) = P$, where the latter P is an m-ary relation defined on D. (It is any m-ary relation; we are not fixing it.)

(c) $\phi(\approx) = E$, where E is a binary relation defined on D satisfying:

(i) E is an equivalence relation.

(ii) If $(s, t) \in E$, then for every $t_1, \ldots, t_{i-1}, t_{i+1}, \ldots t_n \in D$ and for every n-ary function (symbol) f occurring in X,

$$(f(t_1, \ldots, t_{i-1}, s, t_{i+1}, \ldots t_n), f(t_1, \ldots, t_{i-1}, t, t_{i+1}, \ldots t_n)) \in E.$$

(iii) If $(s, t) \in E$, then for every $t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_n \in D$ and for every n-ary predicate P occurring in X, $P(t_1, \ldots, t_{i-1}, s, t_{i+1}, \ldots t_n)$ holds iff $P(t_1, \ldots, t_{i-1}, t, t_{i+1}, \ldots t_n)$ holds.

The **Herbrand interpretation** of X is the pair $H_I = (D, \phi)$. By assigning truth values 0 or 1 to the atomic formulas $P(t_1, \ldots, t_n)$, it can be determined whether a Herbrand interpretation is a model of the formula or not.

For example, let $X = (Qyx \to Px) \wedge (Py \to Rc)$. The Herbrand universe is the singleton $D = \{c\}$. The Herbrand interpretation asks for satisfying the formula $(Qcc \to Pc) \wedge (Pc \to Rc)$, obtained from X by substituting the variables x, y by c, the only element in D. By assigning truth values to the atomic formulas Qcc, Pc, Rc we may see whether the Herbrand interpretation is a model of X or not.

Define an assignment i by: $i(Qcc) = 0$, $i(Pc) = 1 = i(R(c))$. Then, we see that $i((Qcc \to Pc) \wedge (Pc \to Rc)) = 1$. Hence we have a Herbrand model of X. On the other hand, with $j(Qcc) = 1 = j(Pc)$, $j(Rc) = 0$, you see that $j((Qcc \to Pc) \wedge (Pc \to Rc)) = 0$. That is, the same Herbrand interpretation with j is not a model of X. The **Herbrand model** with i as the assignment of truth values can also be written as (see the third way of propositional semantics): $\{\neg Qcc, Pc, Rc\}$ and the Herbrand interpretation with j as the assignment can be written as $\{Qcc, Pc, \neg Rc\}$. In such a formalism of writing out the Herbrand interpretations, the atomic formulas Qcc, Pc, Rc are called **ground atomic formulas**, and the literals $Qcc, \neg Qcc, Pc, \neg Pcc, Rc, \neg Rcc$ are termed as **ground literals**. A Herbrand interpretation is then a set of ground literals such that both Y and

$\neg Y$ are not its members. Corresponding to each ground atomic formula Z of X, exactly one of Z or $\neg Z$ is an element of a Herbrand interpretation.

Thus, our definition of a Herbrand interpretation should also have included such an assignment i or j. But we have not done so. Why? The reason is, whatever be the assignment of truth values to the atomic formulas here, the formula X is satisfied in the Herbrand universe provided the corresponding formula $(Qcc \to Pc) \wedge (Pc \to Rc)$ holds. The latter can be checked propositionally. Our approach is not to finish the satisfiability check at a time, but to transfer the responsibility to propositional satisfiability, and then go on checking for satisfiability propositionally. So, instead of a Herbrand model, we define a Herbrand expansion, which comes from instantiating the given formula (in Skolem form) by taking the elements of the Herbrand universe. The **Herbrand expansion** H_E is the set

$$H_E = \{X[x_1/d_1]\cdots[x_m/d_m] \;:\; x_1,\ldots,x_m \text{ are all the free variables of } X \text{ and } d_1,\ldots,d_m \in D\}$$

The formulas in the Herbrand expansion of a formula X are also called the **ground instances** of X. Since ϕ is syntactic, we would like to go for computing the Herbrand expansion directly. Note that the Herbrand expansion for the formula $X = (Qyx \to Px) \wedge (Py \to Rc)$ is simply the singleton $\{(Qcc \to Pc) \wedge (Pc \to Rc)\}$. In general, existence of a Herbrand model would simply be equivalent to the satisfiability of the Herbrand expansion.

EXAMPLE **2.20** Let $X = \neg Pxf(x) \wedge Pya$, where a is a constant. Then the Herbrand universe is $D = \{a, f(a), f(f(a)), \ldots\}$. The Herbrand expansion is

$$H_E = \{\neg Paf(a) \wedge Paa, \neg Paf(a) \wedge Pf(a)a, \neg Paf(a) \wedge Pf(f(a))a, \ldots,\\ \neg Pf(a)f(f(a)) \wedge Paa, \neg Pf(f(a))f(f(f(a))) \wedge \neg Pf(a)a, \ldots\}$$

The formulas in H_E, the ground instances of X, are obtained by the substitutions on X by choosing all possible ground terms of X.

Now, $H_I \models X$ iff H_E is propositionally satisfiable, which holds since we can have a truth assignment which would be a (PL-) model of H_E. For example, i defined by $i(Pst) = 0$ if $s, t \in D$ and $t = f(s)$; otherwise, $i(Pst) = 1$ is such a model. The Herbrand model of X written as a set of literals is $\{Pst : s, t \in D \text{ and } t \neq f(s)\} \cup \{\neg Psf(s) : s \in D\}$.

EXAMPLE **2.21** Let $X = Pxf(x) \wedge \neg Pay$. Then, $D = \{a, f(a), \ldots\}$ as in Example 2.20, but the Herbrand expansion is

$$H_E = \{Paf(a) \wedge \neg Paa, Paf(a) \wedge \neg Paf(a), \ldots,\\ Pf(a)f(f(a)) \wedge \neg Paa, Pf(a)f(f(a)) \wedge \neg Paf(a), \ldots\}$$

This is not propositionally satisfiable since the clause $Paf(a) \wedge \neg Paf(a)$, a member here, is unsatisfiable.

***EXAMPLE* 2.22** Let $X = Pxy \wedge (Pxy \to Qxy) \wedge \neg Qxz \wedge (y \approx z)$. Then, $D = \{\eta\}$ and $H_E = \{P\eta\eta \wedge (P\eta\eta \to Q\eta\eta) \wedge \neg Q\eta\eta \wedge E\eta\eta\}$. H_E is propositionally unsatisfiable. Is X also unsatisfiable?

***EXAMPLE* 2.23** Let $X = Pxf(x) \wedge \neg Pxx \wedge (x \approx f(x))$. As X has no constants, $D = \{\eta, f(\eta), f(f(\eta)), \ldots\}$. The Herbrand expansion is

$$H_E = \{P\eta f(\eta) \wedge \neg P\eta\eta \wedge E\eta f(\eta),$$
$$Pf(\eta)f(f(\eta)) \wedge \neg Pf(\eta)f(\eta) \wedge Ef(\eta)f(f(\eta)), \ldots\}$$

Since $E\eta f(\eta)$, $P\eta f(\eta)$ holds iff $Pf(\eta)f(\eta)$ holds. If $i(Pf(\eta)f(\eta)) = 1$, then $i(Pf(\eta)f(\eta)) = 1$ and $i(\neg Pf(\eta)f(\eta)) = 0$. This forces $i \not\models H_E$. Therefore, H_E is unsatisfiable.

In Examples 2.20 and 2.21, satisfiability of H_E is propositional, i.e., its satisfiability is determined by taking each atomic formula in H_E as a sentence of PL and then assigning them a truth value 0 or 1. It is still applicable in Example 2.22, though the equality predicate $\approx$ is involved. However, in Example 2.23, equality predicate has some nontrivial effect. Satisfiability of H_E is determined in this case by using a property of the relation E. So, how do we assert that satisfiability is still propositional here? This should be remedied, as we already have remarked, by taking the equivalence classes imposed on the domain, and in turn, on H_E.

Theorem 2.20 (Syntactic Interpretation) *Let X be a formula in Skolem standard form. Let H_I and H_E be the Herbrand interpretation and the Herbrand expansions of X, respectively. Then X is satisfiable iff $H_I \models X$ iff H_E is propositionally satisfiable.*

Proof It is clear from the construction of H_I and H_E that $H_I \models X$ iff H_E is propositionally satisfiable. Now, suppose that $H_I \models X$. Then X has a model (this model). That is, X is satisfiable. Conversely, suppose that X is satisfiable. Let $I = (A, \psi)$ be a model of X. Let $H_I = (D, \phi)$ be the Herbrand interpretation of X. We want to show that $H_I \models X$.

Define a function $\mu : D \to A$ by

(a) If $\eta \in D$, then $\mu(\eta)$ is some fixed element of A.

(b) For each constant c occurring in X, $\mu(c) = \psi(c)$.

(c) For each n-ary function symbol f occurring in X, and for any $t_1, \ldots, t_n \in D$, $\mu(f(t_1, \ldots, t_n)) = \psi(f)(\mu(t_1), \ldots, \mu(t_n))$.

(d) For any m-ary predicate P occurring in X, and for any $t_1, \ldots, t_m \in D$, $P(t_1, \ldots, t_m)$ holds iff $(\mu(t_1), \ldots, \mu(t_m)) \in \psi(P)$.

Note that (A, ψ) is an interpretation of X, thus, $A \neq \emptyset$, $\psi(c) \in A$, and $\psi(f) : A^n \to A$. Hence the function μ is well defined. This function μ, called an *embedding*, helps us relate the model I with the Herbrand interpretation H_I. We show that $H_I \models X$ by induction on the number of free variables of X.

In the basis step, if X has no free variables, then since $I \models X$, due to the construction of μ, $H_I \models X$.

Lay out the induction hypothesis that whenever the number of free variables of X are less than k, we have $H_I \models X$. Let X be a formula (in Skolem standard form) having, k number of free variables. Since $I \models X$, for each $t \in D$ ($\mu(t) \in A$), we have $I \models X[x/\mu(t)]$. This also implies that $H_I \models X[x/t]$ (Why?) for each $t \in D$. Hence $H_I \models \forall x X$, i.e., $H_I \models X$. ▮

Since Herbrand expansions are countable, we have the following remarkable result. If X is a satisfiable formula, then its Herbrand expansion is countable. Moreover, the Herbrand interpretation with the Herbrand domain is a model of the formula. Hence, it has a countable model, namely, the Herbrand interpretation. This observation is easily generalized to a countable set of formulas since the Herbrand expansion of such a set is also countable. However, as a caution, you must choose different indical functions for (even same variables in) different formulas in this countable set, while skolemization is applied. Can it be done? Moreover, why is it necessary to choose different indical functions?

Theorem 2.21 (Skolem-Löwenheim) *Any countable set of formulas has a countable model.*

Having the dual considerations, you can have, similarly, a Herbrand universe for the functional form of a formula. You can then show that a formula in functional form is valid iff it has a Herbrand model. Regarding the Herbrand (validity) expansion, you could say that some finite disjunction of all the formulas in the expansion is valid.

Note that the Herbrand expansion is simply a set of formulas which are obtained from X by replacing the free variables with terms or elements of the Herbrand universe. If $x_1, \ldots, x_n$ are all the free variables of X, we may write X as $X(x_1, \ldots, x_n)$. Further, writing $\bar{x}$ for the n-tuple $(x_1, \ldots, x_n)$ and $\bar{t}$ for $(t_1, \ldots, t_n)$, we will abbreviate the series of substitutions $[x_1/t_1][x_2/t_2] \cdots [x_n/t_n]$ as $[\bar{x}/\bar{t}]$. Then, the Herbrand expansion can be written schematically as

$$H_E = \{X[\bar{x}/\bar{t}] : \bar{t} \in D^n\}$$

If X is in functional form, then after constructing its corresponding D, you find that X is valid iff the set $H_E = \{X[\bar{x}/\bar{t}] : \bar{t} \in D^n\}$ is valid. The discussion is summarized as in the following theorem.

Theorem 2.22 (Herbrand) *For any formula X, there are formulas Y and Z having free variables $\bar{y}$ and $\bar{z}$, and (possibly infinite) sets of terms $\overline{S}$ and $\overline{T}$ such that*

(a) X is satisfiable iff $\{Y[\bar{y}/\bar{s}] : \bar{s} \in \overline{S}\}$ is satisfiable.

(b) X is valid iff $\{Z[\bar{z}/\bar{t}] : \bar{t} \in \overline{T}\}$ is valid.

Suppose that you have been given a valid formula. To show that it is indeed valid, you can always have a calculation which uses the quantifier laws first, and then only propositional laws to get $\top$. That is, no quantifier law is required to be used before any propositional laws. This is how you can interpret the Herbrand's theorem. Can you see it that way? Another question of importance: can the sets $\overline{S}$ and $\overline{T}$ be chosen to be finite sets?

As in PL, a **first order theory** starts with a first order language and then goes for what can be deduced from assuming certain formulas. For example, the various theories found in mathematics are first order theories, and some, which do not look like first order theories, can be formulated as first order theories. In the theory of groups, we have four axioms which are just some formulas assumed to hold in the theory. Then whatever theorem you prove in the theory is simply a conclusion of a consequence whose premises are these four formulas. Similarly, the theory of real numbers $\mathbb{R}$ can be thought of as a first order theory; it is simply a complete ordered archimedean field. It is of foundational importance to know how set theory can be presented as a first order theory. Moreover, the Skolem-Löwenheim theorem would imply that every first order theory, which is known to have a model (which contains no contradictions), can have a countable model, namely, its Herbrand model. What about Cantor's theorem that the set $\mathbb{R}$ is uncountable? This result says that whatever set that satisfies all the axioms of $\mathbb{R}$ is uncountable. Now in light of Skolem-Löwenheim, there is a set (the Herbrand model) which satisfies all the axioms of $\mathbb{R}$, and yet, is countable. This phenomenon is called the *Skolem paradox*. A possible explanation of this is that the mapping that makes the model countable is not (cannot be constructed) inside the model, which is essentially Cantor's theorem.

SUMMARY

In this chapter, you have learnt that it is not enough to consider propositions as units. For expressiveness, we require a logic which would provide us with means to deal with the internal structure of propositions. First order logic is such a logic which takes its simple units as the constants, variables, function symbols, predicates, and quantifiers. The function symbols along with constants and variables express the definite descriptions by way of terms, and the predicates express the relations between terms. The two quantifiers (for all, there exists) quantify over the variables so that propositions or sentences could be constructed. A first order language is defined by taking some of the nonlogical symbols from among the constants, function symbols, and predicates. Such a language is defined recursively with the help of punctuation marks so that unique parsing would hold.

Meanings to the syntactic entities are supplied by taking a nonempty set, called the domain (or universe of discourse) and then by assigning constants to objects in the domain, function symbols to concrete functions on the domain, and predicates to relations over the domain. The variables are assigned to elements in the domain by valuations or assignment functions. The quantifier 'for all' and 'there exists' are given meaning through these valuations. Finally, the semantics enables you to categorize formulas (and sentences) as valid, invalid, satisfiable or unsatisfiable by looking at an interpretation which may or may not be a model of the formula. You have seen three types of semantics, which differ in giving meanings to the quantifiers; but their actions turned out to be the same as far as models are concerned. You have also seen how to tackle consequences in a similar way. The semantics further enforces certain laws analogous to PL and you have learnt how to use these laws in calculations. The approach of calculations help categorize further formulas from known formulas. They also provide a means in showing the validity of consequences. Further, the metaresults such as Monotonicity, *reductio ad absurdum*, and Deduction hold in this logic. The deduction theorem requires care since the two metasentences, '$X \models Y$' and 'if $\models X$, then $\models Y$' do not convey the same meaning. Accordingly, we have introduced a new symbol $\Rightarrow$ for using in calculations, where $X \Rightarrow Y$ stands for the weaker metasentence: if $\models X$, then $\models Y$.

You have also learnt how to define normal forms using skolemization. In this connection, you have seen two types of conversion of formulas; one which preserves satisfiability via universal closure, and the other that preserves validity via existential closure. You have learnt how to check satisfiability of formulas by interpreting it syntactically. This technique of Herbrand interpretations enables you to convert first order satisfiability to propositional satisfiability via Herbrand expansions. This has brought forth a remarkable result by T. Skolem and L. Löwenheim, which says that every satisfiable first order theory has a countable model. This result is sometimes referred to as the Skolem-Löwenheim downward theorem. For the Skolem-Löwenheim upward theorem, see Problems 49 and 50 after Summary. To pursue the topics further, the following bibliographic remarks will be helpful.

After Aristotle formalized a part of logic in the form of syllogisms, there was a gap of almost two centuries for another leap in this direction. It was G. Frege who, in 1879, took the leap [26] and later others followed the work to give us the first order logic as we know it today. For an account of such contributions towards the development of the subject, see for example, [39]. For discussions on theoretical connection of FL to topics such as type theory, logic programming, algebraic specifications and term rewriting, see the works [28, 75, 79]. The calculational approach to proofs has been elaborated in [35]. For automatic theorem proving approaches refer [6, 22, 48]. For problem solving using FL, [43] would provide a good reading. If you are interested in metamathematical aspects of FL which asks and tries to answer questions such as representation of theories as first order theories, completeness of systems, decidability, proof of (in)consistency, and questions about arithmetic, then refer the works [36, 66, 67]. A proof of undecidability of FL through reduction of the post correspondence problem is exposed in [41]. The same is proved in [47] through reduction of the tiling problem, and in [67] through a reduction of halting problem of Turing machines. You must also be able to find resources from the net for your specific interests related to teaching and research. For example, a recent work is [4]. Applications of FL are found everywhere in mathematics. We will also have an application of it to program verification in Chapter 5. Before doing all these, attempt the following problems.

PROBLEMS

1. Categorize each of the following as a term, a formula, or neither. Draw the corresponding parse trees. Find the scopes of each occurrence of a quantifier in each of the formulas. Also, mark the free and bound occurrences of variables.

(a) $f_5^4(x_1, x_3, f_6^2(x_1, x_3), f_4^2(f_5^2(x_3, x_6), x_4))$

(b) $(\neg\forall x((P_4^1(x_5) \to \neg P^1(c_4)) \wedge P_3^2(x_3, f_1^1(c_3))))$

(c) $f_1^4(x_2, x_4, f_1^2(x_4, x_5), f_5^2(f_1^2(x_4, x_6), c_3))$

(d) $f((x, y), g(x, y, z), h(f(x, y, z), w))$

(e) $\exists x_1\forall x_2(P_2^2(f_3^1(x_2, x_6, f_5^2(f_3^1((x_4, x_3), c_4)), x_3) \to P_2^1(x_7)))$

(f) $\forall x_1\forall x_2(P_1^2(x_1, f_2^1(x_1)) \to P_2^3(f_6^1(x_1), x_2, f_1^3(x_1, x_2, c_6)))$

(g) $\forall x_1\forall x_2(P_1^3(x_1, x_2, x_3) \leftrightarrow \exists x_2 P_1^3(x_1, x_2, x_3))$

(h) $\forall x_3((\forall x_2 P_2^3(x_1, x_2, x_3) \wedge P_4^2(x_3, c_1)) \leftrightarrow \exists x_4 P_5^2(x_5, x_5))$

(i) $\exists x_1\forall x_2(P_1^2(f_1^3(x_1, x_2, f_1^2(x_3, c_1)), x_2) \to P_2^1(f_1^2(x_2, x_4)))$

2. Prove the unique parsing theorem by first proving a prefix theorem that each substring of a formula is also a formula, then it must be equal to the formula.

3. Prove that each occurrence of a quantifier has a unique scope and hence the notion of scope is well defined.

4. Present the syntax of FL in a prefix (Polish) notation and then prove unique parsing for that language. Define also the notion of scope in that language.

5. For each of the terms $x_1, x_2, f_1^1(x_1), f_3^2(x_2, x_3)$, decide whether or not it is free for x_2 in each of the following formulas:

(a) $(P_4^4(x_3, x_2, x_1) \leftrightarrow P_1^3(x_1, c_3, c_2))$

(b) $\forall x_1(P_4^4(x_3, x_2, x_1) \leftrightarrow P_1^3(x_1, c_3, c_2))$

(c) $\forall x_1((P_4^4(x_3, x_2, x_1) \leftrightarrow P_1^3(x_1, c_3, c_2)) \to \exists x_3 P_3^4(x_1, x_2, x_3, x_4))$

(d) $\exists x_3((P_4^4(x_3, x_2, x_1) \leftrightarrow P_1^3(x_1, c_3, c_2)) \wedge P_5^1(x_3))$

6. Show that if x is not a free variable of a formula X, and t is any term, then $X[x/t] = X$.

7. Let X be a formula, x, y are distinct variables and a, b are constants. Show that $X[x/a][y/b] = X[y/b][x/a]$. What happens if x, y are not distinct?

8. A substitution $[x/t]$ is admissible in the context of a formula X if t is free for x in X. Admissible substitutions take terms into terms and formulas into formulas. How do non-admissible substitutions behave?

9. Let X be a formula, x, y be distinct variables, c be a constant, and t be a term free for x in X. Show that if y does not occur in t, then $X[x/t][y/c] = X[y/c][x/t]$. What happens if y occurs in t?

10. Let X be a formula, x be a free variable of X, y be a variable different from x, z be a variable not occurring in X, and t be a term free for x in X. Then show that the term $t[x/y]$ is free for x in $X[x/y]$. further, show that $X[x/t][y/z] = X[y/z][x/t[y/z]]$.

11. Symbolize each of the following into FL:

(a) All men are women.

(b) No man is a woman.

(c) Some men are women.

(d) Some men are not women.

(e) Some man is not a son of any woman.

(f) All men are sons of some woman.

(g) Any man is not a husband of any woman.

(h) Some man is not a husband of any woman.

(i) Some man is not a husband of some woman.

(j) If anyone is Susy's son, then someone is a daughter of Susy's father's nephew.

(k) Anybody's brother's sister is that man's sister.

(l) Sam and Susy have the same maternal grand-father.

(m) No student attends every lecture and no lecture is attended by all students.

(n) If there are no prizes, then nobody buys a ticket.

(o) Successor of a prime number need not be a prime number.

(p) A number is prime or not can be decided in polynomial time.

(q) A number is composite or not can be decided in polynomial time.

(r) There always are triples of numbers such that one's fifth power is the sum of the fifth powers of the other two.

(s) The binary relation R is reflexive.

(t) The binary relation R is an equivalence relation.

(u) The function f is bijective.

(v) The function f is continuous at a point a.

(w) The function is not continuous on a set A.

(x) The sequence $\{x_n\}$ does not converge.

(y) The series $\Sigma_{n=1}^{\infty} x_n$ is not absolutely convergent.

(z) Every even number bigger than four can be expressed as a sum of two prime numbers.

12. Let Mxy stand for the membership relation between an object x and a set y, and Sxy stand for the phrase 'x is a subset of y. Express the following as formulas.

(a) For any two sets, there exists a set which is their union.

(b) Each set has a complement. For any two sets there exists a set which is their intersection.

(c) Any member of a subset is a member of the original set.

(d) There is a set which is a subset of every set, and this set has no member.

(e) For any set there corresponds another set whose members are the subsets of the first set.

13. Let P be a unary predicate, Q be a binary predicate, f be a binary function symbol, and x, y, z be variables. Let $I = (\mathbb{N}, P', Q', f')$ be an interpretation where $P' = \{m : M \text{ is odd}\}$, Q' be the 'less than' relation, and $f'(m, n) = m + n$. Let $\ell(x) = 3, \ell(y) = 4, \ell(z) = 0$. Decide whether the state I_ℓ satisfies the following formulas:

(a) $Pf(xf(x(fx(fxy))))$

(b) $\forall x \forall y Qxf(xy) \to \forall z Qzf(xz)$

(c) $\forall x \forall y (Px \wedge Py \to Pf(xy)) \leftrightarrow \forall z (Px \wedge Py \to Pf(xy))$

(d) $\forall y (\neg Pf(xy) \leftrightarrow Pf(yz)) \vee \forall x (Qxy \to \exists y (Qzy \wedge Qyz))$

14. Which of the following interpretations $I = (D, P')$ are models of the formula $\exists x \exists y \exists z (Pxy \wedge Pyz \wedge (Pzx \wedge \neg Pxz))$?

(a) $D = \mathbb{N}, P' = \{(m, n) : m > n\}$

(b) $D = \mathbb{N}, P' = \{(m, m + 1) : m \geq 4\}$

(c) $D = 2^{\mathbb{N}}, P' = \{(A, B) : A \subseteq B\}$

(d) $D =$ the set of all strings over $\{0, 1\}$,
$P' = \{(m, n) : m < n$ as binary integers$\}$

(e) $D =$ the set of all strings over $\{0, 1\}$
$P' = \{(a, b) : a$ is a substring of $b\}$

15. Let X be any formula and Y be a formula where the variable x is not free. Then show the following:

(a) $Y \to \forall x X \equiv \forall x(Y \to X)$

(b) $Y \to \exists x X \equiv \exists x(Y \to X)$

(c) $\forall x X \to Y \equiv \forall x(X \to Y)$

(d) $\exists x X \to Y \equiv \exists x(X \to Y)$

(e) if $\models Y \to X$, then $\models Y \to \forall x X$

(f) if $\models X \to Y$, then $\models \exists x X \to Y$

What happens if x is allowed to occur free in Y?

16. Try to construct state-models and models for each of the following formulas (or sets of formulas) and determine their satisfiability and validity:

(a) $P(f(c), g(c)) \wedge P(f(c), c) \to Q(c, f(c))$

(b) $\exists x P x f(x) \leftrightarrow P f(x) x \wedge Q x c \wedge (P c c \to \neg Q c)$

(c) $\exists y \forall x(P x y \to P y x) \wedge (P x f(x) \leftrightarrow P c f(c))$

(d) $\forall x \forall y(P x y \to P y x) \wedge \exists x \exists y(Q x y z \leftrightarrow Q y x z \wedge Q y z x)$

(e) $\forall x \forall y(P x y \wedge P y x) \wedge \forall x \forall y(P x y \vee \neg P x y)$

(f) $\forall x \forall y(P x y \to (\neg P x y \to Q x y f(y)))$

(g) $\forall x \forall y \forall z((P x y z \to \neg Q y z) \wedge (Q y z \vee Q x y) \wedge (Q y z \to \neg P x y z))$

(h) $\forall x \exists y \exists z P x y z \wedge \forall x \forall y(P x y y \to \neg P x x y)$

(i) $\forall x \exists y \forall z((P x y \leftrightarrow P y z) \vee (p x y \leftrightarrow \neg P y x))$

(j) $((\forall x(\neg P x \leftrightarrow P f(x)) \vee (\neg Q x \leftrightarrow Q f(f(x)))) \wedge \exists y(P y \to Q y))$

(k) $\{\forall x \neg P x x, \exists x Q x, \forall x \exists y P x y, \forall x(Q x \to \exists y P y x)\}$

(l) $\{\exists x P x x, \forall x \forall y P x y \to (x \approx y)\}$

(m) $\{\forall x(P x \vee Q x) \to \exists x R x, \forall x(R x \to Q x), \exists y(\neg(P y \to Q y))\}$

(n) $\{\forall x \neg P x x, \forall x \exists y P x y, \forall x \forall y \forall z(P x y \wedge P z y \to P z x)\}$

(o) $\forall x \exists y(P x \to Q y) \to \exists y \forall x(P x \to Q y)$

(p) $\{\forall x(P x \to Q x), \forall x(Q x \to R x), \neg \exists x(P x \wedge Q x)\}$

(q) $(\forall x P x \to S) \to \forall x(P x \to S)$, where S is a sentence.

(r) $\{\exists x P x, \exists x Q x, \neg \exists x(P x \wedge Q x)\}$

17. Let $D = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$. Consider the closed formulas

(i) $\forall x(P x \vee Q x \vee R x \vee S x)$

(ii) $\exists x(P x \wedge Q x \wedge R x \wedge S x)$

(iii) $\forall x(P x \vee \forall x(Q x \vee \forall x(R x \vee \forall x S x)))$

(iv) $\exists x(P x \to \exists x Q x)$.

(a) Prescribe ϕ so that all the four formulas are satisfied by the interpretation (D, ϕ).

(b) Construct ψ so that all the four formulas are falsified by the interpretation (D, ψ).

(c) Does there exist an interpretation (D, ξ) where the first and the fourth are falsified but the second and the third are satisfied? If yes, construct ξ, else give reasons why it is not possible.

18. Let X be a closed formula and I be an interpretation. Show that either $I \models X$ or $I \not\models X$ using the import semantics.

19. Let t be a term which contains at least one occurrence of the variable x. Let s be another term, and ℓ, m be valuations under an interpretation I such that m is equivalent to ℓ along x, and $m(x) = \ell(s)$. Let t' be a term obtained from t by substituting each occurrence of x in t by the term s. Show that $m(t) = \ell(t')$.

20. Let $A(x)$ be a formula having at least one free occurrence of the variable x. Let t be a term free for x in $X(x)$. Let ℓ and m be two valuations under an interpretation I such that m is equivalent to ℓ along x and that $m(x) = \ell(t)$. Let $X(t)$ be a formula obtained from $X(x)$ by substituting each free occurrence of x in $X(x)$ by t. Then show that I_m satisfies $X(x)$ iff I_ℓ satisfies $X(t)$.

21. Let X be a formula with free variables $x_1, \ldots, x_m$. Show that there is a state I_l such that $I_l \models X$ iff $\exists x_1 \cdots \exists x_m X$ is satisfiable.

22. Let I be an interpretation of a formula X and ℓ, m be valuations under I such that m is equivalent to ℓ along x and that $m(x) = \ell(x)$. Can you show that I_ℓ satisfies X iff I_m satisfies X?

23. Let X, Y be formulas, and I_ℓ be a state. Which of the following hold?

(a) If $I_\ell \models X \to Y$, then ($I_\ell \models X$ implies $I_\ell \models Y$).

(b) If ($I_\ell \models X$ implies $I_\ell \models Y$), then $I_\ell \models X \to Y$.

(c) If $X \models Y$, then ($\models X$ implies $\models Y$).

(d) If ($\models X$ implies $\models Y$), then $X \models Y$.

24. Let X, Y, Z be three sentences. Answer the following:

(a) If $X \models Y$, does it follow that $\neg X \not\models Y$?

(b) If $X \wedge Y \models Z$, then does it follow that $X \models Y$ and $X \models Z$?

(c) If $X \wedge Y \models Z$, then does it follow that $X \models Y$ or $X \models Z$?

(d) If $X \models Y \vee Z$, then does it follow that $X \models Y$ and $X \models Z$?

(e) If one of $X \models Y$ or $Z \models Y$ holds, then does $X \vee Z \models Y$ hold?

(f) If $X \models (Y \to Z)$, then do $X \models Y$ and/or $X \models Z$ hold?

25. Prove all the laws in Theorem 2.13 using the import semantics.

26. Let P be a binary predicate, $X = \forall x Pxx, Y = \forall x \forall y(Pxy \to Pyx)$, and $Z = \forall x \forall y \forall z(Pxy \wedge Pyz \to Pxz)$. Show that $\{X, Y\} \nvDash Z$, $\{X, Z\} \nvDash Y$, and $\{Y, Z\} \nvDash X$. [This shows that reflexivity, symmetry and transitivity are independent properties.]

27. Consider the domains $\mathbb{N}, \mathbb{Q}, \mathbb{R}$ of natural numbers, rational numbers, and real numbers, respectively.

(a) For each of these sets, construct a sentence which holds in it but not in the other two.

(b) For each pair of these sets, construct a sentence which holds in both of them, but not in the other.

(c) Construct an invalid sentence that holds in all the three sets.

(d) Construct a satisfiable sentence that does not hold in any of the three domains.

28. Show that the sentence

$$\forall x \exists y Pxy \wedge \forall x \neg Pxx \wedge \forall x \forall y \forall z(Pxy \wedge Pyz \to Pxz)$$

is true in some infinite domain but is false in some finite domain.

29. Show that the sentence

$$\forall x Pxx \wedge \forall x \forall y \forall z(Pxz \to Pxy \vee Pyz) \to \exists x \forall y Pxy$$

is true in any finite domain but is false in some infinite domain.

30. Construct a sentence which is true in a domain with no more than m elements, but false in some domain with more than m elements where m equals 1, 2, or 3. Can you have a general formula for generating such a sentence?

31. Translate the following arguments to FL and then check whether they are valid consequences.

(a) Every computer scientist is a logician and also a programmer. Some computer scientists are old fashioned. Therefore, there are old fashioned programmers.

(b) Some computer scientists like all logicians. No computer scientist likes any old fashioned programmers. Therefore, no logician is an old fashioned programmer.

(c) All doctors take Hippocratic oath. All spouses of the persons who take Hippocratic oath cannot be relied upon. Therefore, no spouse of any doctor can be relied upon.

(d) Everyone who commits a crime receives a jail term. Therefore, if there are no jail terms, then nobody commits a crime.

(e) Every businessman likes all his children to study abroad. Therefore, the eldest child of any businessman is the child of a person who likes all his children to study abroad.

32. Let x, y be variables and X, Y, Z be formulas such that x is a free variable of both X, Y, x does not occur in Z, and y is free for x in all of X, Y, Z. Decide whether each of the following holds:

(a) $\forall x X \models X[x/y]$

(b) $X[x/y] \models \exists x X$

(c) If $\models Y$, then $\models \forall x Y$.

(d) If $\models \exists x Y$, then $\models Y[x/y]$.

(e) If $Y \to Z$, then $\exists x Y \to Z$.

(f) If $\models Z \to Y$, then $\models Z \to \forall x Y$.

33. Check the following consequences for validity. Check also whether the set of premises in each case is satisfiable.

(a) $\forall x((\exists y Pxy \wedge Qy) \to \exists y(Ry \wedge Uxy)) \models \exists x \exists y(Pxy \wedge Qy) \to \exists x Rx$

(b) $\{\exists x Px \wedge \exists x Qx \to \exists x(Px \wedge Qx), \forall x \exists y Rxy \to \exists y \forall x Rxy\}$
$\models \forall x(Px \vee Qx) \to \forall x Px \vee \forall x Qx$

(c) $\{\exists x(Px \wedge Qx) \to \forall y(Ry \wedge Hy), \exists x(Rx \wedge \neg Hx)\} \models \forall y(Py \to \neg Qy)$

(d) $\{\forall x(Px \to Qx), \exists x Px, \forall x \neg Qx\} \models \forall x(Px \to Qx) \leftrightarrow \exists x(Px \to Qx)$

(e) $\{\exists x(Px \wedge \forall y(Qy \to Rxy)) \wedge \forall x(Px \to \forall y(Uy \to \neg Rxy))\}$
$\models \forall x(Qx \to \neg Ux)$

34. Suppose that a domain of an interpretation is allowed to be empty. What will be the change in satisfiability and validity of formulas? Will there be formulas which are valid, but now they become invalid, or the converse, etc?

35. Let Σ be a set of formulas and $\Sigma^* = \{X^* : X \in \Sigma\}$, where X^* is the universal closure of X. Let Y be any formula. Show that if $\Sigma \models Y$, then $\Sigma^* \models Y$. Show also that $\Sigma^* \models Y$ does not guarantee that $\Sigma \models Y$.

36. Obtain sdnf, scnf, fdnf, fcnf for the following formulas:

(a) $\forall x Pxf \wedge (\forall x Qx \to \exists y \neg Qx) \vee \forall x \exists y Pxy$

(b) $\forall x \forall y(Pxyz \wedge (\forall x \forall y Qyu \to Rx))$

(c) $\exists x(Px \wedge \forall y(Qy \leftrightarrow Rxy)) \wedge \forall x(Px \to \forall y(Uy \to \neg Rxy))$

(d) $\forall x(Px \leftrightarrow \forall y(Py \to \exists x(Qx \to Qy)) \wedge \exists z Pz) \vee \forall x(Px \to \exists y Qz)$

(e) $(\neg \forall x \neg \forall y \neg \forall z Pxy \to \exists x \exists y(\neg \exists z Qxyz \leftrightarrow Rxy))$

(f) $\forall x(\exists y Pxy \wedge Qy) \to (\exists y(Ry \wedge Uxy) \to Qy)$

37. Under what circumstances the matrix of a formula is a subformula of that formula?

38. Suppose that a formula has a finite model. Can you find a procedure that constructs one such model?

39. Show that every formula is equivalent to one in which no quantifier is within a $\neg$ and no $\neg$ is within another $\neg$.

40. A **negation normal form** formula or an **nnf** is defined by the rules:

(a) For any atomic formula X, both X and $\neg X$ are in nnf.

(b) If X, Y are in nnf, then both $(X \wedge Y)$ and $(X \vee Y)$ are in nnf.

(c) If X is in nnf, then both $\forall x X$ and $\exists x X$ are in nnf.

(d) These are the only way an nnf is generated.

Show that every formula is equivalent to one in nnf.

41. Let X be a sentence in the form $\forall x_1 \cdots \forall x_n X$ having no occurrence of any function symbol.

(a) Give a procedure to determine whether X is valid.

(b) Give a procedure to determine whether X is satisfiable.

(c) What happens to your procedures if function symbols are allowed to occur in X?

(d) Can you modify the procedures so that they would work in the presence of function symbols?

42. Let X be a formula in the form: $\forall x_1 \cdots \forall x_m \exists x Y$, and let P be an $(m+1)$-ary predicate not occurring in Y. Let

$$Z = \forall x_1 \cdots \forall x_m \exists x P x_1 \ldots x_m y \wedge \forall x_1 \cdots \forall x_m \forall x (P x_1 \ldots x_m x \rightarrow Y).$$

Show that X is satisfiable iff Z is satisfiable.

43. Conclude from Problem 42 that there is an algorithm to transform any formula X to a formula Z with a prefix of the form $\forall x_1 \cdots \forall x_m \exists y_1 \cdots \exists y_n$ such that X is satisfiable iff Z is satisfiable. Can you construct a formula W similar to Z so that X is valid iff W is valid?

44. Let X be a prenex form formula such that every atomic subformula contains an existentially quantified variable whose quantifier is in the scope of each quantifier that uses some other variable occurring in that atomic subformula. Show that X is satisfiable iff its matrix is satisfiable.

45. Let X be a closed formula having only unary (monadic) predicates and having no function symbols. (X is a sentence of the monadic first order logic.) Let I be a model of X. Define an equivalence relation on the domain of I by: a is equivalent to b iff for each predicate P occurring in X, $P'(a)$ holds whenever $P'(b)$ holds. Show that the set of equivalence classes also forms a model of X. Use this to show that if a sentence X of monadic first order logic is satisfiable, then it has a finite model. Can you have a procedure to decide whether such a sentence is satisfiable?

46. A sentence in monadic first order logic is said to be in **miniscope form** if whenever $\forall xY$ or $\exists xY$ is a subformula, the only free variable in Y is x. Show that every sentence of monadic first order logic is equivalent to one in miniscope form.

47. *(Craig's interpolation)* Let X, Y be formulas having at least one common atomic subformula. A formula Z having all its atomic subformulas among the common ones of X and Y is called an **interpolant** of $X \to Y$ iff $\models X \to Z$ and $\models Z \to Y$. Show that, if $\models X \to Y$, then either X is unsatisfiable or Y is valid or $X \to Y$ has an interpolant. Show also that whenever an interpolant of $X \to Y$ exists, there is, in fact, an interpolant in which all the constants are among the common constants of X and Y.

48. Construct the Herbrand domain and the Herbrand expansions for the following formulas. Then decide their satisfiability.

(a) $\forall x(Px \vee \neg\forall y\exists z(Qyxz \vee \neg\exists uQuxz))$

(b) $\forall x(Px \vee \neg\forall y\exists z(Qyxz \vee \neg\exists uQuxz))$

(c) $\forall x(\neg Pxx \wedge \exists yPxy \wedge \forall y\forall z((Pxy \wedge Pyz) \to Pxz))$

(d) $\forall y\exists x(Pyx \wedge (Qy \leftrightarrow \neg Qx)) \wedge \forall x\forall y\forall z((Oxy \wedge Pyz) \to Pxz)$
$\wedge \forall x\neg\exists y(Qx \wedge Qy \wedge Pxy)$

49. Show that if a sentence has a model of m elements, then it has models of n elements for each $n > m$ and also it has an infinite model.

50. *Skolem-Löwenheim upward theorem*: If a first order theory has a model of infinite cardinality, then it has models of each higher cardinality. Prove this statement. [*Hint*: See [67].]

3

Resolution

3.1 Introduction

You have seen how normal forms in PL can be used to decide whether a proposition is valid, invalid, satisfiable, or unsatisfiable. To repeat, if you have a pair of complementary literals in every disjunctive clause of a cnf, then the cnf is valid; otherwise, the cnf is invalid. Similarly, a dnf is unsatisfiable if each conjunctive clause in it contains a pair of complementary literals, else it is satisfiable. Suppose that you have a dnf which you know to be satisfiable. You are further interested in determining its validity. What do you do? You apply the law of distributivity to convert the dnf into a cnf, and then decide. For example, using distributivity on the dnf

$$A = (\neg p \wedge r) \vee (\neg q \wedge r) \vee (\neg r \wedge p)$$

you get an equivalent cnf:

$$\begin{aligned} A' = {} & (\neg p \vee \neg q \vee \neg r) \wedge (\neg p \vee \neg q \vee p) \wedge (\neg p \vee r \vee \neg r) \wedge (\neg p \vee r \vee p) \\ & \wedge (r \vee \neg q \vee \neg r) \wedge (r \vee \neg q \vee p) \wedge (r \vee r \vee \neg r) \wedge (r \vee r \vee p) \end{aligned}$$

Now, A' is in cnf, where the fifth (and the last also) clause does not contain a pair of complementary literals. Hence A is not valid. You also know how to construct an interpretation from such a clause which would falsify the proposition.

***Exercise* 3.1** Construct three interpretations, each of which falsifies the proposition A above.

Similar is the case of checking satisfiability of a cnf. For example, (compare this with the above dnf), take the cnf:

$$B = (\neg p \vee r) \wedge (\neg q \vee r) \wedge (\neg r \vee p)$$

As earlier, distribute $\wedge$ over $\vee$ to get an equivalent dnf:

$$\begin{aligned} B' = {} & (\neg p \wedge \neg q \wedge \neg r) \vee (\neg p \wedge \neg q \wedge p) \vee (\neg p \wedge r \wedge \neg r) \vee (\neg p \wedge r \wedge p) \\ & \vee (r \wedge \neg q \wedge \neg r) \vee (r \wedge \neg q \wedge p) \vee (r \wedge r \wedge \neg r) \vee (r \wedge r \wedge p) \end{aligned}$$

You see that B is satisfiable since there is a (at least one) clause in B' which does not contain a pair of complementary literals. Since the problems are dual to each other, we will only consider the latter problem of determining satisfiability of a cnf. Moreover, the method of distributing $\wedge$ over $\vee$, and $\vee$ over $\wedge$ is a costly affair. Can it be made cheaper? The problem is how to determine in a different way than by crude distribution of $\wedge$ over $\vee$ that a cnf is satisfiable? For dnfs, a similar question of validity may be posed. However, by duality, an answer to the first would give us an answer to the second.

The aim of this chapter is to develop a mechanical strategy for answering this question. You will also see how this strategy is extended to FL. We will consider a simple example first. Instead of B, consider the cnf:

$$C = (\neg p \vee r) \wedge (q \vee \neg r)$$

By distributivity,

$$C \equiv C' = (\neg p \wedge q) \vee (\neg p \wedge \neg r) \vee (r \wedge q) \vee (r \wedge \neg r)$$

Now, suppose that $i \models C'$. As $i \not\models (r \wedge \neg r)$,

$$\text{either } \; i \models \neg p, i \models q \; \text{ or } \; i \models \neg p, i \models \neg r \; \text{ or } \; i \models q, i \models r$$

This means, in any case, $i \models \neg p$ or $i \models q$, or both. That is, $i \models \neg p \vee q$.

Alternatively, if you take C as it is, and look for its model, you can see that $i \models C$ when i is a model of both $\neg p \vee r$ and $q \vee \neg r$. But i cannot satisfy both r and $\neg r$ simultaneously. If i is a model of r, then i is not a model of $\neg r$ and, thus, i has to be a model of q. On the other hand, if $i \not\models r$, then since $i \models \neg p \vee r$, i has to be a model of $\neg p$. Hence, $i \models q$ or $i \models \neg p$.

Now, looking at the form of C, where $C = (\neg p \vee r) \wedge (q \vee \neg r)$, we see that in one clause there is r, and in the other there is $\neg r$. If we omit this pair of complementary literals, we will get $\neg p$ and q. We $\vee$ them together to get $\neg p \vee q$. And then any model of C *must* be a model of this proposition.

Similarly, look at the rule Modus Ponens. Since, $A \to B \equiv \neg A \vee B$, you can rewrite this rule as "from A and $\neg A \vee B$, derive B". Here, omission of the complementary literals gives you B. Similarly, the hypothetical syllogism, or transitivity of implication, which says that "from $A \to B$ and $B \to C$, derive $A \to C$", can be rewritten as "from $\neg A \vee B$ and $\neg B \vee C$, derive $\neg A \vee C$". Here again, omitting the pair $B, \neg B$ from the premises, you reach at the conclusion $\neg A \vee C$. Does this method of omitting a pair of complementary literals work in general? We will try to answer this question in the next section and then develop our mechanical strategy, to be called resolution.

3.2 Resolution in PL

Now that all our propositions are in cnfs, we can represent them in a set notation. For example, the clause $p \vee q$, can be rewritten as $\{p, q\}$ and the cnf $(p \vee q) \wedge (r \vee s)$ can be rewritten as $\{\{p, q\}, \{r, s\}\}$. While writing back to the original form, there will be no confusion since a set of sets (of literals) is a conjunction of sets of literals and a set of literals is a disjunction of literals. This notation will make the use of the phrase 'a clause contains a pair of complementary literals' a formal one. But there is a hitch. Once you write a clause as a set of literals, what does an empty set of literals represent? Well, consider $\{p\}$ and $\{p, q\}$ as clauses. $\{p\}$ represents the proposition p and $\{p, q\}$ represents the clause $p \vee q$. we see that $p \models p \vee q$. That is, whenever $A \subseteq B$, we have $A \models B$.

***Exercise* 3.2** Let A and B be two sets of literals (remember, disjunctive clauses), then show that $A \subseteq B$ implies that $A \models B$.

Now, since the empty set is a subset of every set, the empty clause entails every clause. So, what is that formula which entails every clause? A little thought shows that it must be the propositional constant $\bot$. (Show it.) Another way of looking at it is that a disjunctive clause is true under an interpretation only when one of its literals is true; else, it is false. But there is no literal in the empty clause to become true, so, it is false.

What about the empty set of clauses? Now, a cnf is a conjunction of clauses. A conjunction is false under an interpretation only when there is at least one clause in it which is false. But there is none. Hence, it cannot be false. That is, an empty set of clauses is always true. Moreover, if A and B are two sets of clauses (conjunctions of the clauses) with $B \supset A$, we see that B is $A \wedge X$ for some cnf X. Thus, $B \models A$. Since each set of clauses is a superset of the empty set of clauses, each cnf entails the empty cnf. Then you can see that the only such cnf which is entailed by every cnf has to be $\top$.

When p is a propositional variable, we have its negation as $\neg p$ which is obviously a literal. But then its negation, $\neg\neg p$ is not a literal. In this case, we will write the negation of the literal $\neg p$ as p itself, and generically, we will accept that $\neg q$ is a literal even when $q = \neg p$; the literal $\neg q$ being equal to p. Thus we have the following conventions:

If $q = \neg p$ is a literal, then its negation $\neg q$ is the literal p.

A clause is a set of literals.

A cnf is a set of sets of literals, a set of clauses.

The empty set of literals is $\bot$.

The empty set of sets of literals is $\top$.

Our strategy may now be formalized. Let A and B be two clauses (sets of literals). If there is a literal p such that $p \in A$ and $\neg p \in B$, then the **resolvent** $res(A, B; p)$ of A and B with respect to the literal p (or $\neg p$) is the clause $(A - \{p\}) \cup (B - \{\neg p\})$. We also say that this resolvent has been obtained by **resolving upon the variable** p. In such a case, both the literals p and $\neg p$ are also termed as **biform literals**. The variable p (or $\neg p$) is called a **biform variable**. If the literal p is clear from the context, we will simply write $res(A, B; p)$ as $res(A, B)$.

For example, if $A = \{\neg p, r\}, B = \{q, \neg r\}$, by resolving upon q, we get $res(A, B) = res(A, B; q) = \{\neg p, q\}$. If $A = \{\neg p, q, r\}, B = \{\neg q, \neg r\}$, then $res(A, B) = \{\neg p, r, \neg r\}$ resolving upon the variable q. By resolving upon r, we also get $res(A, B) = \{\neg p, q, \neg q\}$.

However, here $res(A, B) \neq \{\neg p\}$ as you would have obtained by omitting both the pairs $q, \neg q$ and $r, \neg r$. You cannot cancel more than one pair at a time! Taking resolvents does not allow it. Why is it so? You have already seen that if $i \models \{\neg p, r\}, i \models \{q, \neg r\}$, then $i \models \{\neg p, q\}$.

Similarly, you expect that if $i \models \{\neg p, q, r\}$ and $i \models \{\neg q, \neg r\}$, then $i \models \{\neg p, r, \neg r\}$. But will it be that $i \models \neg p$? Not necessarily, since the interpretation j with $j(p) = j(q) = 1, j(r) = 0$, is a model of $\{\neg p, q, r\}$ and also of $\{\neg q, \neg r\}$, but $j \not\models \neg p$. (Remember $\{\neg p, q, r\}$ is, by definition, $\neg p \vee q \vee r$.) Let us prove what you expect.

Theorem 3.1 (Resolution Principle for PL) *Let A and B be two clauses, and p be a literal such that $p \in A$ and $\neg p \in B$. Then $\{A, B\} \models res(A, B; p)$. That is, the resolvent of two clauses is their logical consequence.*

Proof Let $A = l_1 \vee l_2 \vee \cdots \vee l_k \vee p$ and $B = m_1 \vee m_2 \vee \cdots \vee m_n \vee \neg p$, where l_i and m_j are literals. Then $res(A, B; p) = l_1 \vee l_2 \vee \cdots \vee l_k \vee m_1 \vee m_2 \vee \cdots \vee m_n$. Let v be an interpretation with $v \models A$ and $v \models B$. Then what can be $v(p)$? We have two cases: (a) $v(p) = 1$, and (b) $v(p) = 0$.

In case (a), $v \not\models \neg p$. Hence, $v \models B$ implies that $v \models m_1 \vee m_2 \vee \cdots \vee m_n$. Thus, $v \models res(A, B; p)$.

In case (b), $v \not\models p$. But $v \models A$. So, $v \models l_1 \vee l_2 \vee \cdots \vee l_k$. Therefore, $v \models res(A, B; p)$. This completes the proof. ∎

***Exercise* 3.3** If clauses are taken as conjunctive clauses, then show that the resolution principle does not work.

Now you realize that resolution is a method to work with cnf only. Let i be an interpretation. For conjunctive clauses you can of course show that if $i \not\models A$ and $i \not\models B$, then $i \not\models res(A, B; p)$. However, we have planned to work with disjunctive clauses only!

***EXAMPLE* 3.1** Consider the cnf $\Sigma = \{\neg p \vee q, \neg q \vee r, p, \neg r\}$. Here, $res(\neg p \vee q, \neg q \vee r; q) = \neg p \vee r, res(\neg p \vee q, p; p) = q, res(\neg q \vee r, \neg r; r) = \neg q$. By the resolution principle, $\Sigma \models \neg p \vee r,\ \Sigma \models q, \Sigma \models \neg q$. Since Σ entails each of its members, we obtain $\Sigma \models \neg p \vee q,\ \Sigma \models \neg q \vee r,\ \Sigma \models p,\ \Sigma \models \neg r$. Taking further resolvents, $res(\neg p \vee r, \neg r; r) = \neg p$, $res(q, \neg q; q) = \bot$, the empty clause, and $res(q, \neg q \vee r; q) = r$. Other resolvents will be repeated as earlier. In addition to the earlier conclusions, you conclude, by the resolution principle, that $\Sigma \models \neg p,\ \Sigma \models \bot,\ \Sigma \models r$.

Of all these conclusions from Σ, the special conclusion $\bot$ signifies something important. It says that if $i \models \Sigma$, then i must also be a model of $\bot$. But there cannot be any model of $\bot$. Therefore, Σ is unsatisfiable.

We may write the resolution principle as an inference rule and then go on extending a deduction towards deriving the empty clause $\bot$. Our **inference rule of resolution for PL** would look like

(RPL) $\dfrac{A \quad B}{res(A, B)}$

where the resolution is taken on some biform literal p.

Remember that in the above rule p is a biform literal means that either $p \in A, \neg p \in B$, or $\neg p \in A, p \in B$. Using this inference rule, the above workout in Example 3.1 can be rewritten as a *proof* for showing that Σ is unsatisfiable. Formally, let Σ be a set of clauses. A **resolution proof** of $\Sigma \models w$ is a finite sequence of clauses, where each clause is either a clause in Σ or is obtained (derived) by an application of the rule (RPL) from two earlier clauses, and the last clause in the sequence is w. A **resolution proof of unsatisfiability** of Σ is a resolution proof of the consequence $\Sigma \models \bot$. A resolution proof of unsatisfiability of Σ is also called a **resolution refutation** of Σ. Here is the proof of unsatisfiability of $\Sigma = \{\neg p \vee q, \neg q \vee r, p, \neg r\}$ as worked out in Example 3.1.

1.	$\neg p \vee q$	H
2.	p	H
3.	q	$res(1, 2; p)$
4.	$\neg q \vee r$	H
5.	$\neg r$	H
6.	$\neg q$	$res(4, 5; r)$
7.	$\bot$	$res(3, 6; q)$

In this proof, we have added the line numbers and the right-most column for documentation. They help us to read the proof; the proof is simply the sequence of clauses in the middle column read from top to bottom. The abbreviation 'H' used in the documentation column refers to the word 'hypothesis', meaning that the clause in that line is indeed an element of the given set Σ of premises. Sometimes, instead of writing the full $res(1, 2; p)$ as in line 3 above, we will only quote (RPL) to say that the clause in that line has been derived by an application of the resolution rule from some two earlier clauses resolving upon some biform variable.

The above refutation can also be depicted as a tree, a directed acyclic graph (DAG), in general. See Figure 3.1 for such a **refutation DAG**. The directed acyclic graph here happens to be a tree; and it is drawn upside down. It has been generated from its leaves which are the given premises. In each deeper level a clause (as a node) is added by applying the resolution rule, i.e., if A, B are the clauses whose resolvent is C, then C appears as the child of the nodes labelled A and B. The literal upon which the resolvent is taken is not usually mentioned in such a DAG.

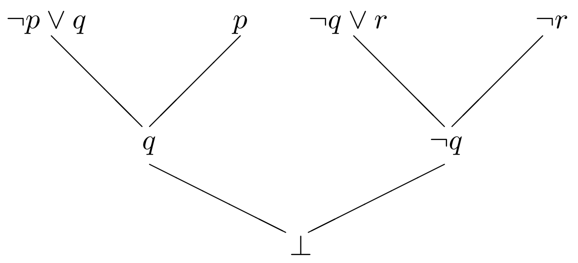


Figure 3.1 A refutation tree.

Though resolution proofs are defined for consequences $\Sigma \models w$, there is a slight advantage in fixing w for ever. This can be done by using RAA, that is, for proving $\Sigma \models w$, it is enough to show that $\Sigma \cup \{\neg w\}$ is unsatisfiable. In this approach, one does not define a resolution proof as we have done; rather a resolution refutation is only defined. Then a proof of $\Sigma \models w$ is redefined to be a resolution refutation of $\Sigma \cup \{\neg w\}$. The advantage is that we now have a unique target in each proof, i.e., the empty clause $\perp$.

***EXAMPLE* 3.2** Show by resolution that

$$\{\neg p \to q, p \to r \vee s, r \to t \wedge u, u \wedge \neg s \to \neg t\} \models \neg s \to q$$

Solution Using RAA, we only construct a resolution refutation of the set

$$\{\neg p \to q, p \to r \vee s, r \to t \wedge u, u \wedge \neg s \to \neg t, \neg(\neg s \to q)\}$$

First of all, the set has to be converted into a cnf. Since the set of propositions means the conjunction of all its elements, (Why?) it is enough to convert each of its elements to cnfs and then take their union. That is, we can convert each premise and the negation of the conclusion into cnf's and then take all the clauses obtained together to form the required set. This set must have a resolution refutation. To this end, we see that

$$\neg p \to q \equiv p \vee q, \quad p \to r \vee s \equiv \neg p \vee r \vee s, \quad r \to t \wedge u \equiv (\neg r \vee t) \wedge (\neg r \vee u),$$

$$u \wedge \neg s \to \neg t \equiv \neg u \vee s \vee \neg t, \quad \neg(\neg s \to q) \equiv (\neg s) \wedge (\neg q).$$

Hence, we have the set

$$\Sigma = \{p \vee q, \neg p \vee r \vee s, \neg r \vee t, \neg r \vee u, \neg u \vee s \vee \neg t, \neg s, \neg q\}$$

Here is a (resolution) refutation of Σ :

1.	$p \vee q$	H
2.	$\neg q$	H
3.	p	RPL
4.	$\neg p \vee r \vee s$	H
5.	$r \vee s$	RPL

6.	$\neg s$	H
7.	r	RPL
8.	$\neg r \vee t$	H
9.	t	RPL
10.	$\neg r \vee u$	H
11.	u	$res(7, 10; r)$
12.	$\neg u \vee s \vee \neg t$	H
13.	$s \vee \neg t$	RPL
14.	$\neg t$	$res(6, 13; s)$
15.	$\perp$	$res(9, 14; t)$

You can also have a refutation DAG, as shown in Figure 3.2.

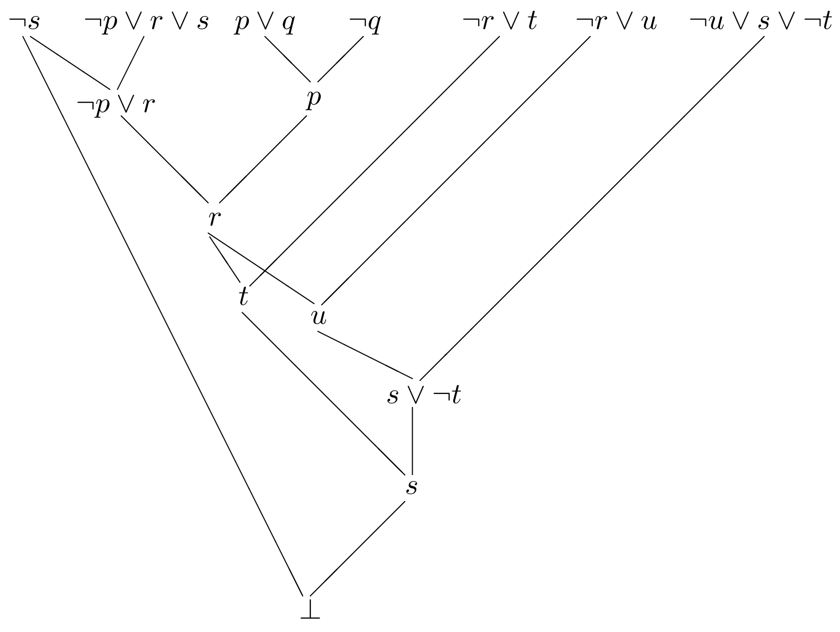


Figure 3.2 A refutation DAG.

***Exercise* 3.4** There is a difference between the resolution refutation above and the refutation DAG of Figure 3.2. Find it.

Is the resolution refutation mechanical? You had to use your intuition to derive $\perp$ from the premises by selectively choosing the premises one after another. How can a machine do it? Can you write a procedure to construct a resolution proof even if the proof is not the shortest one?

A crude method is to generate all possible resolvents of the premises, add them to the premises, and then generate more resolvents till you get $\perp$. Will the procedure work? Yes, provided the resolvents, resolvents of resolvents, etc. are finite in number. But this is a must since there are at the most only a finite number of distinct clauses which might be built upon a finite number of propositional variables. If the cnf has n number of propositional variables, then there are $2n$ number of literals, and then, since each clause is a subset of literals, there are at the most 2^{2n} number of clauses. Among them, there are trivial clauses of the form $l_1 \vee \cdots \vee l_m \vee p \vee \neg p$ which are equivalent to $\top$, and hence, can be omitted. Then you are left with 3^n number of clauses. Thus, quite mechanically, the procedure will terminate and generate $\bot$ somewhere, if at all $\bot$ can be generated. On the other hand, if $\bot$ is not generated but the procedure terminates, then of course, the cnf will be satisfiable.

The first property that 'if $\bot$ is generated, then the given cnf is unsatisfiable' is the **soundness** of resolution. And its converse that "if the cnf is satisfiable, then $\bot$ is eventually generated" is called the **completeness** of resolution. Both the properties are commonly called **adequacy** of resolution. We must be able to prove adequacy of resolution. However, before attempting a proof for adequacy it will be helpful to formally describe the method. Let us do that first.

For any set B of clauses (sets of literals), define

$$\begin{aligned} R(B) = B \cup \{C : C = res(C_1, C_2; p) &\text{ for some clauses} \\ C_1, C_2 \in B, C_1 \neq C_2, &\text{ and for some biform literal } p\}. \end{aligned}$$

Let A be a given set of clauses. Define $R_n(A)$ inductively by

$$R_0(A) = A, \quad R_{i+1}(A) = R(R_i(A))$$

Write $R^*(A) = \cup_{n \in \mathbb{N}} R_n(A)$.

The set $R(A)$ is the set of all clauses of A along with resolvents of all possible pairs of clauses of A that could be resolved upon some biform literal. Sets $R(R(A))$ etc. form an increasing sequence of sets of clauses:

$$A = R_0(A) \subseteq R_1(A) \subseteq R_2(A) \subseteq \cdots \subseteq R_n(A) \subseteq \cdots R^*(A)$$

The set $R^*(A)$ is called the **resolvent closure** of the cnf A. Since there are only finite number of possible clauses which can be generated from A, we have $R^*(A) = R_n(A) = R_{n+1}(A)$ for some natural number n. Now, soundness and completeness of resolution can be restated as

Soundness: If $\bot \in R^*(A)$, then A is unsatisfiable.

Completeness: If A is unsatisfiable, then $\bot \in R^*(A)$.

To study these properties, we start with an enumeration of all the propositional variables. We had them already as $p_0, p_1, p_2, \ldots$

Let A_m denote the set of all clauses which can be formed from the first m propositional variables including the propositional constant $\bot$. Note that we do not need to include $\top$ separately since a disjunctive clause $p \vee \neg p$ is equivalent to $\top$. For example, $A_0 = \{\bot, p_0, \neg p_0, p_0 \vee \neg p_0\}$, etc. Let A be any cnf and $R^*(A)$ be its resolvent closure. We show the following statement.

Lemma 3.2 *If $R^*(A) \cap A_m$ is satisfiable, then $R^*(A) \cap A_{m+1}$ is satisfiable.*

Proof All propositional variables that occur in $R^*(A) \cap A_m$ are from $\{p_0, p_1, \ldots, p_{m-1}\}$. Suppose that $R^*(A) \cap A_m$ is satisfiable. Then we have an interpretation $i : \{p_0, \ldots, p_{m-1}\} \to \{0, 1\}$ which is a model of all the clauses in $R^*(A) \cap A_m$. Construct two more interpretations from i by extending it to the set $\{p_0, \ldots, p_{m-1}, p_m\}$ as in the following:

Let $j, \bar{j} : \{p_0, \ldots, p_m\} \to \{0, 1\}$ be such that $j(p_k) = \bar{j}(p_k) = i(p_k)$ for $1 \le k \le m - 1$, and $j(p_m) = 0,\ \bar{j}(p_m) = 1$.

Now suppose that $R^*(A) \cap A_{m+1}$ is unsatisfiable. Then neither j nor $\bar{j}$ satisfies all the clauses in $R^*(A) \cap A_{m+1}$. Thus, there are clauses C and $\bar{C}$ in $R^*(A) \cap A_{m+1}$ such that $j \not\models C$ and $\bar{j} \not\models \bar{C}$. Can it happen that neither p_m nor $\neg p_m$ is a member of C? If yes, then all the propositional variables occurring in C are from $\{p_0, \ldots, p_{m-1}\}$ and then C must be in $R^*(A) \cap A_m$. As j agrees with i on the set $\{p_0, \ldots, p_{m-1}\}$, we have $j(C) = i(C)$. Since i is a model of $R^*(A) \cap A_m$, $i \models C$. But then we arrive at a contradiction that $j \not\models C$ and $i \models C$. Therefore, at least one of p_m or $\neg p_m$ is in C.

Now, if $\neg p_m$ is in C, then $C = D \vee \neg p_m$ for some clause D. (If $C = \neg p_m$, then we take $D = \top$.) As $j \not\models C$, you see that $j \not\models \neg p_m$. But this is not correct, since $j(p_m) = 0$. Thus, $p_m \in C$. Then, C and $\bar{C}$ can be written as $C = D \vee p_m$, $\bar{C} = \bar{D} \vee \neg p_m$ for some clauses $D, \bar{D} \in R^*(A) \cap A_m$. The clause $B = res(C, \bar{C}; p_m) = D \vee \bar{D} \in R^*(A) \cap A_m$. As i is a model of $R^*(A) \cap A_m$, we have $i(D \vee \bar{D}) = 1$. We must explore two possibilities: (a) $i(D) = 1$, or (b) $i(\bar{D}) = 1$.

In case (a), since i agrees with j on $\{p_0, \ldots, p_{m-1}\}$, and both $D, \bar{D}$ contain (possibly) variables from $\{p_0, \ldots, p_{m-1}\}$, we get $j(D) = 1$. This contradicts the fact that $j \not\models C$.

In case (b), i agrees with $\bar{j}$ and $i(\bar{D}) = 1$ gives $j(\bar{D}) = 1$, contradicting the fact that $j \not\models \bar{C}$. This contradiction shows that $R^*(A) \cap A_{m+1}$ is satisfiable. ▮

Lemma 3.3 *For every $n \in \mathbb{N}$, if $\bot \notin R^*(A) \cap A_n$, then $R^*(A) \cap A_n$ is satisfiable.*

Proof We use induction on n. For $n = 0$, $A_n = A_0 = \{\bot, p_0, \neg p_0, p_0 \vee \neg p_0\}$. Assume that $\bot \notin R^*(A) \cap A_0$. Now, $R^*(A) \cap A_0$ cannot contain both p_0 and $\neg p_0$, since otherwise, their resolvent which is $\bot$, will also be in $R^*(A) \cap A_0$. If both p_0 and $p_0 \vee \neg p_0$ are in $R^*(A) \cap A_0$, then their resolvent $\neg p_0$ will also be there, and then another resolvent of p_0 and $\neg p_0$ will generate $\bot$. Similarly, both $\neg p_0$ and $p_0 \vee \neg p_0$ cannot be in $R^*(A) \cap A_0$. Hence, $R^*(A) \cap A_0$ can be one of the singleton sets $\{\{p_0\}\}$, $\{\{\neg p_0\}\}$ or $\{\{p_0 \vee \neg p_0\}\}$. In either case, it is satisfiable. (Alternatively, you could have omitted the trivial clause $p_0 \vee \neg p_0$ to make the argument shorter.)

Lay out the induction hypothesis that $\bot \notin R^*(A) \cap A_m$ implies that $R^*(A) \cap A_m$ is satisfiable. Suppose that $\bot \notin R^*(A) \cap A_{m+1}$. Since $R^*(A) \cap$

$A_m \subseteq R^*(A) \cap A_{m+1}$, $\perp \notin R^*(A) \cap A_m$ either. By induction hypothesis, $R^*(A) \cap A_m$ is satisfiable. By Lemma 3.2, we conclude that $R^*(A) \cap A_{m+1}$ is satisfiable. ∎

We take help from these lemmas to prove our main result.

Theorem 3.4 (Adequacy of Resolution) *Let A be any cnf and $R^*(A)$ be its resolvent closure. Then A is unsatisfiable iff $\perp \in R^*(A)$.*

Proof If $\perp \in R^*(A)$, then by resolution principle and induction, it is clear that $A \models \perp$. This shows that A is unsatisfiable.

Conversely, suppose that $\perp \notin R^*(A)$. Let A have only the propositional variables from the set $\{p_0, p_1, \ldots, p_{m-1}\}$. Then $\perp \notin R^*(A) \cap A_m$. By Lemma 3.3, $R^*(A) \cap A_m$ is satisfiable. However, $A \subseteq R^*(A) \cap A_m$. Hence A is also satisfiable. ∎

In other words, you have proved that $\perp \in R^*(A)$ iff $R^*(A)$ is unsatisfiable. This is the closure property of the resolvent closure. Look at Example 3.1 again. You will be convinced that it is indeed wasteful to compute the whole of $R^*(A)$ in case that A is unsatisfiable, as is prescribed by the above discussion. When solving the examples above you had in fact taken some choices of clauses in each stage which might be resolved upon. It is better to develop such strategies so that resolution can be made faster. See the follow up remarks for such strategies.

We will consider a very general strategy here in cutting down some wasteful generation of resolvent clauses. One such obvious strategy is that "once $\perp$ has been generated, do not proceed further". Similarly, **trivial clauses** which contain a pair of complementary literals may be omitted at any stage of taking further resolvents. Another strategy can be developed by looking at a somewhat simple example.

***EXAMPLE* 3.3** Use resolution to decide the satisfiability of the cnf

$$A = (\neg p \vee q) \wedge (p \vee q) \wedge (\neg p \vee \neg q)$$

Solution

$$R_0(A) = \{\{\neg p, q\}, \{p, q\}, \{\neg p, \neg q\}\},$$

$$R_1(A) = R_0(A) \cup \{\{q\}, \{\neg p\}, \{q, \neg q\}, \{p, \neg p\}\}.$$

For computing $R_2(A)$, we have to take further resolvents of clauses in $R_1(A)$. You can verify that further resolvents do not add any new clause. Therefore, $R^*(A) = R_2(A) = R_1(A)$. But $\perp \notin R^*(A)$; hence A is satisfiable.

While computing $R_2(A)$, among others, you had to take resolvents of $\{p, q\}$ with the possible clauses, namely, $\{\neg p\}, \{q, \neg q\}, \{p, \neg p\}$. This gives $\{q\}, \{p, q\}, \{\neg p, q\}$. This is wasteful due to two reasons. First, there is no need to resolve with the trivial clauses as we have mentioned earlier. This is because $\{\{\neg p\}, \{q, \neg q\}, \{p, \neg p\}\}$ is logically equivalent to $\{\{\neg p\}\}$. It is enough to keep only the nontrivial clauses. The nontrivial clauses are also called **fundamental clauses**; these are the clauses which do not contain a pair of complementary literals. The strategy is

Delete all nonfundamental clauses.

The second source of wasteful generation is to keep all of the clauses $\{q\}, \{p, q\}, \{\neg p, q\}$. It is because the clause $\{q\}$ is already a subset of other clauses. So what? When you write $R_2(A)$ as a cnf, it will look like

$$\cdots q \wedge (p \vee q) \wedge (\neg p \vee q) \cdots$$

Now, $q \wedge (p \vee q)$ simplifies to q and $q \wedge (\neg p \vee q)$ also simplifies to q. Thus it should be enough to keep only q. The strategy is

Keep only a subset and delete all its supersets.

For example, $R_1(A)$ above would become modified to $\{\{q\}, \{\neg p\}\}$. You can check at this point that this new set is logically equivalent to the set $R_1(A)$ obtained earlier.

To express these strategies formally, let C and D be two clauses. C is said to **subsume** D if $C \subseteq D$. If A is a set of clauses, then the **residue of subsumption** of A is the set

$$\begin{aligned} RS(A) = \{C \in A : \, & C \text{ is not subsumed by any other clause of } A \\ & \text{and } C \text{ is fundamental}\} \end{aligned}$$

Thus, $RS(R_1(A)) = \{\{q\}, \{\neg p\}\}$. You can also verify that $A \equiv RS(A)$ as cnfs. While generating $R^*(A)$, we can take residue of subsumption on each resolvent set $R_n(A)$ and then proceed further. That is, for a set of clauses A, we compute the sequence

$$A_0 := A, \; B_0 := RS(A_0); \; A_1 := R(B_0), B_1 = RS(A_1);$$

$$A_2 = R(B_1), B_2 = RS(A_2); \; \ldots . \; \text{ until } A_{n+1} = B_n \text{ for some } n.$$

The termination criterion $A_{n+1} = B_n$ is bound to be met because the totality of all clauses that could be generated from the propositional variables of A are finite in number. Whenever for some n, we have $B_n = A_{n+1}$, we denote the set of clauses as $RS^*(A)$, that is, $RS^*(A) = B_n = A_{n+1}$. Then from the adequacy of resolution it will follow that $\bot \in RS^*(A)$ iff A is unsatisfiable.

***EXAMPLE* 3.4** Show by resolution that the following consequence is valid: $\{p \vee q \to r, r \to s \vee t, s \to u, \neg(\neg t \to u)\} \models \neg p$.

Solution Here we will not have a refutation by choosing the clauses. We will rather follow the above strategy literally, and see how a machine might work on the problem. The clause set corresponding to this consequence is (include negation of the conclusion and then convert each proposition)

$$A = \{\{\neg p, r\}, \{\neg q, r\}, \{\neg r, s, t\}, \{\neg s, u\}, \{\neg t\}, \{\neg u\}, \{p\}\}$$

We notice that there is no nonfundamental (tautological or trivial) clause in A and no clause is a proper subset of another. Hence,

$$B_0 = RS(A) = A_0 = A.$$

$$A_1 = R(B_0) = \{\{\neg p, r\}, \{\neg q, r\}, \{\neg r, t, s\}, \{\neg s, u\}, \{\neg t\}, \{\neg u\}, \{p\}, \{r\}, \{\neg p, t, s\}, \{\neg q, t, s\}, \{\neg r, t, u\}, \{\neg r, s\}, \{\neg s\}\}$$

since

$$res(\{\neg p, r\}, \{p\}; p) = \{r\},$$
$$res(\{\neg p, r\}, \{\neg r, t, s\}; r) = \{\neg p, t, s\},$$
$$res(\{\neg q, r\}, \{\neg r, t, s\}; r) = \{\neg q, t, s\},$$
$$res(\{\neg r, t, s\}, \{\neg s, u\}; s) = \{\neg r, t, u\},$$
$$res(\{\neg r, t, s\}, \{\neg t\}; t) = \{\neg r, s\},$$
$$res(\{\neg s, u\}, \{\neg u\}; u) = \{\neg s\}.$$

Then,

$$B_1 = RS(A_1) = \{\{\neg t\}, \{\neg u\}, \{p\}, \{r\}, \{\neg p, t, s\}, \{\neg q, t, s\}, \{\neg r, t, u\}, \{\neg r, s\}, \{\neg s\}\}$$

as $\{r\}$ subsumes $\{\neg p, r\}, \{\neg q, r\}$; $\{\neg r, s\}$ subsumes $\{\neg r, t, s\}$, and $\{\neg s\}$ subsumes $\{\neg s, u\}$.

$$A_2 = R(B_1) = \{\{\neg t\}, \{\neg u\}, \{p\}, \{r\}, \{\neg p, t, s\}, \{\neg q, t, s\}, \{\neg r, t, u\}, \{\neg r, s\}, \{\neg s\}, \{\neg p, s\}, \{\neg q, s\}, \{\neg r, u\}, \{\neg r, t\}, \{t, s\}, \{t, u\}, \{s\}, \{\neg p, t\}, \{\neg q, t\}, \{\neg r\}.$$

$$B_2 = RS(A_2) = \{\{\neg t\}, \{\neg u\}, \{p\}, \{r\}, \{\neg s\}, \{t, u\}, \{s\}, \{\neg p, t\}, \{\neg q, t\}, \{\neg r\}\}.$$

$$A_3 = R(B_2) = \{\ldots, \{\neg s\}, \ldots, \{s\}, \ldots, \bot, \ldots\}.$$

Thus, A is unsatisfiable and, therefore, the consequence is valid.

Other strategies that are followed to cut down waste in resolution base on the following two observations. Prove the observations and then employ them in a resolution algorithm.

Observation 1: Given a set of clauses A, if there is a literal l that occurs among the (some) clauses of A, but $\neg l$ never occurs, then the set of clauses A' obtained from A by deleting every clause that contains l is satisfiable iff A itself is satisfiable.

Observation 2: Given a set of clauses A, if A includes a **unit clause** (a clause having a single literal) $\{l\}$, then delete from A every clause that contains l to obtain the set of clauses A'. Next, update each clause of A' by deleting the occurrence of $\neg l$ from every clause (wherever $\neg l$ occurs) and call the new set of clauses as A''. Then A'' is satisfiable iff A is satisfiable.

For the cnf in Example 3.3, resolution along with subsumption gives

$$A_0 = A = \{\{\neg p, q\}, \{p, q\}, \{\neg p, \neg q\}\}, \qquad B_0 = RS(A_0) = A_0,$$
$$A_1 = A_0 \cup \{\{q\}, \{\neg p\}, \{q, \neg q\}, \{p, \neg p\}\}, \; B_1 = RS(A_1) = \{\{q\}, \{\neg p\}\}.$$

Further resolvents are not possible; so, $RS^*(A) = B_1$. Of course, A is satisfiable as you have already concluded in Example 3.3.

***Exercise* 3.5** Write a procedure for resolution employing subsumption as described in Example 3.4.

Note that the clauses in $RS^*(A)$ have a specific property, i.e., each one of them is a logical consequence of A but no one is a logical consequence of any other. Moreover, A is logically equivalent to $RS^*(A)$. Such clauses are called the prime implicates of a cnf.

Formally, let A be a cnf and D be a disjunctive clause. D is an **implicate** of A iff $A \models D$. D is a **prime implicate** of A iff D is an implicate of A and there is no other implicate C of A such that $C \models D$. That is, no other implicate comes in between A and D with respect to the consequence relation.

By way of trying to make resolution efficient, we have landed in computing all prime implicates of a cnf. Prime implicates help in minimizing Boolean circuits also. For more information on prime implicates, look at the works mentioned in the summary at the end of the chapter. Denote by $\pi(A)$ the set of all prime implicates of A and interpret this set as the conjunction of all its clauses as usual. Then try the following exercise.

***Exercise* 3.6** For a cnf A, show that $A \equiv \pi(A)$ and $RS^*(A) = \pi(A)$.

3.3 Unification of Clauses

You have seen how the resolution works for deciding satisfiability of a finite set of clauses. Can the procedure be applied to FL? Take a formula of FL. In order that resolution be applied, first of all, we require a cnf or some analogous conversion of the formula. We have, of course, the skolem conjunctive normal form conversion which says that a formula is satisfiable iff its scnf is satisfiable. Can we apply the resolution straightforward on the scnf? For example, consider the scnf

$$A = \{\{\neg Px, Qxy\}, \{Px\}, \{\neg Qxy\}\}$$

A resolution refutation would look something like the tree in Figure 3.3. But this may not be enough. For example, take

$$B = \{\{\neg Hx, Mx\}, \{Ha\}, \{\neg Ma\}\}$$

The scnf B corresponds to the consequence $\{\forall x(Hx \to Mx), Ha\} \models Ma$. We know that B is unsatisfiable. But how do we proceed in resolution? It looks as though we should be able to resolve $\{\neg Hx, Mx\}$ and $\{Ha\}$ to get $\{Ma\}$, and then $\bot$ may be derived as in Figure 3.4.

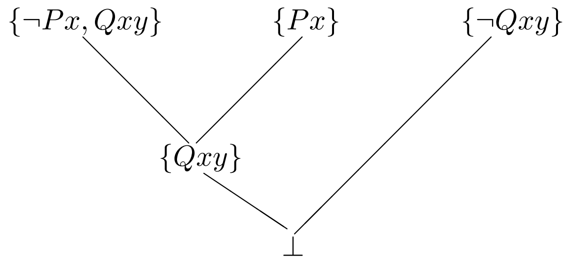


Figure 3.3

That is, somehow $\neg Hx$ and Ha have to be resolved to get Ma from the clauses $\{\neg Hx, Mx\}$ and $\{Ha\}$. It is clear that when we have $\{\neg Hx, Mx\}$, we have indeed the formula (as per convention in scnf) $\forall x(\neg Hx \vee Mx)$. From this, by specification, we surely have $\neg Ha \vee Ma$. Hence we should be able to extend our definition of resolvents in such a way that the substitution $[x/a]$ can be used to effect the specification. But how do we choose this particular substitution among many others. Moreover, if more than one variable are involved (as x above), then we may require a substitution such as $[x/a, y/b, z/c, \ldots]$ for simultaneously substituting the constants in place of the variables. Again, taking one substitution may work for a single step and you may require a different substitution for another step in the process of resolution. When you take stock, you realize the need to define the composition of all such substitutions. In this section, we will try to formally describe such notions and mechanize the choice of an appropriate substitution. Note that this corresponds to choosing the right specification of a universal quantifier, and serially doing so.

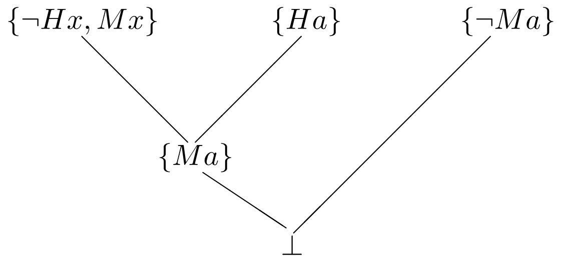


Figure 3.4

A **substitution** is a finite set of expressions of the form x_i/t_i, where x_i is a variable and t_i is a term. To distinguish between a substitution and a set, we will write a substitution by enclosing these expressions within a matching pair of square brackets instead of the braces. That is, a substitution is an expression of the form $[x_1/t_1, x_2/t_2, \ldots x_n/t_n]$, where each $x_1, \ldots, x_n$ are variables and $t_1, \ldots, t_n$ are terms, $x_i \neq x_j$ for $i \neq j$, and $x_i \neq t_i$. The fractions x_i/t_i need not be written in the same order always; both $[x_1/t_1, x_{11}/t_{11}]$ and $[x_{19}/t_5, x_5/t_{21}]$ are substitutions as long as x's are variables and t's are terms. But $[x/c, x/f(a)]$ and $[x/c, y/y]$ are not substitutions.

If $\sigma = [x_1/t_1, \ldots, x_n/t_n]$ is a substitution and t is a term, then the result $t\sigma$, of applying the substitution σ on the term t is a term in which each occurrence of each of the variables x_i in t is replaced by the corresponding term t_i simultaneously. For example,

$$f(a)[x/b] = f(a)$$

$$g(f(a), x)[x/f(b)] = g(f(a), f(b))$$

$$g(f(x), y)[x/f(y), y/a] = g(f(f(y)), a)$$

$$g(f(x), y)[x/f(y)][y/a] = g(f(f(y)), y)[y/a] = g(f(f(a)), a)$$

You can also define $t\sigma$ recursively using the grammar of terms. If X is a formula and $\sigma = [x_1/t_1, x_2/t_2, \ldots, x_n/t_n]$ is a substitution, then $X\sigma$ is the result of simultaneously replacing each free occurrence of each variable x_i by t_i in X. For example, if

$$X = \forall x(Pxy \to Qx) \wedge Rxy,\ \sigma = [x/a],\ \theta = [x/a, y/b, z/c]$$

then

$$X\sigma = \forall x(Pxy \to Qx) \wedge Ray$$

$$X\theta = \forall x(Pxb \to Qx) \wedge Rab$$

$$(X\theta)\sigma = (\forall x(Pxb \to Qx) \wedge Rab)\sigma = \forall x(Pxb \to Qx) \wedge Rab$$

$$(X\sigma)\theta = (\forall x(Pxy \to Qx) \wedge Ray)\theta = \forall x(Pxb \to Qx) \wedge Rab$$

Let $\sigma = [x_1/s_1, x_2/s_2, \ldots, x_m/s_m]$ and $\theta = [y_1/t_1, y_2/t_2, \ldots, y_n/t_n]$ be two substitutions. Their **composition** $\sigma \circ \theta$ is again a substitution obtained in the following way: Consider the set (written with square brackets here)

$$[x_1/s_1\theta,\ x_2/s_2\theta, \ldots, x_m/s_m\theta,\ y_1/t_1,\ y_2/t_2, \ldots, y_n/t_n]$$

Delete all elements of the form y_i/t_i from this set if $y_i \in \{x_1, \ldots, x_m\}$. Next, delete all elements of the form $x_j/s_j\,\theta$ if $s_j\,\theta = x_j$. The remaining set is $\sigma \circ \theta$.

For example, to compute the composition $[x/f(y)] \circ [z/y]$, we begin with the set $[x/f(y)[z/y],\ z/y]$, which is equal to $[x/f(y), z/y]$. There is nothing to delete thus, $[x/f(y)] \circ [z/y] = [x/f(y), z/y]$.

As another example, take

$$\sigma = [x/f(y), y/z],\ \theta = [x/a, y/b, z/c]$$

For $\sigma \circ \theta$, we form the set

$$[x/f(y)\theta, y/z\theta, x/a, y/b, z/c]$$

Since $f(y)\theta = f(b), z\theta = c$, the set is rewritten as

$$[x/f(b), y/c, x/a, y/b, z/c]$$

Delete x/a, y/b as the variables x, y already appear in other replacements (due to replacements of the form $x_j/s_j\,\theta$). The updated set is

$$[x/f(b), y/c, z/c]$$

Now, there is nothing to delete, and hence

$$\sigma \circ \theta = [x/f(b), y/c, z/c]$$

What is $\theta \circ \sigma$? Well, we form the set

$$[x/a\sigma, y/b\sigma, z/c\sigma, x/f(y), y/z] = [x/a, y/b, z/c, x/f(y), y/z]$$

computing with σ. The last two elements are deleted to obtain

$$\theta \circ \sigma = [x/a, y/b, z/c]$$

Now you see that $\sigma \circ \theta \neq \theta \circ \sigma$; order does matter in taking compositions.

If $\delta = [x/a, y/b, z/y]$, then for the composition $\sigma \circ \delta$, we form the set $[x/f(b), y/y, x/a, y/b, z/y]$. From this we first delete x/a and y/b, then from the updated set, we delete y/y to get $\sigma \circ \delta = [x/f(b), z/y]$. Note that the deletions have to be done in the specified order, else you may land up at a different substitution.

***Exercise* 3.7** Show that the composition of substitutions is associative, i.e., for any substitutions σ, θ, δ, we have $(\sigma \circ \theta) \circ \delta = \sigma \circ (\theta \circ \delta)$.

For $X = Pxyz$, what are $X(\sigma \circ \delta)$ and $(X\sigma)\delta$? We obtain

$$X\sigma = Pxyz[x/f(y), y/z] = Pf(y)zz$$

$$(X\sigma)\delta = Pf(b)yy, \; X(\sigma \circ \delta) = Pxyz[x/f(b), z/y] = Pf(b)yy$$

They are equal! Can you prove it for any formula or any term, and any pair of substitutions σ and δ?

Lemma 3.5 *Let θ and δ be two substitutions, t be any term, and X be any formula. Then, $t(\theta \circ \delta) = (t\theta)\delta$ and $X(\theta \circ \delta) = (X\theta)\delta$.*

Proof Use induction on the structure of terms and formulas. For terms, use induction on the number of occurrences of function symbols and for formulas, on the number of occurrences of connectives and quantifiers. ∎

***Exercise* 3.8** Find $\sigma \circ \theta$ in each of the following cases:

(a) $\sigma = [y/x, z/f(y)], \; \theta = [x/a, y/x, z/f(a)]$

(b) $\sigma = [x/a, y/c, z/x], \; \theta = [y/z, x/a]$

(c) $\sigma = [x/a, y/a, z/g(x)], \; \theta = [x/a, y/f(z, a), z/y]$

Substitutions are applied on sets of formulas as well. That is, for a set Σ of formulas, and a substitution θ, we have $\Sigma\,\theta = \{X\theta : X \in \Sigma\}$, i.e., θ is applied on every formula in Σ and then the formulas are collected together to have $\Sigma\,\theta$.

Renaming of variables in a formula can be seen as an application of a substitution. These special substitutions are called variants. A **variant** is a substitution of the form $[x_1/y_1, x_2/y_2, \ldots, x_m/y_m]$, where x_i's are distinct and y_i's are distinct, and no y_j is an x_i. We need to use variants in such a way that two clauses will have different variables after renaming. If B and C are two clauses, then two substitutions σ and θ are called **a pair of separating variants** iff $B\sigma$ and $C\theta$ have no common variables. Renaming is done in such a way that they do not coalesce on any variable.

Recollect that our plan was to choose substitutions in resolving clauses mechanically. We should have a procedure to choose the substitution $[x/a]$ while resolving the clauses $\{\neg Hx, Mx\}$ and $\{Ha\}$. This is done by a unifier, which is a substitution that makes both $\neg Hx$ and $\neg Ha$ the same clause.

Let $A = \{A_1, A_2, \ldots, A_m\}$ be a clause where each A_i is a literal. A substitution σ is a **unifier** of A iff $A_1\sigma = A_2\sigma = \cdots = A_m\sigma$. That is, the set $A\sigma$ is a singleton. For example, $\{\neg Hx, \neg Ha\}$ has a unifier $[x/a]$ as $\{\neg Hx, \neg Ha\}\sigma = \{\neg Ha\}$, a singleton. But $\{\neg Hx, Ha\}$ has no unifier. We say that a clause is **unifiable** iff there is a unifier of it. For example, the clause $A = \{Pxyf(g(z)), Puf(u)f(v)\}$ is unifiable since $A\sigma = \{Paf(a)f(g(a))\}$. Here, $\sigma = [x/a, y/f(a), u/a, z/a, v/g(a)]$ is the unifier. We also see that the substitutions $\theta = [x/u, y/f(u), v/g(z)]$ and $\delta = [x/u, y/f(u), z/a, v/g(a)]$ also unify A as $A\theta = \{Puf(u)f(g(z))\}$ and $A\delta = \{Puf(u)f(g(a))\}$. Thus, a unifier is not necessarily unique.

Look at $A\sigma, A\theta, A\delta$ closely. It is easy to guess that $\sigma = \theta[u/a, z/a]$ and $\delta = \theta[z/a]$. You can also see separately that these compositions are correct. The point to note here is that if you take θ as your unifier, then later, you can still choose another substitution to get back the effect of σ or of δ by a composition. A substitution such as θ here is called a most general unifier. A unifier θ of a clause A is called a **most general unifier** of A if for every other unifier δ of A, there exists a substitution σ such that $\delta = \theta\sigma$. Similarly, a most general unifier of a set of terms is also defined. We are, in fact, interested in a most general unifier since we do not want to restrict our universal specification in such a way that we lose information.

However, a most general unifier of a clause (or of a set of terms) need not be unique. For example, take $A = \{Pxyf(g(z)), Puf(u)f(v)\}$. We have already a most general unifier $\theta = [x/a, y/f(a), v/g(z)]$. Yet another most general unifier of A is $\lambda = [u/x, y/f(x), v/g(z)]$. You can see, of course, that $\theta = \lambda[x/u]$ and $\lambda = \theta[u/x]$. But there is a way out. It looks that the most general unifiers can be obtained from other most general unifiers by applying variants. In that case, we will not lose any information by using any one of the possibly many most general unifiers.

***Exercise* 3.9** See that the clause $\{Pxyf(g(z)), Puf(u)f(v)\}$ has most general unifiers $\theta = [x/a, y/f(a), v/g(z)]$ and $\lambda = [u/x, y/f(x), v/g(z)]$. Does it have any other most general unifier?

The problem is, how to mechanically construct a most general unifier? We will try to write a procedure by simulating the way we do unification manually. To this end, let us see how to unify the clause

$$A = \{Pxyf(g(z)), Puf(u)f(v)\}$$

Your target is to construct a substitution so that both the literals will become the same after the substitution is effected. You will naturally be scanning the clauses, say, from left to right, symbol by symbol. The first symbol P matches. A discrepancy occurs at the second symbol: one is x and the other is u. So, start with a substitution $[x/u]$. Then,

$$A[x/u] = \{Puyf(g(z)), Puf(u)f(v)\}$$

Matching the literals in this clause, you find that there is a discrepancy at the third symbol. One is y, a variable, and the other is a term $f(u)$. So, form the substitution $[y/f(u)]$. Applying this, you get

$$(A[x/u])[y/f(u)] = \{Puf(u)f(g(z)), Puf(u)f(v)\}$$

Next discrepancy suggests the substitution $[v/g(z)]$, applying which you have the unified clause

$$((A[x/u])[y/f(u)])[v/g(z)] = \{Puf(u)f(g(z))\}$$

Since the compositions of the substitutions are associative, Lemma 3.5 gives you the unifier

$$\sigma = [x/u] \circ [y/f(u)] \circ [v/g(z)] = [x/u, y/f(u), v/g(z)]$$

The unification procedure described below does just this. We write the procedure for a set of literals; it is rewritten for a set of terms without any discomfort. Do this immediately after you understand the procedure. We will refer to both the procedures as the *unification algorithm.* Henceforth, we write the composition $\sigma \circ \theta$ simply as $\sigma\theta$.

```
PROCEDURE : Unification
    Input: A clause $A$ (or a set of terms $A$)
    Output: A most general unifier of $A$ if one exists; else, '$A$ is not unifiable'
    1. If a literal in $A$ starts with $\neg$ and another does not, then output
        '$A$ is not unifiable'.
    2. If all elements of $A$ do not use the same predicate symbol, then
        output '$A$ is not unifiable'.
    3. $A_0 := A$; $\sigma_0 := [\,]$, the empty substitution; $\theta_0 = \sigma_0$, $k := 0$.
    4. If $A_k$ is singleton, then output $\theta_k$.
    5. Scan the first two elements of $A_k$ to find a mismatch.
    6. If the mismatch is due to different function symbols, then output
        '$A$ is not unifiable'.
    7. If the mismatch is due to a variable $x$ in one and a term $t$ in
        the other, then if $x$ occurs in $t$, then output '$A$ is not unifiable',
        else, $\sigma_{k+1} := [x/t], \theta_{k+1} := \theta_k\sigma_{k+1}$.
    8. $A_{k+1} := A_k\sigma_{k+1}$; $k := k + 1$; go to Step 4.
```

In Step 7, if t is also a variable, then the variable of the first literal is taken as x and the variable in the second is taken as t, to break the nondeterminism in the above procedure. Note that the unified clause is $A\theta_{k+1}$, where the most general unifier is $\theta_{k+1} = \sigma_1\sigma_2 \cdots \sigma_{k+1}$. To fix the idea, see the following examples and solve the exercises.

***EXAMPLE* 3.5** Use *Unification* on $A = \{Pxf(y), Pf(y)f(x)\}$.

Solution The first mismatch is at x in the first literal and $f(y)$ in the second. Thus,

$$\theta_1 = \sigma_1 = [x/f(y)], \text{ and } A_1 = A\sigma_1 = \{Pf(y)f(y), Pf(y)f(f(y))\}$$

The next mismatch is at y in the first literal and $f(y)$ in the second. Since y occurs in $f(y)$, the clause is not unifiable (Step 7).

***Exercise* 3.10** Show that $\theta = [x/a, y/g(f(a)), z/f(a), u/f(f(a))]$ is a unifier of $A = \{Pxyf(z), Pag(z)u\}$. Find a most general unifier σ of A and also a nonempty substitution δ such that $\theta = \sigma\delta$.

***EXAMPLE* 3.6** Unify $A = \{P(x, y, g(x, z, f(y))), P(z, f(x), u)\}$.

Solution

$$\begin{aligned}
&\sigma_1 = [x/z],\ \theta_1 = \sigma_1 \\
&A_1 = A\sigma_1 = \{P(z, y, g(z, z, f(y))), P(z, f(z), u)\} \\
&\sigma_2 = [y/f(z)],\ \theta_2 = \sigma_1\sigma_2 \\
&A_2 = A_1\sigma_2 = \{P(z, f(z), g(z, z, f(f(z)))), P(z, f(z), u)\} \\
&\sigma_3 = [u/g(z, z, f(f(z))],\ \theta_3 = \theta_2\sigma_3 \\
&A_3 = A_2\sigma_3 = \{P(z, y, g(z, z, f(y)))\}
\end{aligned}$$

Since A_3 is a singleton, the most general unifier is

$$\theta_3 = [x/z][y/f(z)][u/g(z, z, f(f(z))] = [x/z, y/f(z), u/g(z, z, f(f(z))].$$

***Exercise* 3.11** Find the most general unifiers, if they exist, for each of the following clauses:

(a) $\{Pxf(x)g(f(y)), Pcf(g(z))g(y)\}$

(b) $\{P(x, f(x, g(x), y), f(x, y, z)), P(c, g(y), g(z))\}$

(c) $\{P(f(c), x, y), P(z, f(c), y), P(z, x, f(c))\}$

Henceforth, we will use the term '*the* most general unifier' or the acronym **mgu** for the one computed by the *Unification* algorithm whenever a clause or a term is indeed unifiable.

***Exercise* 3.12** Show that the procedure *Unification* computes correctly a most general unifier, if one such exists.
[*Hint*: Use induction on the number of loops the procedure executes with any input clause. Look at k.]

3.4 Extending Resolution

To see how mgu's can be used in the resolution method, consider the clause $\{\neg Hx \vee Mx, Ha, \neg Ma\}$. The first clause is $C_1 = \{\neg Hx, Mx\}$, and the second is $C_2 = \{Ha\}$. The mgu of $\neg Hx$ and $\neg Ha$ (note the extra $\neg$ with Ha) is $[x/a]$. Now, $C_1[x/a] = \{\neg Ha, Ma\}, C_2[x/a] = \{Ha\}$. Taking the resolvent, we get $\{Ma\}$. Intentionally, we had taken $C_2[x/a]$ to keep generality. Look at one more example.

Suppose that $A = \{\neg Hxy \vee Mxy, Hya\}$. What is the most general unifier of $\{\neg Hxy, \neg Hya\}$? (Look at the additional $\neg$ again.) The mgu is $\sigma = [x/y][y/a] = [x/a, y/a]$. Now, $A\sigma = \{\neg Haa \vee Maa, Haa\}$ from which resolution gives Maa. Originally, A represents the formula $\forall x \forall y(Hxy \to Mxy) \wedge \forall y Hya$, which logically implies (entails) $\forall y Mya$. However, we have only Maa. What was wrong? Why did we lose information? The reason is, when the substitution $[y/a]$ was applied after $[x/y]$, the variable x had to be replaced by a. In general, this is not required. In the formula $\forall x \forall y(Hxy \to Mxy) \wedge \forall y Hya$, the variable y that occurs in $\forall y Hya$ could have been renamed. Suppose we do that now. Then the formula looks like (is $\equiv$ to): $\forall x \forall y(Hxy \to Mxy) \wedge \forall z Hza$. The clause set representation of this formula is $B = \{\neg Hxy \vee Mxy, Hza\}$. Now, the mgu of $\{\neg Hxy, \neg Hza\}$ is $[x/z]$ which, when applied on B, gives $\{\neg Hza \vee Mza, Hza\}$. And the resolvent of the clauses in B is Mza, which keeps the generality intact. The lesson is

> There should not be common variables in clauses; then only resolution would work as expected.

So, you find the usefulness of separating variants! If the clauses in a clause set have common variables, then we use separating variants to have distinct variables in different clauses. Since this can always be done, we assume, without loss of generality, that no pair of clauses in a set have any common variables. The resolvent of clauses can then be defined as follows:

Definition 3.1 Let C_1 and C_2 be two clauses with no common variables, and $l_1 \in C_1, l_2 \in C_2$ be two literals such that σ is the most general unifier of l_1 and $\neg l_2$. Then $res(C_1, C_2; l_1) = ((C_1 - \{l_1\}) \cup (C_2 - \{l_2\}))\sigma$ is the **resolvent** of the clauses C_1 and C_2 with respect to the literal l_1 (or l_2). Clauses C_1 and C_2 are called the **parent clauses** of the resolvent clause $res(C_1, C_2; l_1)$.

Sometimes, we will write $res(C_1, C_2; l_1)$ as $res(C_1, C_2; l_1, \sigma)$ to explicitly mention the mgu σ in the resolvent. Sometimes, we will omit the literal and the mgu altogether and write it as $res(C_1, C_2)$.

***EXAMPLE* 3.7** Find all possible resolvents of the clauses:

$A = \{Pxf(y), Qg(y), Rxzb\}$ and $B = \{\neg Pxf(b), \neg Qz, Rxab\}$

Solution The clauses A and B have common variables. We first use separating variants. When implemented in a machine, the separating variants will be something like $\delta_1 = [x/x_1, y/x_2, z/x_3]$ and $\delta_2 = [x/x_4, z/x_5]$ so that instead of A, B, we will be considering the clauses as $A\delta_1, B\delta_2$, respectively. While doing manually, we will keep one of the clauses as it is and rename the common variables in the second clause. Here, we take $\delta = [x/u, z/v]$ and rename B to have the clause $B_1 = B\delta = \{\neg Puf(b), \neg Qv, Ruab\}$.

Now, with A and B_1, we have a literal $Pxf(y)$ in A and $\neg Puf(b)$ in B. The mgu of first and $\neg$ of the other is to be computed. Since the literal $\neg\neg Puf(b)$ is taken as $Puf(b)$, we compute the most general unifier of $\{Pxf(y), Puf(b)\}$. It is $\sigma = [x/a, y/b]$, and the resolvent is

$$\begin{aligned} res(A, B_1; Pxf(y)) &= ((A - \{Pxf(y)\}) \cup (B_1 - \{Puf(b)\}))[x/u, y/b] \\ &= \{Qg(b), Ruzb, \neg Qv, Ruab\}. \end{aligned}$$

Similarly, choosing the literals $Qg(y)$ from A and $\neg Qv$ from B_1, we have the mgu $[v/g(u)]$ for the set $\{Qg(y), Qv\}$. This gives

$$res(A, B_1, Qg(y)) = \{Pxf(y), Rxzb, \neg Puf(b), Ruab\}$$

Choosing $Rxzb$ from A, you find that the corresponding literal from B should be $Ruab$. But the set $\{Rxzb, \neg Ruab\}$ is not unifiable, and so it gives no resolvent.

As in PL, we would expect the resolvent to be a logical consequence of its parent clauses. In the above example, we would have $\{A, B_1\} \models C$, where $C = \{Qg(b), Ruzb, \neg Qv, Ruab\}$. Is it so?

Let $I = (D, \phi)$ be an interpretation and m be a valuation under I. Suppose that the state $I_m \models A$ and $I_m \models B_1$. That is,

$$I_m \models Pxf(y) \vee Qg(y) \vee Rxzb, \;\; I_m \models \neg Puf(b) \vee \neg Qv \vee Ruab$$

Now, if $I_m \not\models \neg Puf(b)$, then $I_m \models \neg Qv \vee Ruab$. In this case, $I_m \models C$. On the other hand, if $I_m \models \neg Puf(b)$, then as u is universally quantified in $\neg Puf(b)$, $I_m \not\models Pxf(y)$. Thus, $I_m \models Qg(y) \vee Rxzb$.

Since x and y are universally quantified, $I_m \models Qg(b) \vee Ruzb$. Consequently, $I_m \models C$. This shows that in any case, $I_m \models C$.

Theorem 3.6 (Resolution Principle) *Let A and B be two first order clauses having a resolvent as a clause C. Then, $\{A, B\} \models C$.*

Proof Repeat the argument discussed for a particular case above. ∎

So, you can proceed as in PL, to see whether $\perp$ is generated from a given set of clauses, by way of taking resolvents. If at all $\perp$ is generated, then the set of clauses is unsatisfiable.

***EXAMPLE* 3.8** Apply the resolution principle to show that the set of clauses $A = \{\{\neg Px, Qx\}, \{Pa\}, \{Rba\}, \{\neg Qy, \neg Rby\}\}$ is unsatisfiable.

Solution We try to take resolvents of clauses from A, their resolvents, and so on with a goal to get $\perp$. Here is one such trial.

1. $\{\neg Px, Qx\}$	H
2. $\{Pa\}$	H
3. $\{Qa\}$	$res(1,2; \neg Px, [x/a])$
4. $\{\neg Qy, \neg Rby\}$	H
5. $\{\neg Rba\}$	$res(3,4; Qa, [y/a])$
6. $\{Rba\}$	H
7. $\bot$	$res(5,6; \neg Rba, [\ ])$

The refutation can be seen as a DAG also; see Figure 3.5.

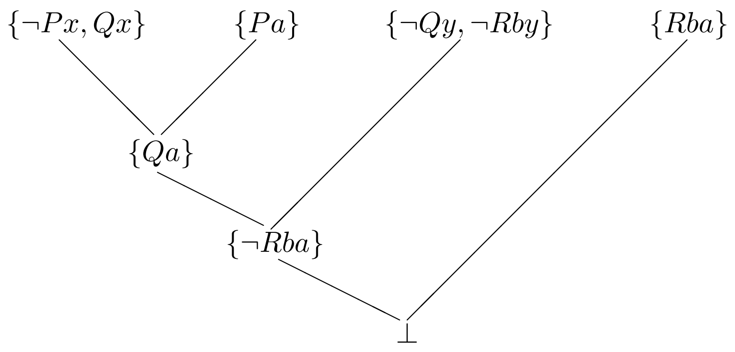


Figure 3.5

***EXAMPLE* 3.9** Show by resolution that $A = \{\{Px, Py\}, \{\neg Pu, \neg Pz\}\}$ is unsatisfiable.

Solution Let us try a resolution deduction of $\bot$ from the set of clauses.

1. $\{Px, Py\}$	H
2. $\{\neg Pu, \neg Pz\}$	H
3. $\{Py, \neg Pz\}$	$res(1,2; Px, [x/u])$
4. $\{\neg Pz, \neg Pv\}$	$res(2,3; \neg Pu, [u/y])$

Why does $\{\neg Pz, \neg Pv\}$ come as the resolvent of clauses 2 and 3; why not $\{\neg Pz\}$? Because, resolution is taken only after it is guaranteed that the clauses have no common variables. Thus, (2) is kept as it is, whereas (3) is first rewritten as $\{Py, \neg Pv\}$ by using a renaming substitution $[z/v]$ (in fact, the pair of variants $[\ ]$ and $[z/v]$). In a machine implementation, lines (2) and (3) will appear as

$(2')$ $\{\neg Px_3, \neg Px_4\}$

$(3')$ $\{Px_5, \neg Px_6\}$

whose resolvent $\{\neg Px_4, \neg Px_6\}$ will again be rewritten as

$(4')$ $\{\neg Px_7, \neg Px_8\}$

Do you see? We are in trouble! It is clear now that whenever we apply resolution, there will be a clause in one of the following forms:

$$\{Px_i, Px_j\}, \{\neg Px_i, Px_j\}, \{\neg Px_i, \neg Px_j\}$$

And this process will never give us $\bot$.

Well, that is quite possible, the author of this book may be wrong in asking you to show that the set A in Example 3.9 is unsatisfiable. But is A really unsatisfiable? As an FL-formula,

$$A \equiv \forall x \forall y(Px \vee Py) \wedge \forall u \forall z(\neg Pu \vee \neg Pz)$$

Since $\forall x \forall y(Px \vee Py) \equiv \forall x Px$ and $\forall u \forall z(\neg Pu \vee \neg Pz) \equiv \forall y(\neg Py)$, we have

$$A \equiv \forall x Px \wedge \forall y(\neg Py)$$

This is clearly unsatisfiable.

So, what is wrong? The Resolution Principle says that whatever had been deduced in the above derivation is a consequence of the clauses. But this does not say that every unsatisfiable set will eventually yield $\bot$. Perhaps, resolution is not a complete method in FL. We require an extra rule. The above discussion gives us a hint. We need a rule to capture the equivalence $\forall x \forall y(Px \vee Py) \equiv \forall x Px$. That is, we must be able to deduce $\{Pz\}$ from $\{Px, Py\}$. Such a clause $\{Pz\}$ is called a factor of the clause $\{Px, Py\}$. We give a formal definition.

Definition 3.2 Let C be a clause and $D \subseteq C$, with D having at least two literals. Let σ be a (the) most general unifier of D. Then $C\sigma$ is called a **factor** of C. A factor of C with respect to the clause D and mgu σ is written as $fac(C; D, \sigma)$ or as $fac(C, \sigma)$ if D is obvious from the context.

We have used an mgu σ, because from $\{Px, Py\}$, we want to deduce $\{Pz\}$, and not just $\{Pa\}$. Along with resolvents we will also use factors.

Example 3.9 Redone:

1. $\{Px, Py\}$	H
2. $\{Py\}$	$fac(1, [u/z])$
3. $\{\neg Pu, \neg Pz\}$	H
4. $\{\neg Pz\}$	$fac(3, [u/z])$
5. $\bot$	$res(2, 4; Py, [y/z])$

EXAMPLE **3.10** Show that the set $C = \{\{a \approx b\}, \{Px\}, \{\neg Pb\}\}$ is unsatisfiable by using resolvents and factors.

A trial solution C is clearly unsatisfiable. But how to deduce $\bot$? There is no variables at all. So, taking factors is useless since $\{a \approx b\}$ and $\{Pa\}$ would not give $\{Pb\}$. We are stuck. Reason? We do not know yet how to handle the special predicate $\approx$. We must be able to deduce Pb given the literals $a \approx b$ and Pa. In general, if we have $X[x/s]$ and $s \approx t$, then we must be able to deduce $X[x/t]$. More generally, from $t \approx u$ and $X[x/s]$, with σ being the mgu of s and t, we must deduce $X[x/u\sigma]$. Here we need unification of terms also. The following definition captures the idea. We will use the notation $X(s)$ for $X[x := s]$ as a shorthand. Note that $X[x/s]$ is the formula obtained from X by replacing all free occurrences of the variable x in X by the term s, whereas $X(s)$ is a formula obtained from X by replacing some (not necessarily all) free occurrences of x in X by s. Thus, from $X(s)$ you get $X(t)$ by replacing *some of the occurrences* of s in X by t.

Definition 3.3 Let X be a formula, s, t, u be terms. Let A and B be two clauses such that $(t \approx u) \in A, X(s) \in B$, and that A, B have no common variables. Let σ be a most general unifier of the terms s and t (s, t have no common variables). Then the **paramodulant** of A and B is the clause $((A - \{t \approx u\}) \cup (B - \{X(s)\}) \cup \{X(u)\})\sigma$.

The operation of taking a paramodulant is clear: delete $(s \approx t)$ from A, delete $X(u)$ from B, and add $X(t)$ to all the remaining literals; the only restriction is that apply the mgu σ all the time. Adding a paramodulant is often referred to as paramodulation. We use factors and paramodulants in solving Example 3.10.

Solution to Example 3.10 For $C = \{\{a \approx b\}, \{Px\}, \{\neg Pb\}\}$, let $A = \{a \approx b\}$, and $B = \{Px\}$. As in Definition 3.3, we have

$$t = a, u = b, s = x, X(s) = Px, A - \{t \approx u\} = B - \{X(s)\} = \emptyset$$

The mgu of s, t (in fact of $\{s, t\}$) is $\sigma = [x/b]$. Hence the paramodulant is

$$(\emptyset \cup \emptyset \cup X(u))\sigma = Pb[x/b] = Pb$$

Now, complete the solution.

***EXAMPLE* 3.11** What is the paramodulant of $\{a \approx b\}, \{Pa\}$?

Solution With $A = \{a \approx b\}, B = \{Pa\}, s = a, t = a$, we have $\sigma = [\;]$ and $u = b$. Since $X(s) = Pa$, the paramodulant is $X(u)\sigma = Pb$.

***EXAMPLE* 3.12** What is the paramodulant of the clauses

$$\{f(f(a, c), h(c)) \approx f(f(b, c), h(c))\} \text{ and } \{f(x, f(y, z)) \approx f(f(x, y), z)\}?$$

Solution Since two equalities are involved, you can choose one of them as $(t \approx u)$ and the other as $X(s)$. Let us take the first as $(t \approx u)$. Then $t = f(f(a, c), h(c)),\ u = f(f(b, c), h(c))$.

With $s = f(x, f(y, z))$, we find that s and t are not unifiable. But with $s = f(f(x, y), z)$, s and t are unifiable and the mgu is $\sigma = [x/a, y/c, z/h(c)]$. Then,

$$X(s) = (f(x, f(y, z)) \approx s),\ X(t) = (f(x, f(y, z)) \approx f(f(b, c), h(c)))$$

The paramodulant is

$$X(u)\sigma = (f(a, f(c, h(c))) \approx f(f(b, c), h(c)))$$

There is one more way of computing a paramodulant here; that is, by taking the second clause as $t \approx u$. In that case, you have

$$t = f(f(x,y),z),\ u = f(x,f(y,z)),\ s = f(f(a,c),h(c))$$
$$X(s) = (s \approx f(f(b,c),h(c)))$$

And then, $\sigma = [x/a, y/c, z/h(c)]$, and the paramodulant is

$$X(u)\sigma = (u \approx f(f(b,c),h(c)))\sigma = (f(a,f(c,h(c))) \approx f(f(b,c),h(c)))$$

Something curious happens. Can you justify that $\{\neg(t \approx t)\}$ is unsatisfiable, using all the three rules of resolution, factor and paramodulant? Definitely not. Each of them requires more than one clause, and we have only one here. To accommodate this, we have to take an extra rule for equality, namely,

for any term t, derive $(t \approx t)$ even when there is no premise.

That is, $(t \approx t)$ is added anywhere in a deduction. This simple looking rule supplements the factors also. To see this, consider the derivation of Pa from Px, which is simply the universal specification. Of course, you can show by resolution that $\{\{Px\}, \{\neg Pa\}\}$ is unsatisfiable. Then this will show that $Px \models Pa$. This indirect way can be circumvented by using the above rule of equality along with the paramodulant. Take the new clause $\{a \approx a\}$, then compute the paramodulant of this and $\{Px\}$. You have $t = a, u = a, s = x, X(s) = Px$, and the mgu of s and t as $\sigma = [x/a]$. The paramodulant is

$$((\{a \approx a\} - \{a \approx a\}) \cup (\{Px\} - \{Px\}) \cup \{X(s)\})\sigma = \{Px[x/a]\} = \{Pa\}$$

3.5 Resolution for FL

We rewrite the operations of taking resolution, factor, paramodulant and adding the equality clauses of the form $t \approx t$ as rules. All the rules taken together form the **method of resolution**. You can define what a resolution refutation (or deduction) means following the footprints of PL. Summing up, we have the following four rules for working out a resolution refutation or a resolution deduction:

(R) From two clauses, deduce their resolvents:

$$\frac{C_1 \vee l_1 \qquad C_2 \vee l_2}{(C_1 \vee C_2)\sigma} \quad (\sigma \text{ is the mgu of the literals } l_1 \text{ and } \neg l_2.)$$

(F) From a clause derive its factor(s):

$$\frac{C \vee l_1 \vee \cdots \vee l_k}{(C \vee l_1)\sigma} \quad (\sigma \text{ is the mgu of the literals } l_1 \vee l_2 \vee \cdots \vee l_k.)$$

(P) From two clauses, one of them being in the form $t \approx u$ or $u \approx t$, derive their paramodulants:

$$\frac{X(s) \qquad (t \approx u) \vee D}{(X(u) \vee D)\sigma} \qquad\qquad \frac{X(s) \qquad (u \approx t) \vee D}{(X(u) \vee D)\sigma}$$

(σ is the mgu of the terms s and t.)

(E) For any term t, derive $(t \approx t)$ from any clause:

$$\frac{\cdot}{(t \approx t)}$$

To show that a set of clauses as unsatisfiable, we only derive $\perp$ from it. Let us work out some examples.

***EXAMPLE* 3.13** Use resolution to show that the following argument is valid: No student reads a boring book seriously. This book is read seriously by at least one student (you). Therefore, this book is not boring.

Solution We fix a vocabulary for translation into FL, say, Sx: x is a student, b: this book, Rxy: x reads y seriously, and Bx: x is a boring book. The consequence is

$$\{\neg\exists x \exists y(Sx \wedge By \wedge Rxy), \exists x(Sx \wedge Rxb)\} \models \neg Bb$$

By RAA, the consequence is valid iff the set of formulas

$$\{\neg\exists x \exists y(Sx \wedge By \wedge Rxy), \exists x(Sx \wedge Rxb), Bb\}$$

is unsatisfiable. To apply (the method of) resolution, we first rewrite this set as a set of clauses.

By skolemization, $\exists x(Sx \wedge Rxb)$ gives $Sa \wedge Rab$. The other formula

$$\neg\exists x \exists y(Sx \wedge By \wedge Rxy)$$

is equivalent to

$$\forall x \forall y(\neg Sx \vee \neg By \vee \neg Rxy)$$

This gives $\neg Sx \vee \neg By \vee \neg Rxy$. Hence the clause set is

$$\{\{\neg Sx, \neg By, \neg Rxy\}, \{Sa\}, \{Rab\}, \{Bb\}\}$$

We try a resolution deduction of $\perp$ applying possibly all the four rules as mentioned above.

1.	$\{\neg Sx, \neg By, \neg Rxy\}$	H
2.	$\{Sa\}$	H
3.	$\{\neg By, \neg Ray\}$	1,2,R
4.	$\{Bb\}$	H
5.	$\{\neg Rab\}$	3,4,R
6.	$\{Rab\}$	H
7.	$\perp$	R

***EXAMPLE* 3.14** Show that $\exists x \forall y(Py \lor Px \lor \neg Qyx) \models \exists z(Pz \lor \neg Qzz)$.

Solution We must show that $\{\exists x \forall y(Px \lor Py \lor \neg Qyx), \neg \exists z(Pz \lor \neg Qzz)\}$ is unsatisfiable. The corresponding clause set is (verify)

$$\{\{Pa, Py, \neg Qya\}, \{\neg Pz\}, \{Qzz\}\}$$

The following is a resolution refutation:

1.	$\{Pa, Py, \neg Qya\}$	H
2.	$\{Pa, \neg Qaa\}$	F:$[y/a]$
3.	$\{\neg Pz\}$	H
4.	$\{\neg Qaa\}$	2,3,R
5.	$\{Qzz\}$	H
6.	$\bot$	4,5,R

***EXAMPLE* 3.15** Show by resolution that

$$\{\forall x(x \approx f(x,a)), \forall x \forall y \forall z(f(f(x,y),z) \approx f(x, f(y,z))),$$
$$\forall x(a \approx f(x, h(x)))\} \models \forall x \forall y \forall z((f(x,y) \approx f(x,z)) \to (y \approx z)).$$

If you interpret $f(x,y)$ as $x + y$, then the first premise says that a is the identity of addition, the second premise asserts the associativity of addition, and $h(x)$ becomes the additive inverse of x as the third premise says. You are then asked to show the cancellation law in a group.

Solution Changing into scnf and adding the negation of the conclusion to the set of premises, we require a refutation of the clause set

$$A = \{\{f(x,a) \approx x\}, \{a \approx f(x, h(x))\}, \{f(f(x,y),z) \approx f(x, f(y,z))\},$$
$$\{f(b,c) \approx f(d,c)\}, \{\neg(b \approx d)\}\}$$

where b, c, d are the skolem (new) constants. Here is a refutation; we will not write the outer braces in the clauses here for better reading. Enclose each literal in each line by a pair of matching braces for a formal refutation.

1.	$a \approx f(x, h(x))$	H
2.	$c \approx c$	E
3.	$a \approx f(c, h(c))$	$1, 2,$ P:$[x/c]$
4.	$f(b,c) \approx f(d,c)$	H
5.	$f(f(b,c), h(c)) \approx f(f(b,c), h(c))$	E
6.	$f(f(b,c), h(c)) \approx f(f(d,c), h(c))$	$4, 5,$ P:[ ]
7.	$f(f(x,y),z) \approx f(x, f(y,z))$	H
8.	$f(b, f(c, h(c))) \approx f(d, f(c, h(c)))$	$6, 7,$ P:$[x/b, y/c, z/h(c)]$
9.	$f(b,a) \approx f(d,a)$	H
10.	$f(x,a) \approx x$	H
11.	$b \approx f(d,a)$	$9, 10,$ P:$[x/b]$

12.	$b \approx d$	$10, 11, \mathrm{P}{:}x/d]$
13.	$\neg(b \approx d)$	H
14.	$\bot$	$13, 14, \mathrm{R}$

***Exercise* 3.13** Use resolution to determine whether the following formulas are satisfiable:

(a) $\forall y \exists x (Pyx \wedge (Qy \leftrightarrow \neg Qx)) \wedge \forall x \forall y \forall z ((Pxy \wedge Pyz) \to Pxz)$
$\wedge\, \forall x \neg \exists z (Qx \wedge Qz \wedge Pxz)$

(b) $\forall x \forall y \forall z ((Px \wedge Qy \wedge Rzy \wedge Syx) \to Rzx)$
$\wedge\, \neg \exists x \exists y \exists z (Px \wedge Py \wedge \neg(x \approx y) \wedge Rzx \wedge Rzy) \wedge Qa$
$\wedge\, Rba \wedge Sac \wedge Pc \wedge Pd \wedge \neg(c \approx d) \wedge Rsd \wedge (s \approx b)$

The *soundness* and *completeness* of resolution for FL can be proved as earlier. Soundness is accomplished by showing that for each rule having a numerator Σ and denominator w, the corresponding consequence $\Sigma \models w$ is valid. Though soundness is straightforward, completeness is quite involved. The central idea is the observation that due to Rules F, E and P, if $X(x)$ is in A, then $X(t)$ can be derived for any closed term t. (A closed term is a term having no variables.) Then by induction you will show that every finite subset of the Herbrand expansion of A can be derived by resolution. Then you will use the compactness theorem for PL, which guarantees that if a set of propositions is unsatisfiable, then a finite subset of it is also unsatisfiable (we have not proved it yet). Now since every finite set of the Herbrand expansion is derivable, so is this particular finite subset. Moreover, this finite subset of the Herbrand expansion is propositionally unsatisfiable. Due to the completeness of resolution (Rule R is used here) for PL, we conclude that $\bot$ is derivable from this finite subset. This proves the completeness of resolution for FL.

Again, with compactness for FL (not yet proved), which says that if a set of (FL-) formulas is unsatisfiable, then a finite subset of this set is unsatisfiable, we see that $\Sigma \cup \{\neg W\}$ is unsatisfiable iff for some finite subset Σ_0 of Σ, $\Sigma_0 \cup \{\neg W\}$ is unsatisfiable. (Without loss of generality, we may take $\Sigma_0 \cup \neg W$, why?) Writing $\Sigma_0 = \{X_1, \cdots, X_m\}$, we obtain $X_1 \wedge \cdots \wedge X_m \wedge \neg W$ is unsatisfiable (why?). By using completeness of resolution for FL, we then conclude that $\bot$ is derivable from $\Sigma_0 \cup \{\neg W\}$. Then it will follow that $\bot$ is derivable from $\Sigma \cup \{\neg W\}$ (why?). This will prove the strong completeness of resolution.

3.6 Horn Clauses in FL

The Rules E and P of equality and paramodulant take care of the equality predicate. The fragment of FL without equality is handled by the other two Rules R and F of resolution and factor. The two rules of resolution and factor are equivalent to a single rule called the rule of 'full resolution'. In Rule R, recollect that, from $A \vee C$ and $B \vee \neg D$, you derive $(A \vee B)\sigma$, where σ is the most general unifier of the literals C and D. Now, instead of just two literals, if you take an arbitrary number of literals, the resulting rule is the *full resolution.* In this context, the Rule R, as stated above, is often referred to as the rule of *binary resolution.* The full resolution rule says that

> From $A \vee C_1 \vee \cdots \vee C_m$ and $B \vee \neg D_1 \vee \cdots \vee \neg D_n$, derive $(A \vee B)\sigma$, where σ is the mgu of $\{C_1, \cdots, C_m, D_1, \cdots D_n\}$ for literals C_i, D_j, and clauses A, B.

This is written schematically as

$$\textbf{(FR)} \quad \frac{\{A_1, \ldots, A_l, C_1, \ldots, C_m\} \quad \{B_1, \ldots, B_k, \neg D_1, \ldots, \neg D_n\}}{\{A_1\sigma, \ldots, A_l\sigma, B_1\sigma, \ldots, B_k\sigma\}}$$

$(\sigma$ is the mgu of the literals $C_1, \ldots, C_m, D_1, \ldots, D_n.)$

We will not attempt a completeness proof of Rule FR to FL without equality. This would be achieved by showing that the Rules R and F can be derived from Rule FR. However, we will rewrite the clauses in a different form and see how resolution may possibly be implemented.

A clause of the form $\{C_1, C_2, \ldots, C_m, \neg D_1, \neg D_2, \ldots, \neg D_n\}$, where C_i, D_j are atomic formulas (not just literals) can be rewritten as

$$D_1 \wedge D_2 \wedge \cdots \wedge D_n \rightarrow C_1 \vee C_2 \vee \cdots \vee C_m$$

due to the equivalences $X \rightarrow Y \equiv \neg X \vee Y$, $\neg X \vee \neg Y \equiv \neg(X \wedge Y)$. Traditionally, it is written with the arrow in reverse direction, i.e., as

$$C_1 \vee C_2 \vee \cdots \vee C_m \leftarrow D_1 \wedge D_2 \wedge \cdots \wedge D_n$$

the reverse arrow being read as 'if'. Moreover, since the connective on the left of $\leftarrow$ is only $\vee$ and on the right is only $\wedge$, both of them are replaced by commas to rewrite such a formula as

$$C_1, C_2, \cdots, C_m \leftarrow D_1, D_2, \cdots, D_n$$

You must however, remember that the commas on the left of $\leftarrow$ are $\vee$'s and the commas on the right of $\leftarrow$ are all $\wedge$'s. A clause in this form is said to be in **Kowalski form**. When $m = 1$, such a clause is called a **Horn clause** as in PL.

The reason that a clause is written this way is its associated procedural meaning of searching for a model. Let us see the case of a Horn clause

$$C \leftarrow D_1, \ldots, D_n$$

For example, we may express the $grandfather$ relation as

$$\mathit{grandfather}(x, y) \leftarrow \mathit{father}(x, z), \mathit{father}(z, y)$$

To check whether a is a grand-father of b, you would first find out whether there is some c such that a is father of c and also that c is a father of b. Quantification of the variables in such a clause is seen as

$$\forall x \forall y (grandfather(x, y) \leftarrow \exists z (father(x, z), father(z, y)))$$

i.e., the extra variables to the right of *if* are all existentially quantified whose scope begins just after the symbol $\leftarrow$, and universal quantification of all the other variables is taken with the whole formula in their scopes.

Suppose that you have already built up a database of *facts* describing the relation of *father* as follows:

$$father(rajiv, rahul).$$
$$father(firoz, rajiv).$$
$$father(rajiv, Atul).$$
$$father(firoz, sanjay).$$
$$father(gandhi, firoz).$$
$$father(gandhi, devan).$$

Along with this, you also have the *rule*

$$grandfather(x, y) \leftarrow father(x, z), father(z, y).$$

Then you can query such a database, and the resolution is applied for concluding whether $grandfather(gandhi, rahul)$ holds or not. It is, in fact, enough just to unify x, y, z with the constants

$$rajiv, rahul, firoz, sanjay, gandhi, devan$$

one after another and then try to see whether they hold by matching with the facts. Let us see how the procedural meaning is given to the clauses (and quantifiers) here. Suppose you have the query

$$? - grandfather(firoz, rahul).$$

First of all, $grandfather(.,.)$ is not a fact. So, the given rule (not the resolution rules) of the database, i.e.,

$$grandfather(x, y) \leftarrow father(x, z), father(z, y)$$

is applied. While doing this, x becomes bound to (is instantiated to) $firoz$ and y becomes bound to $rahul$, so that the *goal* would be satisfied provided both $father(firoz, z)$ and $father(z, rahul)$ become true for some z. Now, in our database, we have the first fact to match with $father(firoz, z)$ is $father(firoz, rajiv)$. Thus, z becomes bound to $rajiv$ as a trial solution. This would be a solution provided that $father(rajiv, rahul)$ is true. Since it is a fact, $grandfather(firoz, rahul)$ holds.

This is how the logic programming language PROLOG works for satisfying a goal. It looks fine for a Horn clause. But Horn clauses do not cover the whole of FL. We must work out a similar plan for Kowalski forms. A first attempt is whether we can write every formula as a set of (conjunction of) Horn clauses. Though this is not possible (why?), a subclass of formulas of Kowalski form can be written as a conjunction of Horn clauses; this is practically enough for many problem solving tasks. Suppose that you can write a Kowalski form

$$C_1 \vee \cdots \vee C_m \leftarrow D_1 \wedge \cdots \wedge D_n$$

(depending on the problem domain) as a conjunction of Horn clauses

$$\begin{aligned}
&C_1 \leftarrow D_{11} \wedge D_{12} \wedge \cdots \wedge D_{1k_1}\\
&C_2 \leftarrow D_{21} \wedge D_{22} \wedge \cdots \wedge D_{2k_2}\\
&\qquad\qquad\vdots\\
&C_m \leftarrow D_{m1} \wedge D_{m2} \wedge \cdots \wedge D_{mk_m}
\end{aligned}$$

where the literals D_{ij} are from among $D_1, \ldots, D_n$. Then, the above procedural meaning can be extended to the Kowalski form. Thus, PROLOG works for the fragment of FL, which can be expressed as conjunctions of Horn clauses. In fact, it works with a slightly bigger fragment. This extension comes from augmenting *negation as failure* to it. For example, to define a subset relationship between two sets x and y, the usual way is to write $x \subseteq y \leftarrow \forall z(z \in x \rightarrow z \in y)$. As it is, in a Horn clause, we cannot allow $\forall$ on the right side of $\leftarrow$. We must look for alternate ways of writing this formula. We may express $x \subseteq y$ as "no z in x fails to belong to y". That is,

$$x \subseteq y \leftarrow \mathit{not}\, \exists z(z \in x \wedge \mathit{not}\, (z \in y))$$

Here, the connective *not* is not quite the same as $\neg$. The interpretation of *not* is procedural, in the sense that *not* X holds if X cannot be falsified, basing our reasoning on the given database. You can interpret this *not* in defining subsets, as

> x is a subset of y if it cannot be shown on the basis of the database of facts that x is not a subset of y.

So, we must have data for 'is not a subset of'. In terms of this relation, 'not a subset of', the definition will look like

> x is not a subset of y if there is some z in x which fails to be in y.

This is OK as long as we are concerned with 'not a subset of', but it is not quite OK to capture the abstract concept of 'subset of'. However, this procedural meaning of $\neg$ as *not* adds more power to the Horn clauses. Sometimes, 'negation as failure' is referred to as the *closed world assumption.* That is, if something does not hold in the given database (now all of our world), then it is false. Clearly, negation as failure with Horn clauses still do not have the full expressive power of FL. Let us see an example.

The sentence $\neg p \rightarrow p$ semantically entails p. Writing it as a rule, you have

$p \leftarrow q$, where q is $\neg p \rightarrow p$.

Taking $\neg$ as *not*, the rule is rewritten as

$p \leftarrow q$, where q is $not\, p \rightarrow p$.

Now, to satisfy the goal p, we have to first satisfy the condition $not\, p \rightarrow p$, i.e., the new goal $p \leftarrow not\, p$. But this goes on a loop if *not* is interpreted as a failure since it says that *p succeeds only when p fails.* To see it another way, suppose that p is $P(x)$ for a unary predicate P. Now $P(x)$ succeeds if *not* $P(x)$ succeeds. Both the sides of 'if' need to satisfy or falsify $P(x)$. In other words, $P(x)$ to be satisfied, calls the procedure 'falsification of $P(x)$'. Again this happens if "$P(x)$ is falsified" is falsified, calling falsification of 'falsification of $P(x)$'. This goes on *ad infinitum.* Thus, PROLOG fails in proving $P(x)$ given that $P(x) \leftarrow \neg P(x)$. That is, the procedural interpretation along with the Horn clauses would not be sufficient to capture the whole of FL.

You should not look down upon PROLOG due to its inability in recognizing a situation where a goal is unsolvable. In fact, every theorem prover will have this inability; it cannot handle all the situations of solving a goal expressed in FL. This is essentially the undecidability of first order logic.

SUMMARY

In this chapter, you have started with the method of resolution refutation for checking whether a cnf in PL is satisfiable or not. The method bases on a simple observation that if a clause contains a literal and another clause contains the negation of the same literal, then the resolvent of these two clauses formed by putting together all literals in the clauses except this pair of complementary literals is a consequence of the two clauses. Then you have seen how to avoid generating wasteful resolvents by the use of subsumption. You have also learnt how to extend the resolution method to FL by introducing the factor rule. The two rules of resolution (called as binary resolution for FL), and factor are sufficient for FL without equality. For dealing with the equality predicate, two more rules of paramodulation and equality have been introduced. You have also seen some hints towards applications of FL (and of resolution) to problem solving and logic programming. The following bibliographic remarks will be helpful to pursue the approach further.

The resolution method was found by Robinson [61] in 1965, and since then it had received much attention due to its easy machine implementation. An earlier method closely related to the resolution was the Davis Putnam method [9]. An exposition of resolution method basing on different formal systems can be found in [62]. For many modifications towards making the resolution efficient by choosing various strategies are discussed in [6, 48]. Its extension to first order logic through paramodulation has first appeared in [63]. For completeness of the resolution method for first order logic (using the rules R,F,E,P) refer [48]. There have been many refinements of the resolution method. Some refinements such as linear resolution, model elimination, unit resolution, etc. have been tried keeping in view a particular class of clauses, on which they become efficient too. For these refinements, you may consult [43, 48, 62]. The resolution method has been extended to higher order logics also; see for example, [1]. The resolution method is extensively used in computing the set of prime implicants and implicates [77] arising in minimization of Boolean circuits and knowledge compilation. Before exploring the resources, attempt the following problems.

PROBLEMS

1. Construct clause sets corresponding to the following formulas and consequences:

(a) $\exists x X \leftrightarrow \neg\forall x \neg X$

(b) $\forall x(\exists y(Pxy \to Qxy) \wedge \exists y(Qxy \to Pxy))$

(c) $\forall x(\exists y Pxy \wedge \neg Qxy) \vee \forall y \exists z(Qyz \wedge \neg Rxyz)$

(d) $\neg(\exists x \forall y Pxy \to (\forall x \exists z \neg Qzx \wedge \forall y \neg \forall z Rzy))$

(e) $\exists x \forall y \forall z \exists x(\exists u(Pxz \vee Qxy) \leftrightarrow \neg \exists u \neg \exists w(Qxw \wedge \neg Rxu))$

(f) $\exists x \exists y(Pxy \to \forall x(Qx \wedge Rxy))$

(g) $\forall x Px \to \exists x \exists y(Qx \vee Rxy)$

(h) $\forall x \forall y(\forall z Pxyz \vee (\forall y Rxy \to \exists u(Rxu \wedge Quz)))$

(i) $\{p \vee q \to r, \neg s \wedge \neg t \to \neg r, \neg t \vee u, \neg(\neg s \to u)\} \models \neg p$

(j) $\{p \to (q \to r), r \wedge s \to t, \neg s \vee t \to u\} \models p \to (q \to r)$

(k) $\{\forall x(\exists y(Pxy \wedge Qy) \to \exists y(Ry \wedge Bxy)), \forall x \neg Rx\} \models (Qy \to Pxy)$

(l) $\{\forall x(Px \to \forall y(Qxy \to Ry)), \exists y(Py \wedge \forall x(Bx \to Qyx))\} \models \forall z Bz$

2. Construct essentially two different resolution proofs along with the resolution DAGs of $(p \leftrightarrow q) \wedge (q \leftrightarrow r) \to (p \leftrightarrow r)$.

3. Show that the composition of substitutions is associative. Is it also commutative? Justify your answer.

4. Find a pair of separating variants for the following clauses:

(a) $\{Pxyf(z)\}, \{Pzyf(y)\}$

(b) $\{Pxy, Pyz\}, \{Qzy, Rf(y)z\}$

(c) $\{Pxg(x)y\}, \{Pxg(x)y\}$

5. Find clauses A, B and a substitution σ such that $(A - B)\sigma$ is not a subset of $A\sigma - B\sigma$.

6. Use *Unification* algorithm to find the mgu, if it exists, for each of the following clauses:

(a) $\{Pxy, Pf(x)y\}$

(b) $\{Pxayf(y), Pzyg(c)u\}$

(c) $\{Pf(x)y, Pzz, Pxf(u)\}$

(d) $\{Pxf(y)z, Pcg(u)w, Puvg(w)\}$

7. Is it true that if σ and θ are two most general unifiers of a clause A, then $\theta = \sigma\delta$ for a variant δ? If you think 'no', give a counter example. If 'yes', give a proof.

8. Find $R^*(A)$ and $RS^*(A)$ for the following clause sets A:

(a) $\{\{p, \neg q\}, \{p, q\}, \{\neg q\}\}$

(b) $\{\{q, r\}, \{\neg p, \neg q\}, \{\neg r, \neg p\}\}$

(c) $\{\{p\}, \{q\}, \{p, q\}\}$

(d) $\{\{\neg p\}, \{\neg p, r\}, \{p, q, \neg r\}\}$

(e) $\{\{\neg p, \neg r\}, \{\neg q, \neg r\}\{p, q, r\}\}$

9. Attempt resolution proofs of the following consequences. Also construct the resolution DAGs.

(a) $\{\{p, \neg q, r\}, \{q, r\}, \{\neg p, r\}, \{q, \neg r\}, \{\neg q\}\} \models \bot$

(b) $\{\{p, \neg q\}, \{r, p\}, \{\neg q, r\}, \{\neg p, q\}, \{q, \neg r\}, \{\neg r, \neg p\}\} \models \bot$

(c) $\{\{\neg p, q\}, \{\neg q, r\}, \{p, \neg r\}, \{p, q, r\}, \{\neg p, \neg q, \neg r\}\} \models \bot$

(d) $(p \leftrightarrow (q \to r)) \wedge (p \leftrightarrow q) \wedge (p \leftrightarrow \neg r) \models \bot$

(e) $\top \models (((p \to q) \to \neg q) \to \neg q)$

(f) $((p \wedge q) \vee (p \wedge r) \vee (q \wedge r)) \vee (\neg p \wedge \neg q) \vee (\neg p \wedge \neg r) \vee (\neg q \wedge \neg r) \models \bot$

(g) $A \to (B \to A)$

(h) $(A \to (B \to C)) \to ((A \to B) \to (A \to C))$

(i) $(\neg B \to \neg A) \to (A \to B)$

(j) $(A \wedge B) \leftrightarrow \neg(A \to \neg B)$

(k) $(A \vee B) \leftrightarrow (\neg A \to B)$

(l) $(A \leftrightarrow B) \leftrightarrow (A \to B) \wedge (B \wedge A)$

(m) $\{\{Px, Qx\}, \{Pa\}, \{\neg Py\}, \{\neg Qz\}\}$

(n) $\{\{Pxy, Qxa\}, \{Pax, \neg Qyx\}, \{\neg Pyy\}\}$

(o) $\{\{\neg Px, Qx\}, \{Pc\}, \{Rac\}, \{\neg Qz, \neg Raz\}\}$

(p) $\forall x(X \to Y) \to (X \to \forall xY)$, if x does not occur free in X

(q) $\exists xX \leftrightarrow \neg\forall x\neg X$

(r) $\forall xX \to X[x/t]$, if t is a term free for x in X

(s) $\{Pc, \forall x(Px \to Qx), \forall x(Rx \to \neg Qx), Ra\} \models \neg(c \approx a)$

(t) $\{\{Px, Qx, Rxf(x)\}, \{\neg Px, Qx, Sf(x)\}, \{Ac\}, \{Pa\},$
$\{\neg Ray, Ay\}, \{\neg Ax, \neg Qx\}\} \models \exists x(Ax \wedge Sx)$

(u) $\{\forall x\forall y(fxy \approx fyx), \forall x\forall y(fxy \approx y)\forall x\exists y\neg(x \approx y)\}$
$\models \forall x\exists yQxy \wedge \exists y\forall x\neg Qyx$

(v) $\forall x\forall y\forall z(Pxy \wedge Pyz \to \neg Qxz) \wedge \forall x\forall y(Pxy \leftrightarrow (Qyx \vee Rxy))$
$\wedge \forall x\exists yPxy \models \forall xPxx$

10. Use resolution to determine the validity of the following arguments.

(a) Some students read logic books seriously. No student ever reads a boring logic book. All logic books in your library, including this one, is read by all students seriously. Therefore, none of the logic books in your library is boring.

(b) Some people love anyone who likes the leader. Everyone loves someone who likes the leader. None, who loves someone who likes the leader, likes the leader. Therefore, the leader does not like himself.

(c) No teacher who likes to write books in logic or who devotes himself to his students will be in the good books of the administration. No one who is not in the good books of the administration will be promoted. Therefore, no teacher who likes to write books in logic will be promoted.

(d) Arjun loves all and only those who love Urvasi. Urvasi loves all and only those who love Arjun. Arjun loves himself. Therefore, Urvasi loves herself.

11. A clause A *subsumes* a clause B iff there is a substitution σ such that $A\sigma \subseteq B$. Construct an algorithm for determining whether a clause subsumes another.

12. For every propositional variable p, suppose that you rewrite the literals p and its negation $\neg p$ as $\neg p$ and p, respectively, then a resolution proof of a set of clauses when rewritten will remain essentially the same. Show it.

13. Show that deletion of trivial clauses does not affect the adequacy (soundness and completeness) of resolution.

14. If a clause C subsumes D, then removal of D does not affect the adequacy of resolution. Show it.

15. Let A be a clause set with each clause in it having at most two literals. Show that the resolution method determines the satisfiability of A in a time which is a polynomial in the length of A.

16. Let B be a clause set, each clause of which has at most three literals. Give reasons why your solution of Problem 15 may not prove that the resolution method determines satisfiability of such a clause set B in polynomial time.

17. For a clause set A (in PL), define $S^n(A)$ recursively by

$$S_0(A) = A,\ S_{m+1}(A) = \{C : C \text{ is a resolvent of two clauses in } S_m(A)\}$$

and then take $S^*(A) = \cup_{n\in\mathbb{N}} S_n(A)$. Give an example of a clause set A such that A is unsatisfiable but $\bot \notin S^*(A)$.

18. For a clause set A (in PL), define $U^n(A)$ recursively by

$$\begin{aligned} &U_0(A) = A,\\ &U_{m+1}(A) = U_m(A) \cup \{C : C \text{ is a resolvent of two clauses in } U_m(A)\\ &\qquad\qquad\qquad \text{one of which is a singleton}\} \end{aligned}$$

and then take $U^*(A) = \cup_{n\in\mathbb{N}} U_n(A)$. This is called *unit resolution.* Give an example of a clause set A such that A is unsatisfiable but $\bot \notin U^*(A)$.

19. Let A be any clause set in PL. For any set $B \subseteq A$, write $N(A, B) = A \cup \{C : C$ is a resolvent of a clause in B and a clause in $A\}$. Define $UN_n(A)$ inductively by:

$$UN_0(A) = A,\ UN_{m+1}(A) = N(UN_m(A), A)$$

Let $UN^*(A) = \cup_{n\in\mathbb{N}} UN_n(A)$. Show that $\bot \in U^*(A)$ iff $\bot \in UN^*(A)$.

20. Let A be a unifiable clause. Let δ be a substitution computed by the *Unification* algorithm. Show that $A\delta$ is unifiable.

21. By using Problem 20, or in some other way, prove the correctness of the *Unification* algorithm.

22. Show that if a clause A is unifiable, then there exists a most general unifier. Can you prove this without using the previous problem?

23. A trivial example to show that an mgu is not unique is to use a variant. Give a nontrivial example.

24. How many occurrences of the variable x_1 are there in the mgu of the terms $f(x_2, x_3, \ldots, x_n)$ and $f(g(x_1, x_1), g(x_2, x_2), \ldots, g(x_{n-1}, x_{n-1}))$?

25. Show that the factor rule can be restricted to clauses consisting of a pair of literals only.

26. Is a factor of two clauses a conclusion of those two clauses? Justify your answer either with a proof or with a counter example.

27. Show that if a clause is satisfiable, all its factors are also satisfiable.

28. Is a paramodulant of two clauses a conclusion of those two clauses? Justify your answer either with a proof or with a counter example.

29. Let D be a domain (a nonempty set of symbols). Then the *ground terms* from D are defined by using the definition of terms restricting the basic terms to the elements of D rather than taking all variables and constants. The ground literals and ground clauses are defined using the ground terms instead of any arbitrary term. For any clause, its *ground instances* are obtained by replacing a variable in the clause by a ground term. Show that

(a) For ground clauses, whatever that can be derived by full resolution, can also be derived by binary resolution.

(b) For any pair of clauses A, B, every resolvent of a ground instance of A and a ground instance of B is a ground instance of some resolvent of A and B.

(c) For clauses A and B, every ground instance of a resolvent of A and B contains as a subset some resolvent of a ground instance of A and a ground instance of B.

30. Use Problem 29 to show that a clause set is unsatisfiable iff $\bot$ can be derived from it by resolution.

31. Let A be a clause set and $B \subseteq A$ is such that $A - B$ is satisfiable. A clause in $R^*(A)$ *has B-support* iff it is in B or is a resolvent of two clauses, of which at least one has B-support. Let $R_1(A) = A \cup \{C \in R(S) : C$ has B-support$\}$. Define $R_1^*(A)$ in the usual way. Show that A is unsatisfiable iff $\bot \in R_1^*(A)$.

32. Prove strong adequacy of resolution to FL using the idea described at the end of Section 3.5. That is, assuming compactness for both PL and FL, show that for any set Σ of formulas and any formula W, $\Sigma \models W$ iff $\bot$ is derivable from $\Sigma \cup \{\neg W\}$ by resolution. [Here you are using all the Rules R, F, E, and P.]

4

Proofs in PL and FL

4.1 Introduction

You have seen many proofs of theorems, and also have proved many theorems yourself. How does a proof look like? Let us see a simple one. With the axioms of $\mathbb{R}$, the real number system, as you have used in your school days, can you prove that there is only one 0? What does this statement mean? Well, we have the axioms of $\mathbb{R}$ as "+ is commutative, associative, having the identity as 0, and corresponding to each $x \in \mathbb{R}$, there is a $y \in \mathbb{R}$ such that $x + y = 0$". Similar axioms for the operation $\times$ also holds. We also have axioms of the relation $<$, etc. We ask how to use these axioms or rules to show that the identity element 0 of + is unique, i.e., there does not exist any (other than 0) element $e \in \mathbb{R}$ having the property that for every $x \in \mathbb{R}, x + e = x$. Here is a standard proof:

> Let a and b be two such elements. Then, for every $x \in \mathbb{R}$,
> $x + a = x,\ x + b = x$. Consequently, $a = a + b = b + a = b$.

Now, why is such a proof important? Because its correctness can be checked mechanically. All that you have to see is whether the axioms have been applied in the correct way and whether nothing else has been used. Can you have a similar proof mechanism for PL and FL? In this chapter, we will try to answer this question. We will see that not only the proofs in PL and FL can be mechanically checked, but also the proof systems give more insight into the logics.

4.2 Axiomatic System PC

What then constitutes a proof? We start with some axioms, the propositions or formulas which are accepted without any further question. They are like the postulates of geometry, which should be self-evident. We will start with valid propositions or formulas, but evident or not,we cannot say without semantics. Here, in a proof, we are not supposed to use any kind of truth unless we give meaning to the syntactic entities. Then a proof will have only some rules telling how to proceed from the axioms towards deriving other formulas, which would be termed as theorems in the axiomatic system. Such rules will be termed as rules of inference.

We start with an axiomatization of PL and then proceed towards one for FL. In PL, we know that many connectives are redundant, i.e., a subset of connectives can be chosen to do the job, the so-called adequate sets or truth functionally complete sets of connectives. Some such sets of connectives are $\{\uparrow\}, \{\downarrow\}, \{\neg, \to\}, \{\neg, \vee\}$, and $\{\neg, \wedge\}$. We choose the set $\{\neg, \to\}$. Thus we are planning to work with the fragment of PL which uses the only connectives as $\neg$ and $\to$. This does not in any way restrict our method to this fragment of PL alone since other connectives can be tackled by way of definitions such as $p \vee q \equiv \neg p \to q$. Instead of making the system complicated with these definitions, we assume that we have in PL, the only connectives as $\neg$ and $\to$. Similarly, we choose not to use the propositional constants $\top$ and $\bot$ in this fragment of PL since they can be introduced or eliminated by definitions: $\top \equiv (p \to p)$, and $\bot \equiv \neg(p \to p)$. Our **axioms** are:

(A1) $A \to (B \to A)$

(A2) $(A \to (B \to C)) \to ((A \to B) \to (A \to C))$

(A3) $(\neg A \to \neg B) \to (B \to A)$

In fact, these are **axiom schemes**, in the sense that $p \to (q \to p)$ is an axiom as it is obtained from A1 by replacing A by p and B by q throughout. Any such replacement gives an instant of the axiom scheme A1. The instance is the axiom A1. We thus take A1 to A3 as axiom schemes and remember that any instance of an axiom scheme is an axiom. Moreover, an instance of a scheme is obtained by replacing throughout the letters A, B, C by any propositions p, q, r, respectively. In addition to the axioms (axiom schemes), we have a **rule of inference**:

(MP) $\dfrac{A \quad A \to B}{B}$

You can see that this inference rule is the consequence Modus Ponens: $\{A, A \to B\} \models B$, written as a fraction. Since in the axiomatic system there is no truth or falsity, we simply write the consequence as a fraction. You may read the rule MP as "from A and $A \to B$, derive B". This is again a rule scheme in the sense that any instance of this scheme is a rule. That is, if you have already (derived) the propositions p and $p \to q$, then the rule allows you to derive q, whatever the propositions p and q may be. For example, deriving $q \to r$ from $p \to q$ and $(p \to q) \to (q \to r)$ is also an application of the rule MP. Note that we need not explain what the word 'derive' means. Informally, it signals deductions, but formally, it may just turn out to be writing the propositions one after another. An axiom is considered derived from any proposition. Similarly, when we already have (derived) propositions p and $p \to q$, we can write q anywhere thereafter, due to MP. We now give an example of a proof in our axiomatic system.

***EXAMPLE* 4.1** Here is a proof:

1. $(p \to (q \to p))$
   $\to (r \to (p \to (q \to p)))$ A1, $A := p \to (q \to p), B := r$
2. $p \to (q \to p)$ A1, $A := p, B := q$
3. $r \to (p \to (q \to p))$ MP, $A :=$ line 2, $A \to B :=$ line 1

The explanation of the above proof is fairly clear. Our documentation on the right says why the propositions in lines 1 and 2 are axioms. It says that the lines 1 and 2 are instances of A1. The proposition on the third line is derived from the propositions in lines 1 and 2 by an application of the inference rule MP. Note that the line numbers on the left help us annotate a proof by keeping proper documentation. Neither the line numbers nor the annotation forms a part of the proof; the proof is in the middle column. The first and the third columns help us read the proof. Moreover, the proof *proves* the proposition $r \to (p \to (q \to p))$, and this proposition is a theorem of the axiomatic system. Let us give a name to this axiomatic system so that we may easily talk *about* it.

Definition 4.1 The axiomatic system **PC** has all the propositions having the only connectives as $\neg$ and $\to$. PC has axiom schemes A1, A2, A3, and the inference rule MP. A **proof in PC** is a finite sequence of propositions, where each one is either an axiom or is obtained (derived) from earlier two propositions by an application of MP. The last proposition of a proof is a **theorem of PC**; the proof is said to prove the theorem. The fact that "A is a theorem" is written as $\vdash A$. We also read $\vdash A$ as "A is **provable**".

In an axiomatic system such as PC, the notion of a proof is *effective*, i.e., if it is claimed that some object is a proof of a theorem, then it can be checked whether the claim is correct or not in an algorithmic manner. However, construction of a proof is not effective; there may not be an algorithm to construct a proof of a given theorem. Of course, proofs can be generated mechanically by following the rules as given in Definition 4.1; the problem comes when it is targeted towards proving a given theorem. We will see by way of examples how to do it; the point is that we may have to rely on our intuition; it is not mechanical. The examples will, in a way, train you towards achieving a goal towards generating a particular theorem at the end of a proof.

***EXAMPLE* 4.2** Show that $\vdash (p \to p)$.

Solution It is so simple a theorem to be proved, but what do we start with, A1, A2, or A3? A1 is $A \to (B \to A)$. It does not seem to be helpful. But we have $p \to (p \to p)$, as one of its instances. If somehow we are able to eliminate the first p, then we are through. The only way of elimination is the inference rule MP. It would require p so that from p and $p \to (p \to p)$ we will arrive at $p \to p$. But it seems we cannot derive p.

Let us look at A2. It looks like: $(p \to (q \to r)) \to ((p \to q) \to (p \to r))$. If we plan to apply MP, we may derive $p \to r$, provided we have already the propositions: $p \to (q \to r)$ and $p \to q$. But towards reaching $p \to p$, we have to take r as p. Thus, we must have the two propositions (replace r by p): $p \to (q \to p)$ and $p \to q$. Now, the first of these propositions, i.e., $p \to (q \to p)$ is simply A1. So, you see, some progress has been made. At this stage, our plan is to start with A1 as $p \to (q \to p)$ and A2 as $(p \to (q \to p)) \to ((p \to q) \to (p \to p))$, and apply MP to get $(p \to q) \to (p \to p)$. Fine, but then how to eliminate $p \to q$? We have again A1 as $p \to (q \to p)$. Thus, instead of q if we had $p \to q$, then we would have been through. Well, then replace q by $p \to q$ throughout. That is, we would start with A2, where we replace r by p and q by $p \to q$, and start all over again.

Our plan now compels us to start with A2 as $(p \to ((q \to p) \to p)) \to ((p \to (q \to p)) \to (p \to p))$, use A1 as $p \to ((q \to p) \to p)$, conclude by MP that $(p \to (q \to p)) \to (p \to p)$. Next, we use A1 as $p \to (q \to p)$ and apply MP to conclude $p \to p$. Fine, you got it! But we must also write the whole argument as a proof in PC. Here is the proof:

1.	$p \to ((q \to p) \to p)$	A1: $B := q \to p$
2.	$(p \to ((q \to p) \to p))$ $\to ((p \to (q \to p)) \to (p \to p))$	$A2 : B := q \to p, C := p$
3.	$(p \to (q \to p)) \to (p \to p)$	$1, 2, \text{MP}$
4.	$p \to (q \to p)$	$A1 : A := p, B := q$
5.	$p \to p$	$3, 4, \text{MP}$

Henceforth, we will make our heavy documentation light by avoiding writing the replacements of A, B, C by other propositions; you should be able to find out these replacements by looking at the proof and the axiom scheme mentioned over there. Further, when MP is applied to two previous lines, we will not mention the line numbers also. If it is applied to one previous line and another earlier but remote line, we will simply mention this remote line number. For example, the rightmost entry in line 1 in the above proof would be written as A1, omitting the replacement $B := q \to p$, and that in line 3 would simply be written as MP.

Just like axiom schemes and inference rules, theorems are theorem schemes. Once you have a proof of $p \to p$, you can have a proof of $(p \to q) \to (p \to q)$. It is simple; just replace p by $p \to q$ throughout the proof. Thus theorems can be helpful in proving other theorems.

***EXAMPLE* 4.3** Show that $\vdash q \to (p \to p)$.

Solution Look at Examples 4.1 and 4.2. Do they suggest anything? In Example 4.2, we have proved $p \to p$; we will continue that proof by appending an appropriate copy of Example 4.1. Here is the proof:

1.	$p \to ((q \to p) \to p)$	
	$\vdots$	
5.	$p \to p$	Example 4.2
6.	$(p \to p) \to (q \to (p \to p))$	A1
7.	$q \to (p \to p)$	MP

This suggests that we can use (already proved) theorems as theorem schemes to construct other proofs. Though such a construction is not a proof, a proof can be constructed by following the intended replacements in the referred existing proofs. Such proofs which use existing theorems are called *quasi-proofs*; but we will not make our vocabulary heavy by more and more technical words. we will simply accept these as proofs. Whenever we refer to another existing theorem (may be with suitable replacements), we document this by mentioning ‘ Th’ on the rightmost column. The proof (quasi-proof, indeed) in Example 4.3 can be rewritten as

1.	$p \to p$	Th
2.	$(p \to p) \to (q \to (p \to p))$	A1
3.	$q \to (p \to p)$	MP

***EXAMPLE* 4.4** Show that $\vdash \neg q \to (q \to p)$.

Solution It seems A3 will be useful since it is the only axiom which mentions the symbol $\neg$. A3 looks like:

$$(\neg p \to \neg q) \to (q \to p)$$

But, instead of $\neg p \to \neg q$, we have only $\neg q$ in our suggested theorem. Well, $\neg q$ can give us $\neg p \to \neg q$ due to A1. One second, what does it mean to say that "$\neg q$ can give us $\neg p \to \neg q$"? It means that if we have already $\neg q$ appearing in a proof, then we can use A1 to have $\neg q \to (\neg p \to \neg q)$ and apply MP to obtain $\neg p \to \neg q$. Well then, our plan is to use

$$\neg q \to (\neg p \to \neg q) \text{ and } (\neg p \to \neg q) \to (q \to p)$$

But how to obtain $\neg q \to (q \to p)$ from these two? Is there any way to eliminate $\neg p \to \neg q$? Right now, it seems not plausible.

For using MP, it would be worthwhile to try

$$(\neg q \to (\neg p \to \neg q)) \to (\neg q \to (q \to p))$$

since by A1, we already have $\neg q \to (\neg p \to \neg q)$. But how to show this? This proposition matches with the latter part of A2, i.e., with

$$(A \to B) \to (A \to C)$$

The replacements are $A := \neg q, B := (\neg p \to \neg q)$ and $C := (q \to p)$. The missing part is $A \to (B \to C)$ which, with the intended replacements, would look like:

$$\neg q \to ((\neg p \to \neg q) \to (q \to p))$$

If this is the target now, then we find that the proposition after $\neg q \to$ is just A3. We have seen in Example 4.3 that if we have a proof of X, then we can get a proof of $Y \to X$ for any Y. Using this heuristic, we now have the required proof. The analysis till now is backward: "to get the theorem, we should already have so and so", etc. We want to write the proof from axioms to the theorem; not otherwise. This can be done easily once you have visualized the plan. At this point, close the book and try to construct the proof yourself; come back to this page after five minutes. Here is the proof:

1.	$(\neg p \to \neg q) \to (q \to p)$	A3
2.	$((\neg p \to \neg q) \to (q \to p)) \to (\neg q \to ((\neg p \to \neg q) \to (q \to p)))$	A1
3.	$\neg q \to ((\neg p \to \neg q) \to (q \to p))$	MP
4.	$(\neg q \to ((\neg p \to \neg q) \to (q \to p))) \to$ $((\neg q \to (\neg p \to \neg q)) \to (\neg q \to (q \to p)))$	A2
5.	$(\neg q \to (\neg p \to \neg q)) \to (\neg q \to (q \to p))$	MP
6.	$\neg q \to (\neg p \to \neg q)$	A1
7.	$\neg q \to (q \to p)$	MP

Fine, we have got a proof. But let us look at the rough work above, where we thought we might get a proof using $\neg q \to (\neg p \to \neg q)$ and $(\neg p \to \neg q) \to (q \to p)$. That is, if we are allowed to assume $\neg q$, then first, we can derive $\neg p \to \neg q$, and then using A3 and MP, we obtain $q \to p$. This will prove the consequence that "from the sole premise $\neg q$ follows the conclusion $q \to p$". Here is a formal definition of the notion of consequences in our axiomatic system PC.

Definition 4.2 Let Σ be a set of propositions and w be any proposition. A **proof of the consequence** $\Sigma \vdash w$ is a finite sequence of propositions where every proposition is either an axiom, a proposition in Σ, or is obtained from earlier two propositions by an application of MP, and the last proposition in the sequence is w. Each proposition in Σ is called a **premise**, and w is called the **conclusion**. We simply write $\Sigma \vdash w$ to say that there exists a proof of the consequence $\Sigma \vdash w$; in that case, we also say that the consequence is **provable**. When $\Sigma = \{A_1, \ldots, A_n\}$, a finite set, we also write $\Sigma \vdash w$ as $A_1, \ldots, A_n \vdash w$.

We just compare the Definitions 4.1 and 4.2 and observe that when $\Sigma = \emptyset$, the consequence $\Sigma \vdash w$ boils down to $\vdash w$. As in the proofs of theorems, we will have three columns, the first of which will keep record in giving serial numbers to the propositions, the third will be used to document the proof, and the second one will be the real proof. We will mention the letter 'P' in the documentation to say that the proposition used in that line of the proof is a premise. We have a ready-made example:

***EXAMPLE* 4.5** Show that $\neg q \vdash q \to p$.

Solution Here is the proof:

1.	$\neg q$	P
2.	$\neg q \to (\neg p \to \neg q)$	A1
3.	$\neg p \to \neg q$	MP
4.	$(\neg p \to \neg q) \to (q \to p)$	A3
5.	$q \to p$	MP

Let us recollect that in trying to construct the proof in Example 4.4, we could not derive $\neg q \to (q \to p)$ from the two propositions $\neg q \to (\neg p \to \neg q)$ and $(\neg p \to \neg q) \to (q \to p)$ because we did not know how to eliminate $\neg p \to \neg q$. Let us remember, such a semantic consequence was a law. What was it? It says something like: "from the premises $A \to B$ and $B \to C$, derive $A \to C$", the so-called Hypothetical Syllogism. Can you show that in PC?

***EXAMPLE* 4.6** $p \to q, q \to r \vdash p \to r$.

Solution Of course, we can start from a premise, say, from $p \to q$. Then, how to arrive at $p \to r$? If we have $(p \to q) \to (p \to r)$, then we can use MP to get the conclusion. A look at A2 says that this new proposition matches with its second part. So, we should have already derived the first part. The first part of A2, here, would look like: $p \to (q \to r)$. We have the other premise as $q \to r$. Then, we can derive this by using A1 as in Example 4.1. As earlier, try to construct the proof yourself, and return to this page after making some progress. The proof is:

1.	$q \to r$	P
2.	$(q \to r) \to (p \to (q \to r))$	A1
3.	$p \to (q \to r)$	MP
4.	$(p \to (q \to r)) \to ((p \to q) \to (p \to r))$	A3
5.	$(p \to q) \to (p \to r)$	MP
6.	$p \to q$	P
7.	$p \to r$	MP

Each axiom is an axiom scheme. Each rule of inference is a rule scheme. Each theorem is a theorem scheme. Then each consequence is also a consequence scheme. Is it? Of course, yes. If you replace the symbols $p, q, r, \ldots$ by propositions uniformly then the resulting consequence will also have a proof; the proof is exactly the same proof with the replacements done to every line of it. But then each such consequence scheme can be used as an additional inference rule just as already proved theorems are used as axioms. Why? Because whenever it is used, its proof can be duplicated there itself to get a real proof. So, we will allow already proved consequences as new inference rules; these are referred to as **derived rules** of inference. To illustrate this point, we give another proof of the theorem $\vdash \neg q \to (q \to p)$ (see Example 4.4) in Example 4.7 below using the derived rule, Hypothetical Syllogism: $\{p \to q, q \to r\} \vdash p \to r$. The inference rule is rewritten as

$$\textbf{(HS)} \qquad \frac{p \to q \quad q \to r}{p \to r}$$

In such cases, you must take care that in the proof of Example 4.6, the consequence in Example 4.4 has not been used. Why?

***EXAMPLE* 4.7** Here is a proof of $\vdash \neg q \to (q \to p)$ that uses HS:

1.	$\neg q \to (\neg p \to \neg q)$	A1
2.	$(\neg p \to \neg q) \to (q \to p)$	A3
3.	$\neg q \to (q \to p)$	HS

Compare the theorem $\vdash \neg q \to (q \to p)$ and the consequence $\neg q \vdash q \to p$. What does it hint at?

Theorem 4.1 (Deduction Theorem) *Let Σ be a set of propositions and p, q be any propositions. Then, $\Sigma \vdash p \to q$ iff $\Sigma \cup \{p\} \vdash q$.*

Proof Suppose that $\Sigma \vdash p \to q$. To show $\Sigma \cup \{p\} \vdash q$, take the proof of $\Sigma \vdash p \to q$, add to it the lines (propositions) p, q. It looks like:

1.	...	
2.	...	
$\vdots$		
n.	$p \to q$	Proof of $\Sigma \vdash p \to q$
$n+1$.	p	P
$n+2$.	q	MP

This is a proof of $\Sigma \cup \{p\} \vdash q$. Conversely, suppose that $\Sigma \cup \{p\} \vdash q$. This means that we have a proof of it. We want to construct a proof of $\Sigma \vdash p \to q$. This is not so straightforward. We use induction on the number of propositions (number of lines) used in the proof `P` of $\Sigma \cup \{p\} \vdash q$.

In the basis step, suppose that `P` has only one proposition. Then this proposition has to be q. Now, why is this a proof? There are three cases to consider: (a) q is an axiom; (b) $q \in \Sigma$; (c) $q = p$. In each case, we show that we can get a proof of $\Sigma \vdash p \to q$. In case (a), our proof is:

1.	q	An axiom
2.	$q \to (p \to q)$	A1
3.	$p \to q$	MP

Is it a proof of $\Sigma \vdash p \to q$? Yes, since it uses no proposition other than an axiom (as in `P`) or one in Σ. In case (b), the above proof still works, only the first line would be documented as 'P', a premise rather than an axiom. In case (c), $q = p$. In Example 4.2, we proved that $\vdash p \to p$. Thus, with that proof, we now have a one-line proof of $\Sigma \vdash p \to p$ as

1. $p \to p$ Th

We go to the induction step. Lay out the induction hypothesis that "If there is a proof of $\Sigma \cup \{p\} \vdash q$ less than n propositions, then there is a proof of $\Sigma \vdash p \to q$." Suppose now that we have a proof of $\Sigma \cup \{p\} \vdash q$ having n propositions in it. We will construct a proof of $\Sigma \vdash p \to q$.

Observe that we have three trivial cases to consider: (i) q is an axiom; (ii) $q \in \Sigma$; (iii) $q = p$. In all these cases, we have proofs for $\Sigma \vdash p \to q$ as in the basis case. Dispensing with the trivial cases, we have the only nontrivial (and more realistic) case that q has been derived by an application of MP in the proof P of $\Sigma \cup \{p\} \vdash q$. How would P look like then?

$$\begin{array}{lll}
\texttt{P}:\ 1. & \ldots & \\
\quad\ \ 2. & \ldots & \\
\quad\ \ \vdots & & \\
\quad\ \ m. & r & \\
\quad\ \ \vdots & & \\
\quad\ \ m+k. & r \to q & \\
\quad\ \ \vdots & & \\
\quad\ \ n. & q & m,\ m+k, \text{MP}
\end{array}$$

where $m < n, m + k < n$, and r is some proposition. Note that the proof segments:

P1 : lines 1 to m

P2 : lines 1 to $m + k$

in P are themselves proofs of the consequences:

$$\Sigma \cup \{p\} \vdash r$$
$$\Sigma \cup \{p\} \vdash r \to q$$

respectively. Clearly, the proofs P1, P2 have less than n number of propositions. By induction hypothesis, corresponding to P1, P2, there are proofs P3, P4 of the consequences:

$$\Sigma \vdash p \to r$$
$$\Sigma \vdash p \to (r \to q)$$

Suppose that P3 has i number of propositions and P4 has j number of propositions. We use the two proofs P3, P4 to construct the proof P5 of $\Sigma \vdash p \to q$. The proof P5 is constructed by adding P4 to P3, and then some more propositions as illustrated below:

$$\begin{array}{lllll}
\texttt{P5}: & \texttt{P3}: & 1. & \ldots & \\
& & \vdots & & \\
& & i. & p \to r & \\
& \texttt{P4}: & i+1. & \ldots & \\
& & \vdots & & \\
& & i+j. & p \to (r \to q) & \\
& & i+j+1. & (p \to (r \to q)) \to ((p \to r) \to (p \to q)) & \text{A3} \\
& & i+j+2. & (p \to r) \to (p \to q) & \text{MP} \\
& & i+j+3. & p \to q & i,\ j+j+2, \text{MP}
\end{array}$$

It is clear that `P5` is a proof of $\Sigma \vdash p \to q$ as no propositions other than axioms and premises in Σ are used in it. ▮

Note that the deduction theorem is not a theorem of PC. It speaks something about the axiomatic system PC. Thus, it is a metatheorem for the axiomatic system PC. So, do not confuse between a theorem of PC and the results about PC.

***Exercise* 4.1** Construct a proof of $\vdash \neg q \to (q \to p)$ from the proof of $\neg q \vdash q \to p$ as given in Example 4.5 by following the proof of the deduction theorem.

The deduction theorem can be used to construct simpler proofs, especially for propositions involving serial implications.

***EXAMPLE* 4.8** Show that $\vdash ((\neg p \to q) \to ((q \to \neg p) \to p))$
$\to (((\neg p \to q) \to (q \to \neg p)) \to ((\neg p \to q) \to p))$.

Solution

1.	$\neg p \to q$	P
2.	$(\neg p \to q) \to (q \to \neg p)$	P
3.	$q \to \neg p$	MP
4.	$(\neg p \to q) \to ((q \to \neg p) \to p)$	P
5.	$(q \to \neg p) \to p$	1, MP
6.	p	3, MP

Because of the deduction theorem, it was enough to prove

$$(\neg p \to q) \to ((q \to \neg p) \to p), (\neg p \to q) \to (q \to \neg p), \neg p \to q \vdash p$$

Look at line 5. We mention only the line number as 1; this means that we take line 1 along with the previous line, i.e., line 4 and apply MP to derive line 5. Similarly, line 6 is obtained by applying MP to lines 3 and 5, and we have mentioned only line 3 in the justification column.

***Exercise* 4.2** Try to construct a proof of the theorem in Example 4.8 without using the deduction theorem.

***EXAMPLE* 4.9** Show that (a) $\vdash p \to \neg\neg p$; (b) $\vdash \neg\neg p \to p$.

Solution Proof for (a):

1. $\neg\neg\neg p \to (\neg\neg p \to \neg\neg\neg\neg p)$	Th (Example 4.7)
2. $(\neg\neg p \to \neg\neg\neg\neg p) \to (\neg\neg\neg p \to \neg p)$	A3
3. $\neg\neg\neg p \to (\neg\neg\neg p \to \neg p)$	HS
4. $(\neg\neg\neg p \to (\neg\neg\neg p \to \neg p))$ $\to ((\neg\neg\neg p \to \neg\neg\neg p) \to (\neg\neg\neg p \to \neg p))$	A2
5. $(\neg\neg\neg p \to \neg\neg\neg p) \to (\neg\neg\neg p \to \neg p)$	MP
6. $\neg\neg\neg p \to \neg\neg\neg p$	Th
7. $\neg\neg\neg p \to \neg p$	MP
8. $(\neg\neg\neg p \to \neg p) \to (p \to \neg\neg p)$	A3
9. $p \to \neg\neg p$	MP

Proof for (b):

1. $\neg p \to \neg\neg\neg p$	Th (From (a))
2. $(\neg p \to \neg\neg\neg p) \to (\neg\neg p \to p)$	A3
3. $\neg\neg p \to p$	MP

***Exercise* 4.3** Construct a proof of $\vdash \neg\neg p \to p$ without using $p \to \neg\neg p$.

***EXAMPLE* 4.10** Show that $\vdash (p \to q) \to (\neg q \to \neg p)$.

Solution We show that $(p \to q) \vdash (\neg q \to \neg p)$; the rest is taken care of by the deduction theorem. In the proof below, we omit the line numbers altogether.

$\neg\neg p \to p$	Th
$p \to q$	P
$\neg\neg p \to q$	HS
$q \to \neg\neg q$	Th
$\neg\neg p \to \neg\neg q$	HS
$(\neg\neg p \to \neg\neg q) \to (\neg q \to \neg p)$	A3
$\neg q \to \neg p$	MP

***EXAMPLE* 4.11** Show that $\vdash p \to (\neg q \to \neg(p \to q))$.

Solution We can directly conclude from MP that

$$p, p \to q \vdash q$$

By the deduction theorem,

$$\vdash p \to ((p \to q) \to q)$$

We use this as a theorem in the following proof for $p \vdash \neg q \to \neg(p \to q)$:

p	P
$p \to ((p \to q) \to q)$	Th
$(p \to q) \to q$	MP
$((p \to q) \to q) \to (\neg q \to \neg(p \to q))$	Th
$\neg q \to \neg(p \to q)$	MP

***EXAMPLE* 4.12** Show that $\vdash (\neg q \to q) \to q$.

Solution Try to construct a proof before reading the one given below.

$\neg q \to (q \to \neg(\neg q \to q))$	Th
$(\neg q \to (q \to \neg(\neg q \to q))) \to ((\neg q \to q) \to (\neg q \to \neg(\neg q \to q)))$	A2
$(\neg q \to q) \to (\neg q \to \neg(\neg q \to q))$	MP
1. $\neg q \to q$	P
$\neg q \to \neg(\neg q \to q)$	MP
$(\neg q \to \neg(\neg q \to q)) \to ((\neg q \to q) \to q)$	A3
$(\neg q \to q) \to q$	MP
q	1,MP

This shows that $\neg q \to q \vdash q$. By the deduction theorem, we conclude that

$$\vdash (\neg q \to q) \to q$$

If you are observant, you would have got another proof of of the same theorem by employing the theorem proved in Example 4.11. Replacing p by $\neg q$ there, you obtain $\vdash \neg q \to (\neg q \to \neg(\neg q \to q))$. Using the deduction theorem thrice, you see that $\vdash \neg q \to \neg(\neg q \to q)$. Then by A3 and MP, you get $\vdash (\neg q \to q) \to q$. Now, construct a formal proof using this idea.

***EXAMPLE* 4.13** Show that $\{p \to q, \neg p \to q\} \vdash q$.

Solution This is the familiar 'argument by cases'. The proof below uses Example 4.12.

$p \to q$	P
$(p \to q) \to (\neg q \to \neg p)$	Th
$\neg q \to \neg p$	MP
$\neg p \to q$	P
$\neg q \to q$	HS
$(\neg q \to q) \to q$	Th
q	MP

So, you have had some experience in constructing proofs in PC. You must have noticed some generalities while doing proofs. Added to the deduction theorem, here is another metatheorem.

Theorem 4.2 (Monotonicity) *For sets Σ, Γ of propositions, and a proposition w, let $\Sigma \vdash w$. Then $\Gamma \supseteq \Sigma$ implies $\Gamma \vdash w$.*

Proof Does it require any proof? Note that PC does not insist on using all the available premises in a proof of any consequence. So, the very proof of the consequence $\Sigma \vdash w$ is also a proof of the consequence $\Gamma \vdash w$! ▮

We should also have *reductio ad absurdum* (RAA), if at all we are going parallel to the semantics of propositions. But RAA involves unsatisfiability, and PC does not have the concept of truth! We would like to remedy this by defining a syntactic notion parallel to unsatisfiability.

Definition 4.3 A set Σ of propositions is said to be **inconsistent** if for some proposition w, both the consequences $\Sigma \vdash w$ and $\Sigma \vdash \neg w$ are provable; otherwise Σ is said to be **consistent**.

Our hope is that unsatisfiability and inconsistency would match though we have not yet demonstrated it. With inconsistency in PC, we would like to have a formulation of RAA.

Theorem 4.3 (RAA) *Let Σ be a set of propositions and w be any proposition. Then $\Sigma \vdash w$ iff $\Sigma \cup \{\neg w\}$ is inconsistent.*

Proof Suppose that $\Sigma \vdash w$. By monotonicity, $\Sigma \cup \{\neg w\} \vdash w$. Also, trivially, $\Sigma \cup \{\neg w\} \vdash \neg w$. Therefore, $\Sigma \cup \{\neg w\}$ is inconsistent.

Conversely, suppose that $\Sigma \cup \{\neg w\}$ is inconsistent. Then there is a proposition, say p, such that $\Sigma \cup \{\neg w\} \vdash p$ and $\Sigma \cup \{\neg w\} \vdash \neg p$. By the deduction theorem, $\Sigma \vdash \neg w \to p$ and $\Sigma \vdash \neg w \to \neg p$. Then it is enough to show that

$$\{\neg w \to p, \neg w \to \neg p\} \vdash w$$

Here is a proof of this consequence:

$\neg w \to \neg p$	P
$(\neg w \to \neg p) \to (p \to w)$	A3
$p \to w$	MP
$\neg w \to p$	P
$\neg w \to w$	HS
$(\neg w \to w) \to w$	Th
w	MP

▮

Do you now see the rationale behind choosing the three axioms? The deduction theorem requires A1 and A2 while RAA uses A3 explicitly. But what about other connectives and propositional constants? We had only remarked that they could be introduced by definitions. For example, the following definitions would serve our purpose. We use the symbol $\mathrel{\hat=}$ for the expression 'equal to by definition'. As already discussed, definitions are, in fact, definition schemes.

(D1) $p \wedge q \mathrel{\hat{=}} \neg(p \to \neg q)$

(D2) $p \vee q \mathrel{\hat{=}} \neg p \to q$

(D3) $p \leftrightarrow q \mathrel{\hat{=}} (p \to q) \wedge (q \to p)$

(D4) $\top \mathrel{\hat{=}} p \to p$

(D5) $\bot \mathrel{\hat{=}} \neg(p \to p)$

We also require some inference rules to work with the definitions. They would provide us with ways of how to use the definitions. We have the two rules of definition, written as one, as follows:

(RD) $$\frac{X \mathrel{\hat{=}} Y \quad Z}{Z[X := Y]} \qquad\qquad \frac{X \mathrel{\hat{=}} Y \quad Z}{Z[Y := X]}$$

The rule(s) RD says that from $(p \vee q) \to r$ you can derive $(\neg p \to q) \to r$, and so on. This allows us to substitute expressions involving connectives other than $\neg$ and $\to$ with the ones having only these two. The definitions and the rule RD do the job of eliminating as also introducing the other connectives. Thus we lose nothing by limiting our logical constants to the two connectives $\neg$ and $\to$.

4.3 Axiomatic System FC

We have seen how an axiomatic system works. We now plan to extend PC to cover first order logic. As you have guessed, we call this system First Order Calculus (**FC**). The axioms (axiom schemes) of the system FC are A1, A2, A3 of PC, two more for the quantifier $\forall$, and two for the equality predicate $\approx$. We also add one more inference rule. The axioms are as follows:

Axiom schemes of FC: For formulas X, Y, Z, and variable x,

(A1) $X \to (Y \to X)$

(A2) $(X \to (Y \to Z)) \to ((X \to Y) \to (X \to Z))$

(A3) $(\neg X \to \neg Y) \to (Y \to X)$

(A4) $\forall x X \to X[x/t]$, for any term t free for x in X.

(A5) $\forall x(X \to Y) \to (X \to \forall x Y)$, where x does not occur free in X.

(A6) $(t \approx t)$, for every term t.

(A7) $(s \approx t) \to (X[x/s] \to X[x/t])$, for any terms s, t free for x in X.

Rules of Inference of FC:

(MP) $$\frac{X \quad X \to Y}{Y}$$

(UG) $\dfrac{X}{\forall x X}$ provided x is not free in any premise used thus far.

The alphabet of this system has the alphabet of FL without the propositional constants $\top, \bot$, and without the connectives $\wedge, \vee, \leftrightarrow$. We also exclude the quantifier $\exists$. To compensate, we have a definition:

(D6) $\exists x X \triangleq \neg \forall x \neg X$

As mentioned earlier, a **proof** is a finite sequence of formulas, each of which is either an axiom (an instance of an axiom scheme), or is obtained (derived) by an application of some inference rule on earlier formulas. Note that MP is applied on two formulas while UG is applied on a single formula. The last formula of a proof is a **theorem**; the proof is said to *prove* the theorem. The fact that a formula X is a theorem is written as $\vdash X$; in that case, we also say that the formula X is **provable**. We should have rather written this as $\vdash_{FC} X$ to document that X is a theorem in the axiomatic system FC. But our abbreviated notation $\vdash X$ will cause no confusion as all theorems of PC are also theorems of FC. For a set of formulas Σ, and a formula Y, the **proof of the consequence** $\Sigma \vdash Y$ is again a finite sequence of formulas, each of which is an axiom, or a premise (a formula) in Σ, or is derived from earlier formulas by an application of an inference rule; the last formula of the sequence is Y. The fact that there is a proof of the consequence $\Sigma \vdash Y$ is written simply as $\Sigma \vdash Y$; in that case, we also say that the consequence is **provable**. If $\Sigma = \{X_1, \ldots, X_n\}$, a finite set, instead of writing $\Sigma \vdash Y$, we just write $X_1, \ldots, X_n \vdash Y$. It is also clear that $\emptyset \vdash Y$ expresses the same fact as $\vdash Y$. A set of formulas Σ is said to be **inconsistent** if there exists a formula Y such that $\Sigma \vdash Y$ and $\Sigma \vdash \neg Y$, else Σ is said to be **consistent**. We also say that a formula X is inconsistent or consistent according as the set $\{X\}$ is inconsistent or consistent.

Before constructing proofs in FC, let us quickly go through the axioms and the inference rules. Axioms A1 to A3 are the same as in PC. A4 is taken from its semantic counterpart that $\models \forall X \to X[x/t]$. Similarly, A5; but why is there a condition with it? What unusual thing might happen if we do not impose the condition on the applicability of A5? For example, if $X = Pxy, Y = Qyx$, then A5 would read as

$$\forall x(Pxy \to Qyx) \to (Pxy \to \forall x Qyx)$$

Take an interpretation in the set of men, where Pxy means "x is a father of y", and Qyx means "y is a son of x". Then $\forall x(Pxy \to Qyx)$ means:

if for every man x, x is a father of y, then y is a son of x.

This, of course, holds whatever y may be. Now, $Pxy \to \forall x Qyx$ means:

if x is a father of y, then for every x, y is a son of x.

which does not hold. Obviously, we cannot choose an invalid formula as an axiom. This is the reason for the condition imposed on A5.

The axioms A6 and A7 are self-evident. The condition 'free for' is needed just for admissibility of substitutions. Note that the rule UG of universal generalization has a condition. It warns us to take care of each use of UG. We must first check that the variable on which we generalize must not have been free in any premise used up to that point in a proof. In a proof of a theorem, this warning becomes vacuous as there would not be any premise. The reason for this restriction will become clear if we consider the consequence $Px \vdash \forall x Px$. We know that its semantic counterpart, viz., $Px \models \forall x Px$, is not a valid consequence. Now, if the restriction on UG is removed, then we will be able to derive $\forall x Px$ from Px; this will be wrong. In fact, we want the relation $\vdash$ to match with $\models$ somehow. Let us see how the axiomatic system FC works.

***EXAMPLE* 4.14** Show that $\forall x X \vdash \forall x X$.

Solution This is not a new theorem in the sense that by using the axioms A1, A2, A3, you had already proved $p \vdash p$. Now, the same proof, with p replaced everywhere by $\forall x X$ gives the required proof. This points towards a general fact about FC. If you had already a theorem (or a provable consequence) in PC, by replacing the propositional variables with first order formulas uniformly, you will always land up in a theorem (or a provable consequence) in FC. However, you will have another proof of $\forall x X \vdash \forall x X$ by using quantifier rules. Here is one:

$\forall x X$	P
$\forall x X \to X$	A4
X	MP
$\forall x X$	UG

***Exercise* 4.4** Construct a proof of $\vdash \forall x X \to \forall x X$ different from that of $\vdash p \to p$ as done in PC.

***EXAMPLE* 4.15** The following is a proof of $\forall x \forall y X \vdash \forall y \forall x X$:

$\forall x \forall y X$	P
$\forall x \forall y X \to \forall y X$	A4
$\forall y X$	MP
$\forall y X \to X$	A4
X	MP
$\forall x X$	UG
$\forall y \forall x X$	UG

Look at the opening comments in the solution to Example 4.14. You can use all the PC-theorems as already proved theorems in constructing proofs in FC. In the following example, we use the PC-theorem

$$\vdash (p \to q) \to (\neg q \to \neg p)$$

after a suitable uniform replacement of the propositional variables by formulas of FC.

***EXAMPLE* 4.16** The following is a proof of $\forall x\neg X \vdash \neg\forall x X$:

	$\forall x\neg X$	P
	$\forall x\neg X \to \neg X$	A4
1.	$\neg X$	MP
	$\forall x X \to X$	A4
	$(\forall x X \to X) \to (\neg X \to \neg\forall x X)$	Th
	$\neg X \to \neg\forall x X$	MP
	$\neg\forall x X$	1, MP

Have you tried Exercise 4.4? If not, try it now. How to use the quantifier axioms and rules to construct a proof of $\vdash \forall x X \to \forall x X$? Let us try to construct a proof by taking help from the proof of $\forall x X \vdash \forall x X$ as done in Example 4.14. Now, $\forall x X \to X$ is an axiom. This can be universally generalized (by an application of UG) to get $\forall x(\forall x X \to X)$. Then, we can use A5 and MP to complete the proof. Do you see the trick? Since $\forall x X$ has no free occurrence of the variable x, we could do it. Here is the proof:

$\forall x X \to X$	A4
$\forall x(\forall x X \to X)$	UG
$\forall x(\forall x X \to X) \to (\forall x X \to \forall x X)$	A5
$\forall x X \to \forall x X$	MP

So, you have got a hint that, probably, the deduction theorem would work, is it? And once you show the truth of this metastatement, most of the tricks in constructing proofs would become unnecessary. In proving the deduction theorem, we adhere to the definition of a proof that already proved theorems have not been used in a (formal) proof.

Theorem 4.4 (Deduction Theorem) *Let Σ be a set of formulas and X, Y be any formulas. Then $\Sigma \vdash X \to Y$ iff $\Sigma \cup \{X\} \vdash Y$.*

Proof The proof, quite naturally, resembles that for PC. If $\Sigma \vdash X \to Y$, then there is a proof P whose last formula is $A \to B$. We add some more formulas to construct a proof of $\Sigma \cup \{A\} \vdash B$. It is as follows:

P:

$\vdots$

$A \to B$	$\Sigma \vdash A \to B$
A	Premise in $\Sigma \cup \{A\}$
B	MP

Conversely, suppose that P is a proof of $\Sigma \cup \{A\} \vdash B$. We prove $\Sigma \vdash A \to B$ by induction on the length of P, number of formulas (lines) in P. In the basis step, if `P` has only one formula, then it is B, and B is an axiom, a premise in Σ, or the formula A itself. If B is an axiom, then as in PC, we have the following proof of $\Sigma \vdash A \to B$:

$B \to (A \to B)$	A1
B	An axiom
$A \to B$	MP

If B is a premise in Σ, then also the above proof works; only that we mention B to be a premise in Σ instead of an axiom. If B is A itself, then we use the PC-theorem: $\vdash A \to A$ and proof of that works here as a proof of $\Sigma \vdash A \to B$.

Lay out the induction hypothesis that if `P` has less than n formulas, then there is a proof of $\Sigma \vdash A \to B$. Suppose that $\Sigma \cup \{A\} \vdash B$ has a proof `P1` of n formulas. Then the n-th formula is necessarily B. Now what can this B be? It may be

(i) an axiom,
(ii) a premise in Σ,
(iii) A itself,
(iv) derived from two earlier formulas by MP, or
(v) derived from an earlier formula by UG.

In each case, we construct a proof `P4` for $\Sigma \vdash A \to B$. Cases (i), (ii), and (iii) are covered in the basis step. In case (iv), proof `P1` looks like:

```
P1 : 1.          ...
     ⋮
     $n_1$.          $C$            for some formula $C$
     ⋮
     $n_1 + n_2$.    $C \to B$      MP
     ⋮
     $n$.            $B$            $n_1, n_1 + n_2$,MP
```

As $n_1, n_2 < n$, by the induction hypothesis we have proofs `P2, P3` of lengths m_1, m_2 for the consequences $\Sigma \vdash A \to C$, $\Sigma \vdash A \to (C \to B)$, respectively. We use these proofs to construct a proof `P4` for the consequence $\Sigma \vdash A \to B$. Add `P3` to `P2`, use A2 in the form

$$(A \to (C \to B)) \to ((A \to C) \to (A \to B))$$

Apply MP to conclude $(A \to C) \to (A \to B)$. This formula along with $A \to C$ give, by MP, the required conclusion. Try to construct `P4` now with these hints. It is as follows:

$$\begin{array}{llll}
\texttt{P4}:\texttt{P2}: & 1. & & \\
 & \vdots & & \\
 & m_1. & A \to C & \\
\texttt{P3}: & m_1+1. & & \\
 & \vdots & & \\
 & m_1+m_2. & A \to (C \to B) & \\
 & m_1+m_2+1. & (A \to (C \to B)) \to ((A \to C) \to (A \to B)) & \text{A2} \\
 & m_1+m_2+2. & (A \to C) \to (A \to B) & \text{MP} \\
 & m_1+m_2+3. & A \to B & m_1, \text{MP}
\end{array}$$

In case (v), $B = \forall x X$ for some variable x and some formula X. Then the proof P1 of $\Sigma \cup \{A\} \vdash B$ looks like:

$$\begin{array}{llll}
\texttt{P1}: & 1. & \ldots & \\
 & \vdots & & \\
 & k_1. & X & \\
 & \vdots & & \\
 & n. & \forall x X & \Sigma \cup \{A\} \vdash \forall x X
\end{array}$$

If the premise A has not been used in P1, then this proof itself is a proof of $\Sigma \vdash B$. Then using A1 as in the basis step, we get a proof of $\Sigma \vdash A \to B$. (Do it.) More realistically, suppose that A has been used somewhere in P1 and that UG has been applied on the k_1-th formula X to derive the n-th formula $\forall x X$. Then, due to the restriction on the applicability of UG, the variable x does not occur in any premise (in Σ) used in P1. In particular, x is not free in A. (We use this in two crucial situations in the proof below while using UG and A5.) By the induction hypothesis, since $k_1 < n$, there is a proof P5 of length j for $\Sigma \vdash A \to X$. We use P5 to construct the proof P4 for $\Sigma \vdash A \to \forall x X$ as follows:

$$\begin{array}{llll}
\texttt{P4}:\texttt{P5}: & 1. & \ldots & \\
 & \vdots & & \\
 & j. & A \to X & \\
 & j+1. & \forall x(A \to X) & \text{UG} \\
 & j+2. & \forall x(A \to X) \to (A \to \forall x X) & \text{A5} \\
 & j+3. & A \to \forall x X & \text{MP}
\end{array}$$

As $B = \forall x X$, P4 is a proof for $\Sigma \vdash A \to B$. This completes the proof. ∎

It is instructive to take help from the deduction theorem while constructing proofs. In particular, it is often advantageous to use the 'derived rules' which may be obtained by applying the deduction theorem on the axioms. For example, A1, written as a theorem, $\vdash X \to (Y \to X)$, by the deduction theorem, proves the consequence: $\{X, Y\} \vdash X$ and then it gives the derived rule:

From X and Y derive X.

The derived rule corresponding to A7 will be very helpful. We write it now as the inference **rule of equality**:

$$\textbf{(EQ)} \qquad \frac{s \approx t \quad X[x/s]}{X[x/t]}$$

***Exercise* 4.5** Write all the axioms as rules of inference by using the deduction theorem.

Let us see some more examples of theorems, consequences, and proofs.

***EXAMPLE* 4.17** Show that $\vdash \forall x(X \to Y) \to (\neg\forall x \neg X \to \neg\forall x \neg Y)$.

Solution Due to the deduction theorem, it is enough to show that

$$\{\forall x(X \to Y), \neg\forall x\neg X\} \vdash \neg\forall x \neg Y$$

How do we proceed to show this? Suppose we start with $\forall x(X \to Y)$. We would get $X \to Y$. Then how to use $\neg\forall x\neg X$? The symbol $\neg$ blocks further derivations. But you have seen already that it is easier to prove $A \to B$ than $\neg B \to \neg A$. You can, of course, bring $\neg B \to \neg A$ whenever needed by using a PC-theorem. So we try the consequence: $\forall x(X \to Y) \vdash \forall x\neg Y \to \forall x \neg X$.

Hence, direct double application of the deduction theorem would have misled us! But now, we can again use the deduction theorem to prove the consequence: $\forall x(X \to Y), \forall x\neg Y \vdash \forall x\neg X$. Here is such a proof:

	$\forall x(X \to Y)$	P
	$\forall x(X \to Y) \to (X \to Y)$	A4
	$X \to Y$	MP
	$(X \to Y) \to (\neg y \to \neg X)$	Th
1.	$\neg Y \to \neg X$	MP
	$\forall x\neg Y$	P
	$\forall x\neg Y \to \neg Y$	A4
2.	$\neg Y$	MP
	$\neg X$	1,2,MP
	$\forall x \neg X$	UG

Having proved $\forall x(X \to Y), \forall x\neg Y \vdash \forall x\neg X$, we argue as follows: By the deduction theorem,

$$\forall x(X \to Y) \vdash \forall x\neg Y \to \forall x\neg X$$

Since $\vdash (\forall x\neg Y \to \forall x\neg X) \to (\neg\forall x\neg X \to \neg\forall x\neg Y)$, by MP, we have

$$\forall x(X \to Y) \vdash \neg\forall x\neg X \to \neg\forall x\neg Y$$

Again, by the deduction theorem, we obtain

$$\vdash \forall x(X \to Y) \to (\neg\forall x\neg X \to \neg\forall x\neg Y)$$

You probably did not like the roundabout way of arguing over the proof above. This argument can be shortened if you have *reductio ad absurdum* at your disposal. With the notion of inconsistency as introduced earlier (now for formulas and not only for propositions), we formulate RAA.

Theorem 4.5 (RAA) *Let Σ be a set of formulas and Y be any formula. Then $\Sigma \vdash Y$ iff $\Sigma \cup \{\neg Y\}$ is inconsistent.*

Proof Repeat the argument of the proof of RAA in PC (Theorem 4.3) by taking formulas instead of propositions. ∎

***EXAMPLE* 4.18** Show that $\vdash \forall x(X \to Y) \to (\neg\forall x\neg X \to \neg\forall x\neg Y)$.

Solution This is the same formula of Example 4.17; but here we want to use RAA. Our aim is to show that the set

$$\Sigma = \{\forall x(X \to Y), \neg\forall x\neg X, \neg\neg\forall x\neg Y\}$$

is inconsistent. Here is such a proof:

	$\neg\forall x\neg X$	P
	$\forall x(X \to Y)$	P
	$\forall x(X \to Y) \to (X \to Y)$	A4
	$X \to Y$	MP
	$(X \to Y) \to (\neg Y \to \neg X)$	Th
1.	$\neg Y \to \neg X$	MP
	$\neg\neg\forall x\neg Y$	P
	$\neg\neg\forall x\neg Y \to \forall x\neg Y$	Th
	$\forall x\neg Y$	MP
	$\forall x\neg Y \to \neg Y$	A4
2.	$\neg Y$	MP
	$\neg X$	1,2,MP
	$\forall x\neg X$	UG

The first formula in the above proof is the negation of the last formula. That is, this proof proves $\forall x\neg X$ as well as $\neg\forall x\neg X$ as consequences of Σ. Thus Σ is inconsistent.

***EXAMPLE* 4.19** If x is not a free variable of Y, then show that

$$\vdash \neg(\forall xX \to Y) \to \forall x\neg(X \to Y)$$

Solution Using the deduction theorem and RAA, we first transfer the theorem to a convenient consequence in the following manner:

$\vdash \neg(\forall xX \to Y) \to \forall x\neg(X \to Y)$

iff $\neg(\forall xX \to Y) \vdash \forall x\neg(X \to Y)$

iff $\{\neg(\forall xX \to Y), \neg\forall x\neg(X \to Y)\}$ is inconsistent.

iff $\neg\forall x\neg(X \to Y) \vdash \forall xX \to Y$

iff $\{\neg\forall x\neg(X \to Y), \forall xX\} \vdash Y$

iff $\{\neg\forall x\neg(X \to Y), \forall xX, \neg Y\}$ is inconsistent.

iff $\{\forall xX, \neg Y\} \vdash \forall x\neg(X \to Y)$

This consequence has the following proof:

$\forall xX$	P
$\forall xX \to X$	A4
X	MP
$X \to (\neg Y \to \neg(X \to Y))$	Th
$\neg Y \to \neg(X \to Y)$	MP
$\neg Y$	P
$\neg(X \to Y)$	MP
$\forall x\neg(X \to Y)$	UG

***EXAMPLE* 4.20** Show that $\vdash \forall x((x \approx f(y)) \to Qx) \to Qf(y)$.

Solution We show that $\forall x((x \approx f(y)) \to Qx) \vdash Qf(y)$.

$\forall x((x \approx f(y)) \to Qx)$	P
$\forall x((x \approx f(y)) \to Qx) \to ((f(y) \approx f(y)) \to Qf(y))$	A4
$((f(y) \approx f(y)) \to Qf(y)$	MP
$f(y) \approx f(y)$	A6
$Qf(y)$	

***EXAMPLE* 4.21** $\{Pa, \forall x(Px \to Qx), \forall x(Rx \to \neg Qx), R\} \vdash \neg(a \approx b)$.

Solution It looks easier to use RAA. We add the negation of the conclusion to the premises and go for proving inconsistency, i.e., we show that the set

$$\Sigma = \{Pa, \forall x(Px \to Qx), \forall x(Rx \to \neg Qx), Rb, \neg\neg(a \approx b)\}$$

is inconsistent. The proof is as follows:

	$\forall x(Px \to Qx)$	P
	$\forall x(Px \to Qx) \to (Pa \to Qa)$	A4
	$Pa \to Qa$	MP
	Pa	P
1.	Qa	MP

	$\neg\neg(a \approx b)$	P
	$\neg\neg(a \approx b) \to (a \approx b)$	Th
	$a \approx b$	MP
	$(a \approx b) \to (Qa \to Qb)$	A7
	$Qa \to Qb$	MP
2.	Qb	1, MP
	$\forall x(Rx \to \neg Qx)$	P
	$\forall x(Rx \to \neg Qx) \to (Rb \to \neg Qb)$	A4
	$Rb \to \neg Qb$	MP
	Rb	P
3.	$\neg Qb$	MP

Due to formulas numbered 2 and 3 in the above proof, the set Σ is inconsistent. Such a proof where RAA is used is, in fact, a proof containing two proofs; one proves a formula X and the other proves the formula $\neg X$. You can also construct the two proofs separately if it is convenient. But then you can also construct a single proof with both X and $\neg X$ occurring in it without much difficulty.

***Exercise* 4.6** You can see that the the proof in Example 4.21 could be shortened using the derived rule EQ. Give such a proof which uses EQ instead of A7, and without using RAA.

***Exercise* 4.7** Show the interchange of variables: $\vdash \forall x\forall y X \to \forall y\forall x X$.

Complete the two exercises first, then read further. We use the second one as an already proved theorem in the following example.

***EXAMPLE* 4.22** Show that the following consequence is provable:

$$\forall x\forall y(f(x,y) \approx f(y,x)), \forall x\forall y(f(x,y) \approx y) \vdash \neg\forall x\neg\forall y(x \approx y)$$

Solution There are gaps in the following proof. Read it and rewrite the complete proof without omission of any step. Look at the lines 12 and 17. They show that Σ is inconsistent.

1.	$\forall x\forall y(f(x,y) \approx y)$	P
2.	$\forall x\forall y(f(x,y) \approx y) \to \forall y\forall x(f(y,x) \approx x)$	Th
3.	$\forall y\forall x(f(y,x) \approx x)$	MP
4.	$f(x,y) \approx y$	1, A4, MP
5.	$f(y,x) \approx x$	3, A4, MP
6.	$\forall x\forall y(f(x,y) \approx f(y,x))$	P
7.	$f(x,y) \approx f(y,x)$	A4, MP
8.	$y \approx f(y,x)$	4, 7, EQ

9. $y \approx x$	5,8,EQ
10. $x \approx x$	A6
11. $x \approx y$	9, 10, EQ
12. $\forall y(x \approx y)$	UG
13. $\neg\neg\forall x\neg\forall y(x \approx y)$	P
14. $\neg\neg\forall x\neg\forall y(x \approx y) \to \forall x\neg\forall y(x \approx y)$	Th
15. $\forall x\neg\forall y(x \approx y)$	MP
16. $\forall x\neg\forall y(x \approx y) \to \neg\forall y(x \approx y)$	A4
17. $\neg\forall y(x \approx y)$	MP

Recollect that the restriction on the rule UG does not allow you to show that $Px \vdash \forall xPx$. Semantically also, $Px \not\models \forall xPx$, as you have already seen. As far as the deduction theorem is concerned, you have seen that the semantic consequence relation $\models$ and the provability relation $\vdash$ go hand in hand. However, if you drop the restriction on UG, then you can have a proof of the metastatement $Px \vdash \forall xPx$. Then, the concepts of truth and provability will certainly differ, and the metastatement $Px \vdash \forall xPx$ has to be read as "If $\models Px$, then $\models \forall xPx$", and not as "$Px \models \forall xPx$". This is all right for mathematics since all assumptions in mathematics are sentences or closed formulas. But the unrestricted use of UG will force the deduction theorem to take a different shape, like

> "If there is a proof of $\Sigma \cup \{A\} \vdash B$, where UG has not been applied on a free variable of A, then $\Sigma \vdash A \to B$."

It then follows that the metastatement "$\Sigma \cup \{A\} \vdash B$ iff $\Sigma \vdash A \to B$" holds for closed formulas, which would be sufficient for proving theorems in mathematics. But in computer science, we do require to argue with open formulas, formulas with free variables. For example, the state of a program can be described by 'which variable is bound to what value', and such a description will result in a formula with free variables. This is the reason we have adopted a version of UG with a restriction. You should be able to read other texts where UG comes with no restrictions. In that case, you must also look for a restriction on the deduction theorem. Note that the restriction as mentioned above does not require every proof of $\Sigma \cup \{A\} \vdash B$ to satisfy the free-variable condition. It only requires one such proof.

***Exercise* 4.8** Show that $\{Px, \forall xQx\} \vdash \forall x(Px \to \forall xQx)$.

If you complete Exercise 4.8, you would realize that having a restriction on UG or not does not affect monotonicity. But, if you restrict UG too much, e.g., by allowing UG to be applicable only when the variable x is not free in *any* premise, then probably, monotonicity will hold no longer. Try to see the content of this observation with examples and with an attempt to prove monotonicity when this new restriction is imposed on UG.

4.4 Adequacy and Compactness

Why have we chosen exactly those axioms and inference rules in PC and FC? We could have chosen others as well. One answer could be that the proof of the deduction theorem required these axioms; there could be other similar reasons too.

Well, what happens if we drop one of them? Then, of course, we cannot prove many theorems. So what? Anyway, we are not planning to prove that every formula is a theorem! But something drastic might happen if you drop an axiom or an inference rule. The fact is that the one you drop cannot be proved by using others. Your system becomes deficient. Because, as we know, each of our axioms is a valid formula and each inference rule represents a valid consequence. Therefore, if you drop one, then at least, that one is valid but cannot be proved. There will be a clear mismatch between validity and provability. Note that this argument rests on the fact that no axiom or a rule of inference in PC and FC can be proved from the remaining; that there is no redundancy (see Problem 5).

What about adding one more valid proposition or a valid formula as an axiom? It is also a fact that this will not affect the systems in any significant way, because every valid proposition has a proof in PC and every valid formula has a proof in FC. This property of a system is known as **completeness**. Conversely, if you find a proof of a proposition or a formula, then it cannot be invalid. This property of a system is known as **soundness**. Thus the systemic (often called syntactic) notion of provability and the semantic notion of validity match nicely, i.e., both the systems PC and FC are adequate to the logics PL and FL, respectively. **Adequacy** means both soundness and completeness. In this section we try to show adequacy of PC and FC, and then use it to deduce another property of the logics PL and FL. The adjective 'strong' is used with these notions to mark that not only theoremhood and validity of formulas match but provability and validity of consequences also match. We start with strong soundness. The symbol $\vdash$ stands for '$\vdash$ in PC' for propositions and for '$\vdash$ in FC' for formulas; similarly, the symbol $\models$.

Theorem 4.6 (Strong Soundness of PC and FC) *Let Σ be a set of propositions (formulas) and A be a proposition (formula). If $\Sigma \vdash A$, then $\Sigma \models A$.*

Proof All that you have to do is to check that the axioms are valid and the rules of inference are valid consequences, and then apply induction on the lengths of proofs.

Let $\mathtt{P}$ be a proof of $\Sigma \vdash A$. In the proof $\mathtt{P}$, all the premises in Σ might not have been used. Let $\Sigma_{\mathtt{P}}$ be the set of premises that have been actually used in $\mathtt{P}$. Then $\Sigma_{\mathtt{P}}$ is a finite subset of Σ and $\Sigma_{\mathtt{P}} \vdash A$. We will prove by induction on n, the number of propositions or formulas in $\mathtt{P}$ that proves $\Sigma_{\mathtt{P}} \models A$. By monotonicity in PL (or FL), it will follow that $\Sigma \models A$.

In the basis step, $n = 1$; A is either an axiom or a premise in $\Sigma_{\mathtt{P}}$. Clearly, $\Sigma_{\mathtt{P}} \models A$. Lay out the induction hypothesis that if $\Sigma_{\mathtt{P}} \vdash A$ has a proof of less than m propositions (formulas), then $\Sigma_{\mathtt{P}} \models A$. Let P1 be a proof of $\Sigma_{\mathtt{P}} \vdash A$ having m propositions (formulas). If A is again an axiom or a premise in $\Sigma_{\mathtt{P}}$, then clearly $\Sigma_{\mathtt{P}} \models A$ holds. Otherwise, (a) A has been obtained in P1 by an application of MP or (b) A has been obtained by an application of UG in case A is a formula.

In case (a), there are propositions or formulas B and $B \to A$ occurring earlier to A in P1. By the induction hypothesis, $\Sigma_{\mathtt{P}} \models B$ and $\Sigma_{\mathtt{P}} \models B \to A$. Since $\{B, B \to A\} \models A$, we have $\Sigma_{\mathtt{P}} \models A$. This, in fact, completes the strong soundness proof of PC.

In case (b), there is a formula C occurring prior to A in P1 such that $A = \forall x C$, for some variable x. Further, let $\Sigma_{\mathtt{C}}$ be the subset of $\Sigma_{\mathtt{P}}$ containing exactly those formulas which have been used in P1 in deriving C. Then the variable x does not occur free in any formula of $\Sigma_{\mathtt{C}}$ due to the restriction on applicability of UG. By the induction hypothesis, $\Sigma_{\mathtt{C}} \models C$. It follows, by the semantics of FL, that $\Sigma_{\mathtt{C}} \models \forall x C$ (Prove it.), i.e., $\Sigma_{\mathtt{C}} \models A$. Since $\Sigma_{\mathtt{C}} \subseteq \Sigma$, by monotonicity, $\Sigma \models A$. ∎

***Exercise* 4.9** Let Σ be a finite set of formulas and A be a formula. If $\Sigma \models A$ and x is a variable not occurring free in any formula of Σ, then show that $\Sigma \models \forall x A$.

Using RAA, the strong soundness of PC or of FC can be rewritten by connecting satisfiability and consistency. Due to RAA in PC (FC), $\Sigma \vdash A$ iff $\Sigma \cup \{\neg A\}$ is inconsistent. Similarly, due to RAA in PL (FL), $\Sigma \models A$ iff $\Sigma \cup \{\neg A\}$ is unsatisfiable. Hence the strong soundness says that if $\Sigma \cup \{\neg A\}$ is inconsistent, then it is unsatisfiable. Or, by contraposition, if $\Sigma \cup \{\neg A\}$ is satisfiable, then it is consistent. Now, if Γ is any set not necessarily in the form $\Sigma \cup \{\neg A\}$, then does the same metastatement hold? The following theorem gives an affirmative answer to this question. We again name the result as Strong Soundness since it is simply a restatement of Theorem 4.6.

Theorem 4.7 (Strong Soundness of PC and FC) *Let Σ be a nonempty set of propositions (formulas). If Σ is satisfiable, then Σ is consistent.*

Proof Let $\Sigma \neq \emptyset$ be satisfiable. Suppose, on the contrary, that Σ is inconsistent. As Σ is nonempty, there is a proposition (formula) $A \in \Sigma$. Let $\Sigma_0 = \Sigma - \{A\}$. Since $\vdash A \to \neg\neg A$ and $\vdash \neg\neg A \to A$, the set $\Sigma_0 \cup \{\neg\neg A\}$ is inconsistent. (Why?) By RAA in PC (FC), $\Sigma_0 \vdash \neg A$. Theorem 4.6 gives $\Sigma_0 \models \neg A$. By RAA in PL (FL), $\Sigma_0 \cup \{\neg\neg A\}$ is unsatisfiable. As $\neg\neg A \equiv A$, $\Sigma_0 \cup \{A\}$ is unsatisfiable. This contradicts the assumption that Σ is satisfiable. Therefore, Σ is consistent. ∎

***Exercise* 4.10** Show that $\Sigma \cup \{A\}$ is consistent iff $\Sigma \cup \{\neg\neg A\}$ is consistent.

Next, we look at strong completeness of the calculi PC and FC. To understand the proof better, we will first try PC and then attempt FC. Similar to strong soundness, we can formulate strong completeness in two ways: one, by connecting validity and provability, and two, by connecting satisfiability and consistency. Let us try the second alternative:

Strong completeness: If Σ is consistent, then Σ is satisfiable.

To feel what is going on, suppose that Σ is a singleton, say, $\Sigma = \{p\}$. Of course, we can have a truth assignment, say, i such that $i(p) = 1$ so that Σ is satisfiable. But something else happens when we think of PC. By A1 and MP, $p \vdash q \rightarrow p$. This says that $\Sigma \vdash q \rightarrow p$ holds. Fine, this is all right, since when $i(p) = 1$, we also have $i(q \rightarrow p) = 1$. But then, this opens up infinite possibilities. Just by applying axioms and inference rules of PC, an infinite number of propositions can be derived from a single premise. We now show that whatever that can be derived in PC from p is assigned to 1 by i. Can we show this?

One way is to define the set of all conclusions of Σ, call it the deductive closure of Σ, and then show that the set is, indeed, satisfiable. Clearly, $q \rightarrow p$ becomes an element of the deductive closure of $\{p\}$. Then $\neg(q \rightarrow p)$ cannot be a member of this deductive closure as we think that the deductive closure is consistent. (We have not proved it yet!) The second approach comes from this observation that our attempt is directed towards showing consistency of this deductive closure, so why not extend Σ to a consistent set, and to one, which can no more be extended? This is what we want to do next; we define a maximal consistent extension of Σ and then show that this is satisfiable so that Σ as a subset of it becomes satisfiable.

Since the set of all propositions (now, without $\vee, \wedge, \leftrightarrow, \top, \bot$) is countable, we can have an enumeration of it. That is, let the set of all propositions be listed member by member, as

$$q_0, q_1, q_2, \ldots, q_n, \ldots$$

Let Σ be a consistent set of propositions. Define a sequence of sets of propositions Σ_n inductively by

(A) $\Sigma_0 = \Sigma$;

(B1) $\Sigma_{n+1} = \Sigma_n$, if $\Sigma_n \cup \{q_n\}$ is inconsistent;

(B2) $\Sigma_{n+1} = \Sigma_n \cup \{q_n\}$ if $\Sigma_n \cup \{q_n\}$ is consistent.

Then, let $\Sigma' = \cup_{n \in \mathbb{N}} \Sigma_n$. The following lemma lists some of the interesting properties of Σ'.

Lemma 4.8 *Let Σ be a consistent set of propositions and Σ' be the set as constructed above. Let p, q be any propositions. Then the following hold:*

(a) Σ' is consistent.

(b) Σ' is maximally consistent.

(c) Either $q \in \Sigma'$ or $\neg q \in \Sigma'$.

(d) $q \in \Sigma'$ iff $\Sigma' \vdash q$.

(e) If $q \in \Sigma'$, then $p \rightarrow q \in \Sigma'$.

(f) If $p \notin \Sigma'$, then $p \to q \in \Sigma'$.

(g) If $p \in \Sigma$ and $q \notin \Sigma'$, then $p \to q \notin \Sigma'$.

The properties (a) and (b) above are the basic properties of Σ'. These are used to prove other properties. Properties in (c), (e), (f), and (g) essentially capture the semantics of the connectives $\neg$ and $\to$. Property (d) says that the set Σ' is its own deductive closure.

Proof (a) By definition, each Σ_n is consistent. Now, if Σ' is inconsistent, then $\Sigma' \vdash r$ and $\Sigma' \vdash \neg r$ for some proposition r. However, the proofs of both $\Sigma' \vdash r$ and $\Sigma' \vdash \neg r$ contain only finite number of premises from Σ'. Thus, there is an $m \in \mathbb{N}$ such that $\Sigma_m \vdash r$ and $\Sigma_m \vdash \neg r$. This contradicts the fact that each Σ_m is consistent, completing the proof of (a).

(b) Σ' is **maximally consistent** means that it is consistent, and if we add any other proposition to it, then the resulting set would become inconsistent. Since (a) has already been shown, we only show that for any proposition q, if $q \notin \Sigma'$ then $\Sigma' \cup \{q\}$ is inconsistent. For this, suppose that q is a proposition not in Σ' and that $\Sigma' \cup \{q\}$ is consistent. Due to the enumeration $q_0, q_1, q_2, \ldots$ of the set of all propositions, $q = q_m$ for some $m \in \mathbb{N}$. Since $\Sigma' \cup \{q\}$ is consistent and $\Sigma_m \cup \{q\} \subseteq \Sigma' \cup \{q\}$, $\Sigma_m \cup \{q\}$ is consistent. This implies that $q = q_m \in \Sigma_{m+1} \subseteq \Sigma'$, contradicting the fact that $q \notin \Sigma'$. Thus $\Sigma' \cup \{q\}$ is inconsistent.

(c) Let q be any proposition. Suppose that neither $q \in \Sigma'$ nor $\neg q \in \Sigma'$. Since Σ' is maximally consistent, and $q \notin \Sigma'$, the set $\Sigma' \cup \{q\}$ is inconsistent. Similarly, $\Sigma' \cup \{\neg q\}$ is also inconsistent. By RAA (and Exercise 4.10), $\Sigma' \vdash \neg q$ and $\Sigma' \vdash q$. This forces Σ' to be inconsistent; which is not the case. Thus, at least one of q or $\neg q$ must be in Σ'. Now, both of them cannot be in Σ' since Σ' is consistent. Hence, only one of q or $\neg q$ is in Σ'.

(d) Suppose that $q \in \Sigma'$. Then $\neg q \notin \Sigma'$ by (c). By (b), $\Sigma' \cup \{\neg q\}$ is inconsistent. By RAA, $\Sigma' \vdash q$. Conversely, suppose that $q \notin \Sigma'$. By (b), $\Sigma' \cup \{q\}$ is inconsistent. Exercise 4.10 shows that $\Sigma' \cup \{\neg\neg q\}$ is inconsistent. by RAA, $\Sigma' \vdash \neg q$. Since Σ' is consistent, $\Sigma' \nvdash q$.

(e) Let $q \in \Sigma'$. By (d), $\Sigma' \vdash q$. With A1 as $\vdash q \to (p \to q)$ and MP, we see that $\Sigma' \vdash p \to q$. Again by (d), $p \to q \in \Sigma'$.

(f) Let $p \notin \Sigma'$. By (c), $\neg p \in \Sigma'$. By (d), $\Sigma' \vdash \neg p$. By (e), $\Sigma' \vdash \neg q \to \neg p$. With A3 as $\vdash (\neg q \to \neg p) \to (p \to q)$ and MP, we obtain: $\Sigma' \vdash p \to q$.

(g) Let $p \in \Sigma'$ and $q \notin \Sigma'$. By (c) and (d), $\Sigma' \vdash p$ and $\Sigma' \vdash \neg q$. Since $\vdash p \to (\neg q \to \neg(p \to q))$, (see Example 4.11), using MP twice we see that $\Sigma' \vdash \neg(p \to q)$. By (c) and (d), $p \to q \notin \Sigma'$. ∎

Any set having the properties (a) to (g) listed in Lemma 4.8 is called a **Hintikka set** after the logician J. Hintikka. By Lemma 4.8, we have simply shown the existence of a Hintikka set which is an extension of a given consistent set Σ. The fact that any consistent set can be extended to a maximally consistent set is referred to as the **Lindenbaum lemma**. The following metatheorem shows that every consistent set is satisfiable.

Theorem 4.9 (Model Existence Theorem for PC) *Every consistent set of propositions has a model.*

Proof Let Σ be a consistent set of propositions. Let Σ' be the maximally consistent set of Lemma 4.8. Let S be the set of all propositions. Define a function $i : S \to \{0, 1\}$ by $i(p) = 1$ if $p \in \Sigma'$, else, $i(p) = 0$. This is well defined due to the property (c) of Σ'. Now, we see that i is indeed a truth assignment due to the properties (c), (f) and (g) listed in Lemma 4.8. Obviously, i is a model of Σ'. Since $\Sigma \subseteq \Sigma'$ i is a model of Σ as well. ∎

The model existence theorem says that every consistent set is satisfiable, or that every unsatisfiable set is inconsistent. By RAA in both PC and PL, we then obtain the following result.

Theorem 4.10 (Strong Completeness of PC to PL) *For any set Σ of propositions and any proposition w, if $\Sigma \models w$ then $\Sigma \vdash w$.*

Combining the metatheorems of strong soundness (Theorem 4.7) and strong completeness (Theorem 4.10), we obtain the following theorem.

Theorem 4.11 (Strong Adequacy of PC to PL) *For any set Σ of propositions and any proposition w, $\Sigma \vdash w$ iff $\Sigma \models w$.*

As corollaries (with $\Sigma = \emptyset$) to the strong adequacy of PC with respect to PL, we obtain the following theorem.

Theorem 4.12 (Adequacy of PC) *Let w be any proposition. Then the following hold:*

(a) Soundness of PC: *If $\vdash w$, then $\models w$.*

(b) Completeness of PC: *If $\models w$, then $\vdash w$.*

(c) Adequacy of PC: *$\vdash w$ iff $\models w$.*

An important corollary to the completeness theorem is the following.

Theorem 4.13 (Compactness of PL) *Let Σ be any nonempty set of propositions and w be any proposition. Then the following hold:*

(a) If $\Sigma \models w$, then $\Sigma_0 \models w$ for some finite subset Σ_0 of Σ.

(b) If Σ is unsatisfiable, then there is a finite subset Σ_0 of Σ such that Σ_0 is unsatisfiable.

(c) If each nonempty finite subset of Σ is satisfiable, then Σ is satisfiable.

Proof (a) Suppose that $\Sigma \models w$. By the strong completeness of PC, $\Sigma \vdash w$. Thus, we have a proof whose last proposition is w and in which some or all propositions in Σ have been used. However, the proof is a finite sequence of propositions. Thus, it uses only a finite number of propositions from Σ. Let Σ_0 be the set of all such propositions which are used in the proof. Then Σ_0 is a finite set and the same proof shows that $\Sigma_0 \vdash w$. By the strong soundness of PC, $\Sigma_0 \models w$. The statements in (b) and (c) are only restatements of the statement in (a). ∎

Compactness of PL could have been proved directly by means of models also; see the summary to this chapter. However, waiting this far paid off; we could avoid many difficult constructions.

***Exercise* 4.11** Formulate and prove the compactness theorem for FL by using the compactness for PL (Theorem 4.13) and the syntactic interpretation theorem (Theorem 2.20).

Now, we turn towards adequacy of FC with respect to FL. We have already discussed the strong soundness of FC in Theorem 4.7. We can also restate it as "if Σ is inconsistent, then Σ is unsatisfiable". We would then try the completeness of FC with respect to FL. The proof is, of course, similar to that of PC, but the construction of a maximally consistent set is a bit complicated. We start with a formula X having a free occurrence of a variable x.

Let $\mathcal{T}$ be the set of all closed terms (terms having no variables), i.e., $\mathcal{T} = \{t : t \text{ is a closed term}\}$. The set $\mathcal{T}$ is sometimes called the **free universe**. Let $\mathcal{A} = \mathcal{A}_x = \{\forall xY : Y \text{ is a formula having a free variable } x\}$.

Since we have a countable alphabet, the sets $\mathcal{T}$ and $\mathcal{A}$ are countable. Let us then write them as

$$\mathcal{T} = \{t_0, t_1, t_2, \ldots, \}, \ \mathcal{A} = \{\forall xX_0, \forall xX_1, \forall xX_2, \ldots\}$$

Let Σ be a consistent set of formulas. We define a sequence Σ_m inductively by

$$\Sigma_0 = \Sigma, \ \Sigma_{m+1} = \Sigma_m \cup \{X_m[x/t_{m1}] \to \forall xX_m\}$$

where t_{m1} is the first element of $\mathcal{T}$ that does not occur in the first m formulas of $\mathcal{A}$. We show, by induction, that each such set Σ_m is consistent.

Lemma 4.14 *Each Σ_m is consistent.*

Proof (Outline): If Σ_{m+1} is inconsistent, but Σ_m is consistent, then from the definition of Σ_{m+1}, it follows that (why?)

$$\Sigma_m \vdash \neg(X_m[x/t_{m1}] \to \forall xX_m) \tag{1}$$

Let y be a new variable which neither occurs in any formula of Σ_m nor in $\forall xX_m$. Since the term t_{m1} does not occur in any formula of Σ_m and not in $\forall xX_m$, we obtain (why?): $\Sigma_m \vdash X_m[x/t_{m1}] \to X_m[x/y]$. By UG, we see that $X_m[x/y] \vdash \forall yX_m[x/y]$. But renaming gives: $\forall yX_m[x/y] \vdash \forall xX_m$. Hence,

$$\Sigma_m \vdash X_m[x/t_{m1} \to \forall xX_m \tag{2}$$

Now, (1) and (2) would show that Σ_m is inconsistent, contradicting the assumption that Σ_m is consistent. This contradiction shows that consistency of Σ_m implies that of Σ_{m+1} completing the outline. [Expand the outline to a proof.] ∎

However, the construction is not yet over; we have used a very special kind of formulas in defining Σ_m's. We start with the set of all such special formulas, i.e., the union of all these Σ_m's and extend this set in a manner analogous to PC. To this end, we use the countability of formulas of FC.

Let $Y_0, Y_1, Y_2, \ldots$ be an enumeration of all formulas of FC. Define the sequence $\Gamma_0, \Gamma_1, \Gamma_2, \ldots$ inductively by

(a) $\Gamma_0 = \cup_{n\in\mathbb{N}}\Sigma_n$,

(b1) $\Gamma_{n+1} = \Gamma_n$, if $\Gamma_n \cup \{Y_n\}$ is inconsistent, and

(b2) $\Gamma_{n+1} = \Gamma_n \cup \{Y_n\}$, if $\Gamma_n \cup \{Y_n\}$ is consistent.

Finally, take $\Sigma' = \cup_{n\in\mathbb{N}}\Gamma_n$. This is the required Hintikka set corresponding to the given set Σ.

Lemma 4.15 *Let Σ be a consistent set of formulas and Σ' be the set obtained from Σ as constructed above. Let X, Y be any formulas and s, t be any terms. Then the following hold:*

(a) $\Sigma \subseteq \Sigma'$.

(b) Σ' is maximally consistent.

(c) If $\Sigma' \vdash X$, then $X \in \Sigma'$.

(d) Either $X \in \Sigma'$ or $\neg X \in \Sigma'$.

(e) If $Y \in \Sigma'$, then $X \to Y \in \Sigma'$.

(f) If $X \notin \Sigma'$, then $X \to Y \in \Sigma'$.

(g) If $\neg\forall xX \in \Sigma'$, then $\neg X[x/t] \in \Sigma'$ for some closed term t.

(h) For every term t, $(t \approx t) \in \Sigma'$.

(i) If $(s \approx t) \in \Sigma'$, then $(X[x/s] \to X[x/t]) \in \Sigma'$.

Proof Due to the construction of Σ_m's, $\forall xX = \forall xX_n$ for some n. $\Sigma_0 \subseteq \Sigma'$ says that $(X_n[x/t_m] \to \forall xX) \in \Sigma'$. However,

$$\vdash ((X_n[x/t_{n1}] \to \forall xX_n) \to (\neg\forall xX_n \to \neg X_n[x/t_{n1}]))$$

yields $\Sigma' \vdash \neg\forall xX_n \to \neg X_n[x/t_{n1}]$. Then the deduction theorem proves (g). You can prove the other statements yourself. ∎

To give a model of Σ or of Σ', we now start with a Herbrand-like interpretation. Here, instead of the Herbrand universe, take the free universe of all closed terms $\mathcal{T}$ as the domain. Continue, as in the Herbrand interpretation, to define ϕ and the equality relation E. Denote the interpretation $(\mathcal{T}, \phi)$ as I, where $\approx$ is to be interpreted as E. (Review Section 2.8 if you have forgotten the equality relation E.) The necessary definitions follow.

Let $C = \{a : a \text{ is a constant}\}$. Let $\mathcal{F}$ be the set of all function symbols. Then D is defined recursively by:

(a) $C \subseteq \mathcal{T}$.

(b) If $t_1, \ldots, t_n \in \mathcal{T}$ and $f \in \mathcal{F} - C$ has arity n, then $f(t_1, \ldots, t_n) \in \mathcal{T}$.

(c) Each $d \in \mathcal{T}$ is generated by (a) and (b).

The function ϕ for function symbols is defined by:

$$\phi(f)(t_1, \ldots, t_n) = f(t_1, \ldots, t_n)$$

We make use of Σ' for defining ϕ of predicates. Let $\mathcal{P}$ be the set of all atomic formulas without variables, i.e.,

$$\mathcal{P} = \{P(t_1, \ldots, t_n) : P \text{ is } n\text{-ary and } t_1, \ldots, t_n \in \mathcal{T}\} \cup \{s \approx t : s, t \in \mathcal{T}\}$$

Define a relation P' by $P' = \{(t_1, \ldots, t_n) : \Sigma' \vdash P(t_1, \ldots, t_n)\}$. Then take $\pi(P) = P'$. Also, define $\pi(\approx) = E$, the equality relation.

Let I be the interpretation $I = (\mathcal{T}, \phi)$. Since $\mathcal{T}$ is countable, write it as $\mathcal{T} = \{s_0, s_1, \ldots\}$. We also have the set of all variables as $\{x_0, x_1, \ldots$ Define the variable assignment function (valuation) ℓ by taking $\ell(x_n) = s_n$ for $n = 0, 1, \ldots$ Then the state I_ℓ under the interpretation I is ready to connect the syntax with semantics. Use induction on the number of occurrences of $\neg, \rightarrow$ and $\forall$ to prove the following metastatement.

Lemma 4.16 *For each formula X, $I_\ell \models X$ iff $\Sigma' \vdash X$.*

We use it to prove the following Theorem.

Theorem 4.17 (Model Existence Theorem for FC) *Let Σ be any consistent set of formulas. Then Σ has a model.*

Proof Let Σ' and I_ℓ be as constructed above. For each formula $X \in \Sigma \subseteq \Sigma'$, $\Sigma' \vdash X$. By Lemma 4.16, $I_\ell \models X$. Since I_ℓ is a model of every formula $X \in \Sigma$, the set Σ has the model I_ℓ. ∎

As earlier, the statements in the following theorem ensue.

Theorem 4.18 *Let Σ be any set of formulas and X be any formula of FC. Then the following statements hold:*

(a) Strong Adequacy of FC to FL : $\Sigma \vdash X$ iff $\Sigma \models X$.

(b) Adequacy of FC to FL : $\vdash X$ iff $\models X$.

(c) Compactness of FL :

(i) Σ is satisfiable iff every finite subset of Σ is satisfiable.

(ii) $\Sigma \models X$ iff $\Sigma_0 \models X$ for some finite subset Σ_0 of Σ.

***Exercise* 4.12** Fill in all the gaps in the proofs of all the lemmas and theorems above.

***Exercise* 4.13** Use the compactness for PL and the adequacy of PC to derive strong adequacy of PC. Can you prove that FC is strongly adequate by assuming its adequacy and the compactness of FL?

If the restriction on UG had not been imposed, as is done in many texts, we would not have obtained adequacy for all formulas. In that case, adequacy could be guaranteed only for sentences. That would not, of course, result in any loss of generality since validity of any formula can be replaced by validity of its universal closure, which is a sentence. So, be careful about the restrictions on UG while reading other texts.

4.5 Natural Deduction

Though axiomatic systems PC and FC are adequate, they are inconvenient to use due to the stress on formality. Informal methods can be developed from them by allowing all the valid propositions or formulas mentioned earlier as laws (see Chapters 1 and 2) and already proved theorems to be used as axioms. The valid consequences can also be used as inference rules. All these propositions or formulas will be referred to as known theorems and we will document them by writing 'T' on the third column of a proof. We will also refer to the valid consequences by the same letter 'T'. Similarly, a premise will simply be pointed out by writing a 'P'.

Moreover, the deduction theorem can be used inside a proof rather than as a meta-argument. To prove $X \to Y$, we simply introduce X anywhere in the proof and mark its introduction by 'BD' (for beginning an application of the deduction theorem). When we deduce Y later in the proof, the next line will have $X \to Y$, and it will again be marked as 'ED' for telling that here the assumption X has been removed as an assumption, and by the deduction theorem, the formula $X \to Y$ has been obtained. The deduction theorem can be applied many times in a proof and the pairs of 'BD-ED' must be nested like parentheses. In such a case, we may write BD1-ED1, BD2-ED2, and so on.

Similarly, RAA can be used inside a proof by introducing a formula $\neg X$ anywhere in the proof (Document it as 'BC', beginning of a proof by contradiction) and then by deducing formulas Y and $\neg Y$ later. The next line to the last of Y or $\neg Y$ would be X and the documentation would include 'EC' for ending the application of 'BC'. Again, the nesting of many applications of RAA can be done just like parentheses. Such a proof will be referred to as an **informal proof**. Note that a proof can be developed from an informal proof due to adequacy of the systems PC and FC. Look at the following example.

***EXAMPLE* 4.23** Construct an informal proof for the consequence:

$$\{p \to \neg q, r \to s.\neg t \to q, s \to \neg u, t \to \neg v, \neg u \to w\} \vdash p \wedge r \to \neg(w \to v)$$

Solution here is an informal proof. Note the nestings for RAA (lines 2 and 17) and the deduction theorem.

1.	$p \wedge r$	BD
2.	$\neg\neg(w \to v)$	BC
3.	$w \to v$	T
4.	p	1, T
5.	$p \to \neg q$	P
6.	$\neg q$	T (Modus Ponens)
7.	$\neg t \to q$	P

8.	t	$\mathrm{T}\ (\neg q, \neg t \to q \models t)$
9.	$t \to \neg v$	P
10.	$\neg v$	T
11.	r	$1, \mathrm{T}$
12.	$r \to s$	P
13.	s	T
14.	$s \to \neg u$	P
15.	$\neg u$	T
16.	$\neg u \to w$	P
17.	w	T
18.	v	$3, 17, \mathrm{T}$
19.	$\neg(w \to v)$	$10, 17, \mathrm{EC}$
20.	$p \wedge r \to \neg(w \to v)$	$1, 18, \mathrm{ED}$

Constructing such an informal proof requires you to memorize at least all those laws mentioned in Chapters 1 and 2 though we know that only the axioms A1−A5 are sufficient for constructing a proof. We want a system which would make a balance between a formal and an informal proof. For this purpose, we present the so-called **natural deduction system**. In the system **PND** (Propositional Natural Deduction system), we have the following inference rules (read ‘ i ’ for introduction and ‘ e ’ for elimination):

$$(\top i)\ \frac{}{\top} \qquad\qquad (\top e)\ \frac{\boxed{\top \cdots p}}{p} \qquad \frac{\top}{p \vee \neg p}$$

$$(\bot i)\ \frac{p, \neg p}{\bot} \qquad\qquad (\bot e)\ \frac{\bot}{p}$$

$$(\neg i)\ \frac{\boxed{p \cdots \bot}}{\neg p} \qquad \frac{p \wedge \neg q}{\neg(p \to q)} \qquad (\neg e)\ \frac{\neg\bot}{\top} \qquad \frac{\neg(p \to q)}{p \wedge \neg q} \qquad \frac{p \vee \neg q, q}{p}$$

$$(\neg\neg i)\frac{p}{\neg\neg p} \qquad\qquad (\neg\neg e)\frac{\neg\neg p}{p}$$

$$(\wedge i)\ \frac{p, q}{p \wedge q} \qquad\qquad (\wedge e)\ \frac{p \wedge q}{p} \qquad \frac{p \wedge q}{q}$$

$$(\vee i)\ \frac{p}{p \vee q} \qquad \frac{q}{p \vee q} \qquad\qquad (\vee e)\ \frac{\boxed{p \cdots r}\,, \boxed{q \cdots r}\,, p \vee q}{r}$$

$$(\to i)\ \frac{\boxed{p \cdots q}}{p \to q} \qquad (\to e)\ \frac{p,\, p \to q}{q} \qquad \frac{\neg q,\, p \to q}{\neg p}$$

$$(\leftrightarrow i)\ \frac{p \to q,\, q \to p}{p \leftrightarrow q} \qquad (\leftrightarrow e)\ \frac{p \leftrightarrow q}{(p \to q) \wedge (q \to p)}$$

A box written horizontally in the above rules will appear vertical in a proof. A box indicates the conditionality of the premises. For example, in the rule $(\to i)$, the proposition p is introduced from nowhere, an extra assumption, and by following the rules, if you are able to derive q, then you can close the box there and consider that the proposition $p \to q$ has been derived without the extra assumption p. The extra assumption p is, of course, introduced in an actual proof by targeting towards a particular conclusion such as q. But then, all those propositions that are derived with this extra assumption p do depend upon it, and thus must be written inside the box. Once you close the box, and write $p \to q$, all it says is that the propositions outside the box, in particular, $p \to q$ does not depend upon the extra assumption p. Now you see that the rule $(\to i)$ is another form of the deduction theorem better suited to be used inside a proof; it simply says that "from $p \vdash q$, conclude $\vdash p \to q$".

Similarly, the rule $(\vee e)$ means that "if $p \vdash r, q \vdash r$, and $\vdash p \vee q$, then $\vdash r$". The rule $(\top i)$ says that $\top$ can be derived from nothing, i.e., '$\vdash \top$'. This complies with our earlier convention that axioms can be written as fractions with empty numerators.

Similar to PC, you can now define theorems and provable consequences. We assume that the alphabet of PND is the alphabet of PL and the propositions of PND are exactly the propositions of PL. A **derivation** is a sequence of propositions (may be, in boxes), where each is introduced by applying one of the rules. A **proof of the consequence** $\Sigma \vdash p$ is a derivation where each proposition not inside a box is a premise in Σ or is obtained by an application of an inference rule of PND, and the proposition p is the last proposition of the sequence not inside a box. Any box contains a full fledged derivation, where some of the propositions may not be premises. Moreover, a rule can be applied on propositions occurring before the box and the result can be introduced as a proposition inside a box. We have a constraint on the derivations that a box can never cross another box; only nesting of boxes is allowed. A proposition p is a **theorem** of PND, written as $\vdash p$ iff $\emptyset \vdash p$; the derivation is then called a **proof** of the theorem. A set Σ of propositions is called **inconsistent** iff $\Sigma \vdash \bot$ else, it is **consistent**. Note that Σ is consistent means that there cannot be any derivation showing $\Sigma \vdash \bot$. We follow the same three-column style of writing a derivation as in PC, writing 'P' for premises and 'CP' for an extra or conditional premise. Note that we are overusing the symbol $\vdash$. Here, $\vdash$ means 'provability in PND', and not in PC or FC. See the following examples.

***EXAMPLE* 4.24** Construct a PND-proof for $\vdash p \to (q \to p)$.

Solution

$$\begin{array}{l}
\boxed{\begin{array}{l}
\begin{array}{lll} 1. & p & \text{CP} \end{array}\\
\boxed{\begin{array}{lll} 2. & q & \text{CP}\\ 3. & p & 1 \end{array}}\\
\begin{array}{lll} 4. & q \to p & \to i \end{array}
\end{array}}\\
\begin{array}{lll} 5. & p \to (q \to p) & \to i \end{array}
\end{array}$$

***EXAMPLE* 4.25** $\vdash (p \to (q \to r)) \to ((p \to q) \to (p \to r))$.

Solution

$$\begin{array}{l}
\boxed{\begin{array}{l}
\begin{array}{lll} 1. & p \to (q \to r) & \text{CP} \end{array}\\
\boxed{\begin{array}{l}
\begin{array}{lll} 2. & p \to q & \text{CP} \end{array}\\
\boxed{\begin{array}{lll} 3. & p & \text{CP}\\ 4. & q & 2, 3, \to e\\ 5. & q \to r & 1, 3, \to e\\ 6. & r & 4, 5, \to e \end{array}}\\
\begin{array}{lll} 7. & p \to r & 2, 6, \to e \end{array}
\end{array}}\\
\begin{array}{lll} 8. & (p \to q) \to (p \to r) & 1, 7, \to e \end{array}
\end{array}}\\
\begin{array}{lll} 9. & (p \to (q \to r)) \to (p \to q) \to (p \to r)) & 1, 8, \to i \end{array}
\end{array}$$

***EXAMPLE* 4.26** $\vdash (\neg q \to \neg p) \to (p \to q)$.

Solution

$$\begin{array}{l}
\boxed{\begin{array}{l}
\begin{array}{lll} 1. & \neg q \to \neg p & \text{CP} \end{array}\\
\boxed{\begin{array}{lll} 2. & p & \text{CP}\\ 3. & \neg\neg p & \neg\neg i\\ 4. & \neg\neg q & 1, 3, \to e\\ 5. & q & \neg\neg e \end{array}}\\
\begin{array}{lll} 6. & p \to q & 1, 5, \to i \end{array}
\end{array}}\\
\begin{array}{lll} 7. & (\neg q \to \neg p) \to (p \to q) & 1, 6, \to i \end{array}
\end{array}$$

Here are some more examples of derivations in PND. We will follow the earlier convention of omitting unnecessary line numbers. Whenever a rule is applied on one or two propositions immediately preceding the current one, we will not mention them.

***EXAMPLE* 4.27** Construct a natural deduction proof for the consequence in Example 4.23.

Solution

1.	$p \wedge r$	P
	p	$\wedge e$
	$p \to \neg q$	P
	$\neg q$	$\to e$
	$\neg t \to q$	P
	$\neg\neg t$	$\to e$
	t	$\neg\neg e$
	$t \to \neg v$	P
2.	$\neg v$	$\to e$
	r	$1, \wedge e$
	$r \to s$	P
	s	$\to e$
	$s \to \neg u$	P
	$\neg u$	$\to e$
	$\neg u \to w$	P
3.	w	$\to e$
	$\neg v \wedge w$	$2, 3, \wedge i$

In the above derivation, we have not numbered every proposition. While constructing a derivation, you can find out which previous proposition is required to derive the current one. And then, you may only number those relevant ones.

***EXAMPLE* 4.28** Show that the following set Σ of propositions is inconsistent by constructing a proof for the consequence $\Sigma \vdash \bot$:

$$\Sigma = \{p \vee (\neg q \wedge r) \to s, (t \wedge \neg u) \to \neg s, v \vee \neg u, \neg(v \vee \neg r), t, p\}$$

Solution

	p	P
	$p \vee (\neg q \wedge r)$	$\vee i$
	$p \vee (\neg q \wedge r) \to s$	P
1.	s	$\to e$
	t	P

	$\neg u$	CP
	$t \wedge \neg u$	$\wedge i$
	$t \wedge \neg u \to \neg s$	P
2.	$\neg s$	$\to e$
	$\bot$	$1, 2, \bot i$

$\neg\neg u$	$\neg i$
u	$\neg\neg e$
$v \vee \neg u$	P
v	$\vee e$
$v \vee \neg r$	$\vee i$
$\neg(v \vee \neg r)$	P
$\bot$	$\bot i$

You can, of course, construct many other proofs to derive $\bot$ in the last example. Try constructing one different from the above proof.

Examples 4.24-4.26 above show that the axioms of PC are, indeed, theorems of PND. Moreover, the inference rule MP of PC is simply the rule $(\rightarrow e)$ of PND. Hence PND is complete. But not yet, because PND works with all the five connectives and PC has only two. Well, the other connectives and the constants $\top$ and $\bot$ have to be introduced through definitions into PC. Thus, you have to prove these definitions also. That is, show that

$$\top \vdash p \rightarrow p, \quad p \rightarrow p \vdash \top,$$
$$\bot \vdash \neg(p \rightarrow p), \quad \neg(p \rightarrow p) \vdash \bot,$$
$$p \vee q \vdash \neg p \rightarrow q, \quad \neg p \rightarrow q \vdash p \vee q,$$
$$p \wedge q \vdash \neg(p \rightarrow \neg q), \quad \neg(p \rightarrow \neg q) \vdash p \wedge q,$$
$$p \leftrightarrow q \vdash (p \rightarrow q) \wedge (q \rightarrow p), \quad (p \rightarrow q) \wedge (q \rightarrow p) \vdash p \leftrightarrow q.$$

This will complete the proof of completeness of PND. Alternatively, you can also try proving completeness of PND by constructing a maximally consistent set, etc. You can also show that all the rules of PND are sound, i.e., the rules represent valid consequences of PL. Since the proofs in PND are finite in length, using compactness of PL, you can show the following.

Theorem 4.19 (Strong Adequacy of PND) *For any set of propositions Σ and any proposition w, $\Sigma \models w$ iff $\Sigma \vdash w$ in PND.*

***Exercise* 4.14** Formulate and prove monotonicity and RAA for PND.

Basing on PND, we construct a natural deduction system for FL. Let us call the system **FND**, first order natural deduction system. FND has all the inference rules of PND, where p, q, r are taken as formulas. In addition, it has the following inference rules for the predicate $\approx$, and the quantifiers.

For formulas X, Y, variables x, y, terms s, t, and a constant c,

$$(\approx i) \quad \frac{\cdot}{t \approx t}$$

$$(\approx e) \quad \frac{s \approx t, X[x/s]}{X[x/t]} \quad \text{provided that } s, t \text{ are free for } x \text{ in } X.$$

$$(\forall i)\qquad \begin{array}{l}\boxed{\begin{array}{lr} & y \\ \vdots & \\ X[x/y] & \end{array}} \\ \forall x X\end{array}\ , \quad \text{where } y \text{ is a new variable.}$$

$$(\forall e)\qquad \frac{\forall x X}{X[x/t]}$$

$$(\exists i)\qquad \frac{X[x/t]}{\exists x X}$$

$$(\exists e)\qquad \begin{array}{l}\exists x X \\ \boxed{\begin{array}{lr} & c \\ X[x/c] & \\ \vdots & \\ Y & \end{array}} \\ Y\end{array}\ , \quad \text{where } c \text{ is a new constant not occurring in } Y.$$

Except the boxes, other rules are self-evident. As earlier, we follow the restriction on boxes that no box will intersect another, though one box can completely lie inside another. We will explain the use of boxes taking the appropriate rules in turn. In case of $(\forall i)$, the 'for all introduction', the box means that

> If starting with a new variable y, you are able to prove some formula $X[x/y]$, then you have proved $\forall x X$.

Thus the variable y must not occur anywhere outside the box. The box is only a check for an important phrase used in mathematics: "Let x be a fixed but arbitrary number such that ... ". This is a formal expression of the informal statement that if you can prove $X(y)$ for an arbitrary y, then you can prove $\forall x X(x)$. The restriction allows you to draw the conclusion $\forall x X$ only when you have arrived at X in such a way that none of the assumptions you have used contains x as a free variable. And that is exactly the restriction on the rule (UG) in FC, as you have seen earlier. Before explaining $(\exists e)$, let us have an example which uses the rule $\forall i$.

***EXAMPLE* 4.29** Show in FND that $\{\forall x(Pxy \to Qx), \forall z Pzy\} \vdash \forall x Qx$.

Solution

1.	$\forall x(Pxy \to Qx)$	P
2.	$\forall z Pzy$	P

$$\boxed{\begin{array}{ll} & u \\ Puy \to Qu & \forall e,\ [x/u] \\ Puy & 2,\ \forall e \\ Qu & \to e \end{array}}$$
$$\begin{array}{ll} \forall x Qx & \qquad\qquad\qquad \forall i \end{array}$$

The box here controls the scope of the new variable rather than the scope of a conditional premise. Now, what about the box in ($\exists e$)? Whenever you have proved that $\exists xX$, you cannot possibly show your finger at a particular constant a and say that X holds for this a; you cannot bring in a particular term, for that matter. All that you can assert is that for some constant (think in terms of the import semantics) a, the formula $X[x/a]$ holds. Hence this constant must be a new symbol, a new constant.

Alternatively, you can use this $X[x/a]$ in a still different way. Suppose that you have got $X[x/a]$ after elimination of $\exists$ from $\exists xX$. Then you use this formula $X[x/a]$ and deduce another formula Y. If Y does not have an occurrence of this newly introduced constant a, then it does not matter whatever value the constant a might have taken. In any case, Y has been proved. This is the way the rule ($\exists e$) is used in FND. But be careful about the restriction on the rule that the new constant c cannot occur outside the box.

Thus the formula Y is repeated twice; first time, inside the box, just to show that this last formula inside the box, though follows from the earlier formulas, does not have any occurrence of the new constant c, and second time, outside the box, due to the *reason* that this last formula does not have any occurrence of c. The box controls the scope of the fresh constant c which is also documented at the rightmost corner of the box. See the following example and the proof therein. In the proof, $\exists e$ is documented twice: first, in the first line of the box, and second, outside the box. The first one controls the box, where the new constant c has been introduced.

***EXAMPLE* 4.30** $\forall x(Pxy \to Qx), \exists zPzy \vdash \exists xQx$.

Solution

$$\begin{array}{lll} 1. & \forall x(Pxy \to Qx) & \mathrm{P} \\ 2. & \exists zPzy & \mathrm{P} \end{array}$$

$$\boxed{\begin{array}{ll} & c \\ Pcy & 2,\ \exists e \\ Pcy \to Qc & 1,\ \forall e \\ Qc & \to e \\ \exists xQx & \exists i \end{array}}$$

$$\begin{array}{ll} \exists xQx & \qquad\qquad\qquad \exists e \end{array}$$

***EXAMPLE* 4.31** $\vdash \forall xX \to X[x/t]$ for any term t free for x in X.

Solution

$$\begin{array}{ll}
\begin{array}{|lr|}\hline
\forall xX & \text{CP}\\
X[x/t] & \forall e\\ \hline
\end{array} & \\
\forall xX \to X[x/t] & \to i
\end{array}$$

***EXAMPLE* 4.32** $\vdash \forall\, x(X \to Y) \to (X \to \forall\, xY)$ if x is not free in X.

Solution

$$\begin{array}{ll}
\begin{array}{|ll|}\hline
1.\ \forall x(X \to Y) & \text{CP}\\
\begin{array}{|ll|}\hline
X & \text{CP}\\
\begin{array}{|lr|}\hline
 & y\\
X \to Y[x/y] & 1, \forall e\\
Y[x/y] & \to e\\ \hline
\end{array} & \\
\forall xY & \forall i\\ \hline
\end{array} & \\
X \to \forall xY & \to i\\ \hline
\end{array} & \\
\forall x(X \to Y) \to (X \to \forall xY) & \to i
\end{array}$$

***EXAMPLE* 4.33** Show that $\vdash (s \approx t) \to (X[x/s] \to X[x/t])$.

Solution

$$\begin{array}{ll}
\begin{array}{|ll|}\hline
s \approx t & \text{CP}\\
\begin{array}{|lr|}\hline
X[x/s] & \text{CP}\\
X[x/t] & \approx e\\ \hline
\end{array} & \\
X[x/s] \to X[x/t] & \to i\\ \hline
\end{array} & \\
(s \approx t) \to (X[x/s] \to X[x/t]) & \to i
\end{array}$$

Since (A6) of FC is simply the rule ($\approx i$), the above examples and completeness of PND yield the following result.

Theorem 4.20 (Strong Adequacy of FND) *Let Σ be any set of formulas and X be any formula. Then $\Sigma \models w$ iff $\Sigma \vdash w$ in FND.*

Proof Use the definition of $\exists$ in terms of $\forall$ and complete the proof. ▮

The following examples will make you better acquainted with FND.

***EXAMPLE* 4.34** $Pa, \forall x(Px \to Qx), \forall x(Rx \to \neg Qx), Rb \models \neg(a \approx b)$.

Solution We can use RAA in FL and strong adequacy of FND to show that $\{Pa, \forall x(Px \to Qx), \forall x(Rx \to \neg Qx), Rb, \neg\neg(a \approx b)\} \vdash \bot$, which will be sufficient for our purpose. However, the system FND has already a form of RAA built in it. See the following proof:

$$\begin{array}{lll}
1. & Pa & \mathrm{P} \\
2. & Rb & \mathrm{P} \\
3. & \forall x(Px \to Qx) & \mathrm{P} \\
4. & \forall x(Rx \to \neg Qx) & \mathrm{P} \\
5. & Pa \to Qa & 3, \forall e \\
6. & Qa & 1, 5, \to e \\
7. & Rb \to \neg Qb & 4, \forall e \\
8. & \neg Qb & 2, 7, \to e
\end{array}$$

$$\boxed{\begin{array}{llr}
9. & a \approx b & \mathrm{CP} \\
10. & \neg Qa & 8, 9, \approx a \\
11. & \bot & 6, 10, \bot i
\end{array}}$$

$$\begin{array}{lll}
12. & \neg(a \approx b) & \neg i
\end{array}$$

***EXAMPLE* 4.35** All logicians are wise persons. Therefore, all students of logicians are students of wise persons.

Solution Use the vocabulary:

Lx : x is a logician,

Fx : x is a funny person, and

Sxy : x is a student of y.

You are then asked to prove the consequence:

$$\forall x(Lx \to Fx) \vdash \forall x(\exists y(Ly \wedge Sxy) \to \exists y(Fy \wedge Sxy))$$

Proof:

$$\begin{array}{lr}
1.\ \forall x(Lx \to Fx) & \mathrm{P}
\end{array}$$

$$\boxed{\begin{array}{lr}
2.\ \exists y(Ly \wedge Sxy) & \mathrm{CP} \\
\boxed{\begin{array}{lr}
3.\ Lc \wedge Sxc & \exists e \\
4.\ Lc & \wedge e \\
5.\ Lc \to Fc & 1, \forall e \\
6.\ Fc & \to e \\
7.\ Sxc & 3, \wedge e \\
8.\ Fc \wedge Sxc & \wedge i \\
9.\ \exists y(Fy \wedge Sxy) & \exists i
\end{array}} & \\
10.\ \exists y(Fy \wedge Sxy) & \exists e
\end{array}}$$

$$\begin{array}{lr}
11.\ \exists y(Ly \wedge Sxy) \to \exists y(Fy \wedge Sxy) & \to i
\end{array}$$

Note that CP and $\to i$ serve as opening and closing of a box just like the two mentions of $\exists e$ on the rightmost column that documents the proof.

***Exercise* 4.15** Give FND-proofs of all the laws listed in Theorems 1.8 and 2.13. Prove monotonicity, RAA, and the deduction theorem for FND.

4.6 Gentzen Systems

The natural deduction systems PND and FND were first visualized by the logician G. Gentzen, though, in a different notation. Gentzen's notation used the idea of a sequent. For example, see the following rewriting of the rule $(\wedge e)$:

$$\frac{p \wedge q}{p} \quad \text{rewritten as} \quad \frac{\Gamma, p \wedge q \vdash \Delta}{\Gamma, p \vdash \Delta}$$

This rewritten form uses the extra symbols Γ and Δ, which are arbitrary sets of propositions. This means that whatever the sets of propositions Γ, Δ may be, if the consequence $\Gamma, p \vdash \Delta$ holds, then the consequence $\Gamma, p \wedge q \vdash \Delta$ also holds. Look at the reverse way here; we assert the numerator provided that the denominator holds. This is more convenient than the usual 'denominator if numerator' style of writing a rule. Both the consequences in the numerator and the denominator are called **sequents**. Starting from the consequence to be proved, you go on applying the rules, now called **sequent rules**, to get newer sequents. The new sequents may not necessarily hold or may hold. Thus, we must say which sequents are the basic ones that can be recognized easily to hold. Then our plan is to stop at that point and declare that the sequent we have started with also holds. For example, the derivations

$$\frac{p \wedge q \vdash p}{p \vdash p} \qquad \frac{p \wedge q \vdash r}{p \vdash r} \qquad \frac{p \wedge q \vdash p}{q \vdash p}$$

are obtained by applying the above rule. You see that the first derivation ends at the sequent $p \vdash p$, which seems to be correct, while the denominators in other derivations are not provable consequences. These sequents which represent valid consequences, when the relation of syntactic entailment $\vdash$ is interpreted as $\models$, may be taken as axioms of our system. And then the derivation may be considered as a proof of the topmost sequent.

Now that you have got some ideas as to how our proof system with sequents would work, we must try to formalize it. We will have a different proof system than the natural deduction systems PND and FND. Our aim is to make the construction of a proof as mechanical as possible. At the same time, it would not be too indifferent to our intuition. Let me ask you a question. In PND, how do you construct a proof of a consequence? Well, you write down all the premises (given that they are finite in number) on a piece of paper from top to bottom one after another; write the conclusion on the bottom. Then, try to fill in the gap so that it would look like a proof. While filling in the gap with propositions, you must obey the rules of inference. But then some particular choices have to be made. For example, which premise is to be considered at which stage, which rule to apply when, and which proposition is to be taken to start a box, and so on. Usually, experience guides you. In our presentation of the Gentzen system, this choice will be mechanical in the case of PL and mostly so in the case of FL; we will see how. We start with PL.

A **sequent** is of the form $\Sigma \vdash \Gamma$, where Σ, Γ are sets of propositions. We assume, for technical reasons, that $\top$ is also a sequent, though the symbol $\vdash$ does not appear in it. The stand alone symbol $\top$ will be regarded as a sequent; when it occurs in other sequents, it is taken as a proposition, as usual. We omit the curly brackets around the propositions in Σ and Γ while writing the sequents. For example, the following are all sequents:

$$\top, \qquad p, r \vdash q, s, t \qquad p, r \vdash q \qquad p \vdash q \qquad p \vdash \qquad \vdash q \qquad \vdash$$

Note that the empty sequent '$\vdash$' listed last above represents a consequence which never holds, and $\top$ as a sequent represents a valid consequence. We plan to overuse the symbols $\vdash, \top$ instead of inventing more symbols. We will make use of these shortly and see the reasons for these conventions. The sequent $\top$ may be thought of as the *universal sequent* in contrast to the *empty sequent* $\vdash$. A sequent $\Sigma \vdash \Gamma$ is an **axiom** iff Σ and Γ have a common proposition. For example,

$$p \vdash p \quad p, q \vdash p, s, r \quad p, q \vdash s, p, r \quad s, q, r \vdash p, x, y, q$$

are axioms while $\vdash p$ is not an axiom. Neither $p \vdash$ nor $p, q, r \vdash s, t$ is an axiom. We choose to have only inference rules in our system, and no axioms. These axioms are taken care of by the rule $(\top)$ below. The symbol $\top$ terminates a proof. We name our system as **GPC**, Gentzen's Propositional Calculus. The inference rules of GPC are as follows:

Let Σ, Γ, Δ be sets of propositions and p, q be arbitrary propositions. The rules of GPC are (mnemonic on the right):

$(\top)$ $$\frac{\Sigma, p, \Gamma \vdash \Delta, p, \Omega}{\top} \qquad\qquad \frac{p \vdash p}{\top}$$

$(\top \vdash)$ $$\frac{\Sigma, \top, \Gamma \vdash \Delta}{\Sigma, \Gamma \vdash \Delta} \qquad\qquad \frac{\top \vdash}{}$$

$(\vdash \top)$ $$\frac{\Sigma \vdash \Gamma, \top, \Delta}{\top} \qquad\qquad \frac{\vdash \top}{\top}$$

$(\bot \vdash)$ $$\frac{\Sigma, \bot, \Gamma \vdash \Delta}{\top} \qquad\qquad \frac{\bot \vdash}{\top}$$

$(\vdash \bot)$ $$\frac{\Sigma \vdash \Gamma, \bot, \Delta}{\Sigma \vdash \Gamma, \Delta} \qquad\qquad \frac{\vdash \bot}{}$$

$(\neg \vdash)$ $$\frac{\Sigma, \neg p, \Gamma \vdash \Delta}{\Sigma, \Gamma \vdash p, \Delta} \qquad\qquad \frac{\neg p \vdash}{\vdash p}$$

$$(\vdash \neg) \quad \frac{\Sigma \vdash \Gamma, \neg p, \Delta}{p, \Sigma \vdash \Gamma, \Delta} \qquad \frac{\vdash \neg p}{p \vdash}$$

$$(\vee \vdash) \quad \frac{\Sigma, p \vee q, \Gamma \vdash \Delta}{\Sigma, p, \Gamma \vdash \Delta \qquad \Sigma, q, \Gamma \vdash \Delta} \qquad \frac{p \vee q \vdash}{p \vdash \quad q \vdash}$$

$$(\vdash \vee) \quad \frac{\Sigma \vdash \Gamma, p \vee q, \Delta}{\Sigma \vdash \Gamma, p, q, \Delta} \qquad \frac{\vdash p \vee q}{\vdash p, q}$$

$$(\wedge \vdash) \quad \frac{\Sigma, p \wedge q, \Gamma \vdash \Delta}{\Sigma, p, q, \Gamma \vdash \Delta} \qquad \frac{p \wedge q \vdash}{p, q \vdash}$$

$$(\vdash \wedge) \quad \frac{\Sigma \vdash \Gamma, p \wedge q, \Delta}{\Sigma \vdash \Gamma, p, \Delta \qquad \Sigma \vdash \Gamma, q, \Delta} \qquad \frac{\vdash p \wedge q}{\vdash p \qquad \vdash q}$$

$$(\rightarrow \vdash) \quad \frac{\Sigma, p \rightarrow q, \Gamma \vdash \Delta}{\Sigma, \Gamma \vdash p, \Delta \qquad \Sigma, q, \Gamma \vdash \Delta} \qquad \frac{p \rightarrow q \vdash}{\vdash p \qquad q \vdash}$$

$$(\vdash \rightarrow) \quad \frac{\Sigma \vdash \Gamma, p \rightarrow q, \Delta}{\Sigma, p \vdash \Gamma, q, \Delta} \qquad \frac{\vdash p \rightarrow q}{p \vdash q}$$

$$(\leftrightarrow \vdash) \quad \frac{\Sigma, p \leftrightarrow q, \Gamma \vdash \Delta}{\Sigma, p, q, \Gamma \vdash \Delta \qquad \Sigma, \Gamma \vdash p, q, \Delta} \qquad \frac{p \leftrightarrow q \vdash}{p, q \vdash \qquad \vdash p, q}$$

$$(\vdash \leftrightarrow) \quad \frac{\Sigma \vdash \Gamma, p \leftrightarrow q, \Delta}{\Sigma, p \vdash \Gamma, q, \Delta \qquad \Sigma, q \vdash \Gamma, p, \Delta} \qquad \frac{\vdash p \leftrightarrow q}{p \vdash q \qquad q \vdash p}$$

The rules look numerous. But the short form of the rules written on the right-hand side can be used as mnemonics. Once you go through them, you find that there is nothing to remember at all. You can simply get them by understanding the mechanism. The rules for $\neg$ say that you can flip the sides and while doing so, drop or add a $\neg$. The rules for $\vee$ and $\wedge$ say that whenever $\vee$ is on the right of the sequent symbol $\vdash$, just replace it by a comma; a similar thing happens when $\wedge$ is on the left of $\vdash$. This suggests that we interpret a comma on the left of $\vdash$ as $\wedge$, and one on the right as $\vee$. Whenever $\vee$ is on the left or $\wedge$ is on the right of $\vdash$, the sequent gives rise to two sequents. You can then see that the other rules can be obtained by employing the equivalences: $p \rightarrow q \equiv \neg p \vee q, p \leftrightarrow q \equiv (p \rightarrow q) \wedge (q \rightarrow p)$. The rules $(\top \vdash)$ and $(\vdash \bot)$ say that $\top$ on the left and $\bot$ on the right of $\vdash$ can be omitted. Thus the empty sequent $\vdash$ may be seen as the sequent $\top \vdash \bot$. Similarly, the universal sequent $\top$ may be thought of as the sequent $\bot \vdash \top$ or as $\top \vdash \top$ or as $\bot \vdash \bot$ or as $\vdash \top$ or as $\bot \vdash$.

Now, what is a proof and what is a theorem? A **derivation** is a tree whose root is a sequent and it is generated by applications of rules. A

proof of a sequent is a derivation with the sequent at its root and $\top$ at all its leaves. We say that the sequent $\Sigma \vdash \Gamma$ is **provable** if it has a proof, and in such a case, we write $\Sigma \vdash_g \Gamma$. The subscript g reminds us that p is a theorem in the system GPC. A proposition p is a **GPC-theorem** ($\vdash_g p$) if there is a proof of the sequent $\vdash p$. A set of propositions Σ is **inconsistent** in GPC iff the sequent $\Sigma \vdash$ is provable.

***EXAMPLE* 4.36** Show that $\vdash_g p \to (\neg q \to \neg(p \to q))$.

Solution It is straightforward to construct a proof here. Just use the rules and go on taking symbols from one side to the other till you reach at $\top$, or that when you cannot possibly apply any rule. The following is a proof, a proof tree. We are not drawing the tree, but look at the branching towards the last line. See the following example for a proof and a theorem.

$$\begin{array}{ccl}
\multicolumn{2}{c}{\vdash p \to (\neg q \to \neg(p \to q))} & \vdash\to \\
\multicolumn{2}{c}{p \vdash \neg q \to \neg(p \to q)} & \vdash\to \\
\multicolumn{2}{c}{p, \neg q \vdash \neg(p \to q)} & \vdash \neg \\
\multicolumn{2}{c}{p, \neg q, p \to q \vdash} & \neg \vdash \\
\multicolumn{2}{c}{p, p \to q \vdash q} & \to\vdash \\
p \vdash p, q & p, q \vdash q & \top,\ \top \\
\top & \top &
\end{array}$$

Semantically, the rules claim that the numerator is a valid consequence if the denominator is a valid consequence. When a branching rule is applied in a derivation, you would end up at many leaves. Then, the derivation would claim that whenever all the leaves are valid consequences, the sequent at the root is also a valid consequence. You can also define p to be a GPC-theorem if the sequent $\top \vdash p$ is provable, and similarly, a set Σ is inconsistent if the sequent $\Sigma \vdash \bot$ is provable. Why?

***EXAMPLE* 4.37** Check whether the following are GPC-theorems:

(a) $p \to (q \to p))$

(b) $(p \to (q \to r)) \to ((p \to q) \to (p \to r))$

(c) $(\neg p \to \neg q) \to (q \to p)$

(d) $(p \vee q) \leftrightarrow (\neg p \to q)$

Solution (a) With the hope that GPC really captures PL, you may attempt to show that this proposition is a theorem. Then, your aim is towards constructing a proof tree with root as the sequent $\vdash p \to (q \to p))$ and leaves as $\top$. Here is one such proof:

$$\begin{array}{cl}
\vdash p \to (q \to p) & \vdash\to \\
p \vdash q \to p & \vdash\to \\
p, q \vdash p & \top \\
\top &
\end{array}$$

(b) The following proof shows that the proposition is a GPC-theorem:

$$\begin{array}{lr}
\vdash (p \to (q \to r)) \to ((p \to q) \to (p \to r)) & \vdash\to \\
\quad p \to (q \to r) \vdash (p \to q) \to (p \to r) & \vdash\to \\
\quad p \to (q \to r), p \to q \vdash p \to r & \vdash\to \\
\quad p \to (q \to r), p \to q, p \vdash r & \to\vdash \\
p \to (q \to r), p \vdash p, r \qquad p \to (q \to r), q, p \vdash r & \vdash,\ \ \to\vdash \\
\top \qquad q, p \vdash p, r \qquad q \to r, q, p \vdash r & \vdash,\ \ \to\vdash \\
\top \qquad r, q, p \vdash q, r \qquad r, q, p \vdash r & \vdash,\ \ \vdash \\
\top \qquad \top &
\end{array}$$

At the second line, instead of applying ($\vdash\to$), you could have applied ($\to\vdash$) to branch from the proposition $p \to (q \to r)$. Then, the branching that came at the end would have come on the third line; essentially, you would have had to repeat the third line on every path. Do it now, and see what we are talking about. The tree would have become bigger unnecessarily. The rules that have a single denominator are called the *stacking rules*, and the ones with two denominators are the *branching rules*. It is advantageous to apply a stacking rule first and wait for a branching rule to be applied later. Hence we have the following *heuristic*.

Delay in applying a branching rule if stacking rules can be applied.

(c)

$$\begin{array}{lr}
\vdash (\neg p \to \neg q) \to (q \to p) & \vdash\to \\
\quad \neg p \to \neg q \vdash q \to p & \vdash\to \\
\quad \neg p \to \neg q, q \vdash p & \to\vdash \\
q \vdash \neg p, p \qquad \neg q, q \vdash p & \vdash\neg,\ \neg\vdash \\
p, q \vdash p \qquad q \vdash q, p & \top,\ \top \\
\top \qquad \top &
\end{array}$$

(d)

$$\begin{array}{lr}
\vdash p \vee q \leftrightarrow (\neg p \to q) & \vdash\leftrightarrow \\
p \vee q \vdash \neg p \to q \qquad \neg p \to q \vdash p \vee q & \vdash\to,\ \to\vdash \\
p \vee q, \neg p \vdash q \qquad \vdash \neg p, p \vee q \qquad q \vdash p \vee q & \neg\vdash,\ \vdash\neg,\ \vdash\vee \\
p \vee q \vdash p, q \qquad p \vdash p \vee q \qquad q \vdash p, q & \vee\vdash,\ \vdash\vee,\ \top \\
p \vdash p, q \qquad q \vdash p, q \qquad p \vdash p, q \qquad \top & \top \\
\top \qquad \top \qquad \top &
\end{array}$$

***Exercise* 4.16** Show that the following sequents are provable:
(a) $\vdash p \wedge q \leftrightarrow \neg(p \to \neg q)$ (b) $\vdash (p \leftrightarrow q) \leftrightarrow ((p \to q) \wedge (q \to p))$
(c) $\top \leftrightarrow p \vee \neg p$ (d) $\bot \leftrightarrow p \wedge \neg p$ (e) $p, p \to q \vdash q$

After the exercise, you now know that GPC is complete (strongly) with respect to PC, and hence, with respect to PL. But is it a sound system?

What is the meaning of soundness here? It means that, if every leaf of a derivation tree is the universal sequent $\top$, then the consequence obtained from the root sequent by replacing $\vdash$ by $\models$ must be a valid consequence.

One simple question. If you replace $\vdash$ in a sequent by $\models$, do you always get a consequence in PL? Well, if the sequent is $\Sigma \vdash \Gamma$ and Γ is a singleton, then obviously you get a consequence. If Γ is not a singleton, then? Take the simpler case first. If $\Gamma = \emptyset$, then interpret $\Sigma \vdash \emptyset$ as $\Sigma \models \bot$. Why is it so? Because, $\bot$ serves as an identity element on the right side of $\vdash$. Look at $(\vdash \bot)$ rule. Due to $(\vdash \vee)$ rule, if $\Gamma = \{p_1, \ldots, p_m\}$ we may view the sequent $\Sigma \vdash \Gamma$ as the consequence $\Sigma \models p_1 \vee \cdots \vee p_m$. Note that this goes well with the limiting case $\Gamma = \emptyset$ also since $p_1 \vee \cdots \vee p_m \vee \bot \equiv p_1 \vee \cdots \vee p_m$. With this view of a sequent, you can now confirm soundness. This asks you to verify that for each rule of the form

$$\frac{\Sigma \vdash \Gamma}{\Delta \vdash \Omega}$$

the metastatement: "If $\Delta \models \Omega$, then $\Sigma \models \Gamma$" holds. Then apply induction on the (depth of) derivation trees to complete the soundness proof. In the basis case, however, you have to check whether the universal sequent $\top$ corresponds to a valid consequence, which is the case due to our convention. Thus you have proved the following result.

Theorem 4.21 (Strong Adequacy of GPC) *Let Σ be a set of propositions and w be a proposition. Then, $\Sigma \vdash_g w$ iff $\Sigma \models w$.*

Note that you have actually proved a somewhat stronger assertion than the above theorem. Extend the definition of a consequence in PL to $\Sigma \models \Gamma$ for sets of propositions Σ and Γ by: "for any interpretation i, if i is a model of each proposition in Σ, then i is a model of some proposition in Γ". Then you have proved that $\Sigma \vdash \Gamma$ is a provable sequent iff $\Sigma \models \Gamma$. We turn towards a Gentzen system for FL. Naturally, we take all the rules of GPC *in toto*, remembering that the sets $\Sigma, \Gamma, \Delta, \Omega$ are sets of formulas and p, q are arbitrary formulas instead of propositions. We have the following additional rules to tackle the quantifiers and the equality predicate:

$(\forall \vdash)$ $$\frac{\Sigma, \forall x X, \Gamma \vdash \Delta}{\Sigma, X[x/t], \forall x X, \Gamma \vdash \Delta} \qquad \frac{\forall x X \vdash}{X[x/t], \forall x X \vdash}$$

$(\vdash \forall)$ $$\frac{\Sigma \vdash \Gamma, \forall x X, \Delta}{\Sigma \vdash \Gamma, X[x/y], \Delta} \qquad \frac{\vdash \forall x X}{\vdash X[x/y]}$$

provided that y is a **new variable**, i.e., the variable y does not occur free in the numerator.

$(\exists \vdash)$ $$\frac{\Sigma, \exists x X, \Gamma \vdash \Delta}{\Sigma, X[x/y], \Gamma \vdash \Delta} \qquad \frac{\exists x X \vdash}{X[x/y] \vdash}$$

provided that y is a new variable.

$$(\vdash \exists) \quad \frac{\Sigma \vdash \Gamma, \exists x X, \Delta}{\Sigma \vdash \Gamma, X[x/t], \exists x X, \Delta} \qquad \frac{\vdash \exists x X}{\vdash X[x/t], \exists x X}$$

$$(\approx r) \quad \frac{\Sigma \vdash \Gamma}{\Sigma, (t \approx t) \vdash \Gamma} \qquad \frac{\vdash}{(t \approx t) \vdash}$$

$$(\approx c) \quad \frac{\Sigma, (s \approx t), \Gamma \vdash \Delta}{\Sigma, (t \approx s), \Gamma \vdash \Delta} \qquad \frac{(s \approx t) \vdash}{(t \approx s) \vdash}$$

$$(\approx s) \quad \frac{\Sigma, (s \approx t), X[x/s], \Gamma \vdash \Delta}{\Sigma, (s \approx t), X[x/t], \Gamma \vdash \Delta} \qquad \frac{(s \approx t), X[x/s] \vdash}{(s \approx t), X[x/t] \vdash}$$

Note the **eigenvariable condition**, the restrictions in the rules $(\vdash \forall)$, and $(\exists \vdash)$ on the variable y. It is the same condition we used in the FND-rules $(\forall i)$ and $(\exists e)$. The rules $(\approx r)$, $(\approx c)$ and $(\approx s)$ for the equality predicate are easy to see; $(\approx r)$ is the reflexivity, $(\approx c)$ is the commutativity, and $(\approx s)$ is the substitutivity property of $\approx$. In the case of $(\approx r)$, we can only add $t \approx t$ to the left side since $t \approx t$ is equivalent to $\top$ semantically, and $\top$ behaves as the identity on the left side of $\vdash$. (That is, $p, q \vdash r$ and $p, \top, q \vdash r$ represent the same sequent.) We cannot add $t \approx t$ on the right side since $p \vdash q, \top, r$ is not the same sequent as $p \vdash q, r$. Similarly, in $(\approx s)$, we cannot have a rule with $s \approx t$ on the right side. But you can have a commutativity rule for $\approx$, where $s \approx t$ and $t \approx s$ can both be on the right side. We call the new system **GFC** (Gentzen's First Order Calculus). The examples below will help you in using the proof system GFC.

***EXAMPLE* 4.38** The sequent $\vdash \forall x X \to X[x/t]$ is provable.

Solution

$$\begin{array}{cl}
\vdash \forall x X \to X[x/t] & \vdash\to \\
\forall x X \vdash X[x/t] & \forall\vdash \\
X[x/t], \forall x X \vdash X[x/t] & \top \\
\top &
\end{array}$$

***EXAMPLE* 4.39** The sequent $\vdash \forall x(X \to Y) \to (X \to \forall x Y)$ is provable provided that x does not occur free in X.

Solution

$$\begin{array}{ccl}
\multicolumn{2}{c}{\vdash \forall x(X \to Y) \to (X \to \forall x Y)} & \vdash\to \\
\multicolumn{2}{c}{\forall x(X \to Y) \vdash X \to \forall x Y} & \vdash\to \\
\multicolumn{2}{c}{\forall x(X \to Y), X \vdash \forall x Y} & \vdash\forall, [x/x] \\
\multicolumn{2}{c}{\forall x(X \to Y), X \vdash Y} & \forall\vdash, [x/x] \\
\multicolumn{2}{c}{X \to Y, X \vdash Y} & \to\vdash \\
X \vdash X, Y & Y, X \vdash Y & \top,\ \top \\
\top & \top &
\end{array}$$

***EXAMPLE* 4.40** $\vdash_g (t \approx t)$.

Solution

$$\begin{array}{cc} \vdash (t \approx t) & \approx r \\ (t \approx t) \vdash (t \approx t) & \top \\ \top & \end{array}$$

***EXAMPLE* 4.41** $\vdash_g (s \approx t) \to (X[x/s] \to X[x/t])$

Solution

$$\begin{array}{cc} \vdash (s \approx t) \to (X[x/s] \to X[x/t]) & \vdash\to \\ s \approx t \vdash X[x/s] \to X[x/t] & \vdash\to \\ s \approx t, X[x/s] \vdash X[x/t] & \approx s \\ s \approx t, X[x/t] \vdash X[x/t] & \top \\ \top & \end{array}$$

***Exercise* 4.17** Show that the following inference rules are not sound:

$$\text{Rule-1: } \frac{\Sigma \vdash \Gamma, \Delta}{\Sigma \vdash \Gamma, (t \approx t), \Delta} \qquad \text{Rule-2: } \frac{\Sigma \vdash \Gamma, (s \approx t), X[x/s], \Delta}{\Sigma \vdash \Gamma, (s \approx t), X[x/t], \Delta}$$

[*Hint*: A rule with numerator N and denominator D is sound if the metastatement 'if D, then N' holds. Further, a sequent $\Sigma \vdash \Gamma$ holds if every model of all formulas in Σ is a model of some formula in Γ.]

So, you see that all the axioms and inference rules of FC are theorems or provable sequents of GFC except one that we have not yet attempted. It is the rule of universal generalization or

$$\text{(UG)} : \frac{X}{\forall x X}$$

provided that x is not a free variable of any premise used thus far. How to take care of the condition in UG and write a corresponding sequent? Is it $X \vdash \forall x X$, provided, of course, x is not free in a premise used thus far? But the phrase 'a premise used thus far' has seemingly no meaning in the sequent calculus. Moreover, if we interpret this literally, then the condition would impose the restriction that x is not a free variable of X, as $X \vdash \forall x X$ has a premise X. On the other hand, if we disregard X as a premise used here, then $X \vdash \forall x X$ will be unsound since $Px \not\models \forall x Px$. This brings us back to the question: "Does UG allow one to infer or derive $\forall x X$ from the premise X"? No. If you attempt at a derivation, you would arrive at a two-line proof such as

1.	X	P
2.	$\forall x X$	UG

where X has to be used as a premise, and then this is not a proof due to the restriction on UG. But then, when could we use the rule UG meaningfully? Well, we had used it (Look at some examples in FC with this fresh outlook.) on some formulas which were derived in the course of the proof rather than on a premise. The variables on which it had been applied must have come out of universal specification (A4) used somewhere earlier. In that case, you may have another attempt at proving $X \vdash \forall x X$ in GFC. Here is such an attempt:

$$\begin{array}{ll} X \vdash \forall x X & \vdash \forall,\ [x/x] \\ X \vdash X & \top \\ \quad \top & \end{array}$$

But this is not correct since $(\vdash \forall)$ is not applicable as the variable x could occur free in the first line. So, what is the correct formulation of the rule (UG) in terms of sequents? Does UG say that "If $\vdash X$, then $\vdash \forall x X$"? It does not, because the formula that occurs in a proof before applying UG may not really be valid whereas $\vdash X$ means that X is valid semantically. All that is guaranteed is that X is derived from the premises as used in that particular proof (when we assume $\vdash X$).

Suppose you have got a proof in FC of X by using some (or all) of the premises, and then you are applying UG. For convenience, let us take the premises as $X_1, \ldots, X_n$ which have been used in this proof for deriving X. Since UG is applied next for deriving $\forall x X$, we see that x is not free in any of the formulas $X_1, \ldots, X_n$. Now what is the assumption before the application of UG? It is simply the assertion that the consequence $X_1, \ldots, X_n \vdash X$ is valid in FC. Then, after the application of UG, you have proved that $X_1, \ldots, X_n \vdash \forall x X$. Thus, the formulation of UG in GFC must be the metastatement: "If $X_1, \ldots, X_n \vdash_g X$, then derive $X_1, \ldots, X_n \vdash_g \forall x X$ provided x does not occur free in any of $X_1, \ldots, X_n$." This corresponds to the inference rule (in GFC style):

$$\text{(GUG)} \qquad \frac{\Sigma \vdash \forall x X}{\Sigma \vdash X} \quad \text{if } x \text{ is not free in } \Sigma$$

You can easily see that GUG is a derived rule of GFC as a single application of $(\vdash \forall)$ completes the derivation. See to it that the eigenvariable condition is met since x is not free in the numerator. With this, you have proved the completeness of GFC. At this point you must write a formal proof of the following theorem.

Theorem 4.22 (Strong Adequacy of GFC) *Let Σ be a set of formulas and w be a formula. Then, $\Sigma \vdash_g w$ iff $\Sigma \models w$.*

We give some more examples before closing the section.

***EXAMPLE* 4.42** Show that the following sequent is provable:

$\exists x \exists y \exists z(\neg Qx \wedge \neg Qy \wedge \neg Qz \wedge Qf(f(x,y),z) \vdash \exists x \exists y(\neg Qx \wedge \neg Qy \wedge Qf(x,y))$

Solution We use a vertical bar | for indicating the branchings. We apply, in succession, $(\exists \vdash)$ thrice, $(\wedge \vdash)$ thrice, $(\vdash \exists)$ twice, and $(\vdash \wedge)$ twice to obtain the following proof:

$\exists x \exists y \exists z(\neg Qx \wedge \neg Qy \wedge \neg Qz \wedge Qf(f(x,y),z) \vdash \exists x \exists y(\neg Qx \wedge \neg Qy \wedge Qf(x,y))$

$\neg Qu \wedge \neg Qv \wedge \neg Qw \wedge Qf(f(u,v),w) \vdash \exists x \exists y(\neg Qx \wedge \neg Qy \wedge Qf(x,y))$

$\neg Qu, \neg Qv, \neg Qw, Qf(f(u,v),w) \vdash \exists x \exists y(\neg Qx \wedge \neg Qy \wedge Qf(x,y))$

$\neg Qu, \neg Qv, \neg Qw, Qf(f(u,v),w)$

$\quad \vdash \exists x \exists y(\neg Qx \wedge \neg Qy \wedge Qf(x,y)), \neg Qu \wedge \neg Qv \wedge Qf(u,v)$

$\neg Qu, \neg Qv, \neg Qw, Qf(f(u,v),w)$

$\quad \vdash \exists x \exists y(\neg Qx \wedge \neg Qy \wedge Qf(x,y)), \neg Qu$ $\qquad \top$

$\top$

$| \neg Qu, \neg Qv, \neg Qw, Qf(f(u,v),w)$

$\quad \vdash \exists x \exists y(\neg Qx \wedge \neg Qy \wedge Qf(x,y)), \neg Qv$ $\qquad \top$

$\top$

$| \neg Qu, \neg Qv, \neg Qw, Qf(f(u,v),w)$

$\quad \vdash \exists x \exists y(\neg Qx \wedge \neg Qy \wedge Qf(x,y)), Qf(u,v)$ $\qquad \vdash \exists$ twice

The third path does not terminate here. We apply $(\vdash \exists)$ twice as indicated with the substitutions $[x/f(u,v)]$ and $[y/w]$. In the next line we also suppress the sentence $\exists x \exists y(\neg Qx \wedge \neg Qy \wedge Qf(x,y))$, which you should write when rewriting this proof. Notice the branchings of this third path:

$\neg Qu, \neg Qv, \neg Qw, Qf(f(u,v),w)$

$\quad \vdash \neg Qf(u,v) \wedge \neg Qw \wedge Qf(f(u,v),w), Qf(u,v)$ $\qquad \vdash \wedge$ twice

$\neg Qu, \neg Qv, \neg Qw, Qf(f(u,v),w) \vdash \neg Qf(u,v), Qf(u,v)$ $\qquad \vdash \neg$

$\neg Qu, \neg Qv, \neg Qw, Qf(f(u,v),w), Qf(u,v) \vdash Qf(u,v)$ $\qquad \top$

$\top$

$| \neg Qu, \neg Qv, \neg Qw, Qf(f(u,v),w) \vdash \neg Qw, Qf(u,v)$ $\qquad \top$

$\top$

$| \neg Qu, \neg Qv, \neg Qw, Qf(f(u,v),w) \vdash Qf(f(u,v),w), Qf(u,v)$ $\qquad \top$

$\top$

Let us see a nice interpretation of the consequence you have just proved. In the set of real numbers, interpret Qx as x is a rational number, and $f(x,y)$ as x^y. Then the consequence says that "If there are irrational numbers a, b, c such that $(a^b)^c$ is rational, then there are irrational numbers s, t such that s^t is rational". The antecedent clearly holds since, with $a = b = c = \sqrt{2}$, you see that $\sqrt{2}$ is irrational and $(\sqrt{2}^{\sqrt{2}})^{\sqrt{2}} = (\sqrt{2})^2 = 2$ is rational. So, you conclude that there are irrational numbers s, t such that s^t is rational.

Further, if you interpret Qx as x is rational or not algebraic, then your conclusion would be: "there are irrational algebraic numbers s, t such that s^t is rational". Look at the above proof. You have proved that either $\sqrt{2}^{\sqrt{2}}$

is rational, in which case, you have $s = \sqrt{2}, t = \sqrt{2}$ or else, $(\sqrt{2}^{\sqrt{2}})^{\sqrt{2}}$ is rational, in which case, $s = \sqrt{2}^{\sqrt{2}}, t = \sqrt{2}$. It does not determine whether $\sqrt{2}^{\sqrt{2}}$ is rational or not.

***EXAMPLE* 4.43** Show that the following set Σ is unsatisfiable:

$$\Sigma = \{\forall x \forall y(f(x,y) \approx f(y,x)), \forall x \forall y(f(x,y) \approx y), \forall x \exists y \neg(x \approx y)\}$$

Solution Due to strong adequacy, it is enough to show that Σ is inconsistent in GFC (Why?). We construct a proof of the sequent $\Sigma \vdash$. For ease in writing let $P = \forall x \forall y(f(x,y) \approx f(y,x))$ and $Q = \forall x \forall y(f(x,y) \approx y)$. Here is the proof:

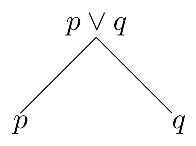
$$\begin{array}{ll}
\forall x \forall y(f(x,y) \approx f(y,x)), \forall x \forall y(f(x,y) \approx y), \forall x \exists y \neg(x \approx y) \vdash & \forall\vdash \\
\forall x \forall y(f(x,y) \approx f(y,x)), \forall x \forall y(f(x,y) \approx y), \exists y \neg(x \approx y) \vdash & \exists\vdash \\
\forall x \forall y(f(x,y) \approx f(y,x)), \forall x \forall y(f(x,y) \approx y), \neg(x \approx z) \vdash & \neg\vdash \\
\forall x \forall y(f(x,y) \approx f(y,x)), \forall x \forall y(f(x,y) \approx y) \vdash x \approx z & \forall\vdash\text{, 4 times} \\
\forall x \forall y(f(x,y) \approx f(y,x)), Q, f(x,z) \approx z, f(z,x) \approx x \vdash x \approx z & \forall\vdash \\
P, Q, f(x,z) \approx z, f(z,x) \approx x, f(x,z) \approx f(z,x) \vdash x \approx z & \approx s\text{ twice} \\
P, Q, f(x,z) \approx z, f(z,x) \approx x, z \approx x \vdash x \approx z & \approx c \\
P, Q, f(x,z) \approx z, f(z,x) \approx x, x \approx z \vdash x \approx z & \top \\
\qquad\qquad\qquad\qquad\top &
\end{array}$$

What about monotonicity, RAA, and the deduction theorem? Observe that provability of $\Sigma \vdash \Delta$ implies the provability of $\Sigma, \Gamma \vdash \Delta$. This proves monotonicity. Since provability of $\Sigma \vdash \Gamma, X \to Y$ implies the provability of $\Sigma, X \vdash \Gamma, Y$ for formulas X, Y, the deduction theorem is proved. Similarly, $\Sigma \cup \{\neg X\}$ is inconsistent iff $\Sigma, \neg X \vdash$ is provable iff $\Sigma \vdash X$. Hence RAA holds in GFC. You can see how simple the proofs of the metatheorems are. No wonder many logicians prefer Gentzen systems for presenting logic. How will Herbrand's theorem look like in GFC?

In the next section we present another proof system, the last one of course, which incorporates the advantages of the tree-like proofs of Gentzen systems staying closer to the semantics.

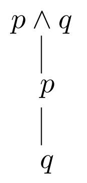
4.7 Analytic Tableau

When can you say that a compound proposition such as $p \vee q$ is true? Obviously, you consider two cases, one, when p is true, and two, when q is true. You can illustrate these cases in a tree:



or, schematically,
$$\begin{array}{ccc} & p \vee q & \\ p & & q \end{array}$$

Similarly, when can you say that $p \wedge q$ holds? This is so when both p and q are true. This is depicted in the tree:

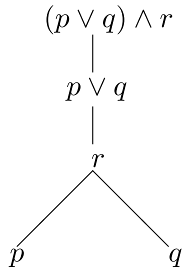


or, schematically,

$$\begin{array}{c} p \wedge q \\ p \\ q \end{array}$$

So, when you branch out, you say that the proposition at the parent node is true provided that at least one of its children is true, and when you stack, the parent proposition is true if both its children are true. This type of **semantic trees** (E. W. Beth's) remind you of the sets of model semantics. If $\mathcal{M}(A)$ denotes the sets of models of A, then $\mathcal{M}(p \vee q) = \mathcal{M}(p) \cup \mathcal{M}(q)$ and $\mathcal{M}(p \wedge q) = \mathcal{M}(p) \cap \mathcal{M}(q)$.

What would happen if you take a more complicated proposition, say, $(p \vee q) \wedge r$? Well, $(p \vee q) \wedge r$ is true when both $p \vee q$ and r are true. Next, go for $p \vee q$ to get the semantic tree as

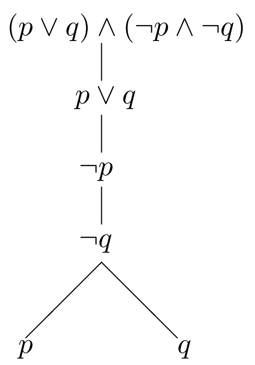


or, schematically,

$$\begin{array}{cc} \multicolumn{2}{c}{(p \vee q) \wedge r} \\ \multicolumn{2}{c}{p \vee q} \\ \multicolumn{2}{c}{r} \\ p & q \end{array}$$

A model of $(p \vee q) \wedge r$ is obtained by looking at both the paths, one from the root $(p \vee q) \wedge r$ to the leaf p, which contains the literals r and p, and the other from the root to the leaf q containing the literals r and q. The two models thus obtained are i, j, where $i(r) = i(p) = 1$, and $j(r) = j(q) = 1$.

In fact, you are getting a dnf representation of the root proposition by these models, i.e., $(p \vee q) \wedge r \equiv (r \wedge p) \vee (r \wedge q)$. If in a path you get an atomic proposition and its negation, then what do you conclude about the root proposition? For example, consider the following semantic tree:



or, schematically,

$$\begin{array}{cc} \multicolumn{2}{c}{(p \vee q) \wedge (\neg p \wedge \neg q)} \\ \multicolumn{2}{c}{p \vee q} \\ \multicolumn{2}{c}{\neg p} \\ \multicolumn{2}{c}{\neg q} \\ p & q \end{array}$$

In the leftmost path, we have the literals $p, \neg q, \neg p$, and the other path contains the literals $q, \neg q, \neg p$. This means that if you have a model of the root proposition, then it must satisfy (at least) one of the sets $\{p, \neg q, \neg p\}$ or $\{q, \neg q, \neg p\}$. But none of these sets is satisfiable since they contain complementary literals. Hence the root proposition is unsatisfiable.

We want to formalize this heuristic of semantic trees by formulating rules to handle the connectives. The resulting system is named as **PT**, the **propositional analytic tableau**. In writing the rules, we adopt the schematic way of depicting the trees as described above. The stacked propositions in a path are simply written one below the other. Similarly, the branchings are separated by some blank spaces. To get the trees, simply join the numerator to the the denominators appropriately. The rules of inference of the system PT are given below, where p, q are any propositions:

$$(\neg\neg) \quad \frac{\neg\neg p}{p} \qquad\qquad (\neg\top) \quad \frac{\neg\top}{\bot}$$

$$(\vee) \quad \frac{p \vee q}{p \qquad q} \qquad\qquad (\neg\vee) \quad \frac{\neg(p \vee q)}{\begin{array}{c}\neg p\\ \neg q\end{array}}$$

$$(\wedge) \quad \frac{p \wedge q}{\begin{array}{c}p\\ q\end{array}} \qquad\qquad (\neg\wedge) \quad \frac{\neg(p \wedge q)}{\neg p \qquad \neg q}$$

$$(\to) \quad \frac{p \to q}{\neg p \qquad q} \qquad\qquad (\neg \to) \quad \frac{\neg(p \to q)}{\begin{array}{c}p\\ \neg q\end{array}}$$

$$(\leftrightarrow) \quad \frac{p \leftrightarrow q}{\begin{array}{cc}p & \neg p\\ q & \neg q\end{array}} \qquad\qquad (\neg \leftrightarrow) \quad \frac{\neg(p \leftrightarrow q)}{\begin{array}{cc}p & \neg p\\ \neg q & q\end{array}}$$

The rules $(\neg\neg), (\neg\vee), (\wedge), (\neg \to), (\neg\top)$ are the **stacking rules** and the others are the **branching rules**. Accordingly, a proposition which is in one of the forms $\neg\neg p, \neg(p \vee q), (p \wedge q), \neg(p \to q), \top, \neg\top, \bot, \neg\bot$, is called a **stacking proposition**, and others, namely, any proposition in one of the forms $(p \wedge q), \neg(p \wedge q), (p \to q), (p \leftrightarrow q), \neg(p \leftrightarrow q), \top, \neg\bot$, is termed as a **branching proposition**.

A **tableau** for a proposition p is a tree whose root is p and it is generated by applying PT-rules. A tableau is generated by first determining in which form the proposition is, and then applying the corresponding rule. The children of the node considered are the denominators of the corresponding rule. A **path in a tableau** is the path from the root to any leaf. If on a path, a rule has been applied on every compound proposition (which are not literals), then the path is called **complete**.

A path in a tableau is called a **closed path** if either contains $\bot$, or it contains p and $\neg p$ for some proposition p. Note that p can also be one of the propositional constants $\top, \bot$. In fact, we will treat $\top$ and $\bot$ as literals. By the rules $(\neg\neg), (\neg\top)$, the propositions $\top$ and $\bot$ are complementary to each other. In a tableau, we put a cross ($\times$) below a closed path. A path which is not closed is called an **open path**. Consequently, a complete path is closed iff it contains a pair of complementary literals. A **completed path** is a path which is either closed or complete (or both). A tableau is called a **closed tableau** if each path in the tableau is a closed path. A tableau is called an **open tableau** if at least one of its paths is an open path. A **completed tableau** is a tableau in which every path is a completed path.

A set of propositions Σ is **inconsistent** if there is a closed tableau with all (or some of) the propositions of Σ in the root. If Σ is a finite set, then it is the same as taking the conjunction of the propositions in Σ as the root. For a set Σ of propositions and a proposition w, $\Sigma \vdash w$ if $\Sigma \cup \{w\}$ is inconsistent. We read $\Sigma \vdash w$ as w follows from Σ, or as Σ **entails** w in PT, or even as "the consequence Σ entails w is PT-provable". Note that RAA is taken as a definition in this system. Thus a proposition w is a **theorem** in PT if $\{\neg w\}$ is inconsistent, i.e., if there is a closed tableau for $\neg w$. In what follows, when a tableau is constructed for a set Σ, we refer to the elements of Σ as **premises** with respect to that tableau.

As in Gentzen systems, it will be a good heuristic to use the stacking proposition first for extending a tableau, keeping branching propositions for later use. This saves space since, when a branching rule is applied on a proposition, we have to add its children to all those leaves of which the proposition is an ancestor. This is because a tableau is generated path-wise. You may generate it either in depth first or in breadth first way, but while generating it breadth first, you must remember to add children of a proposition to every such leaf, the path from which to the root contains the proposition. For example, if there are two propositions, say, $p \vee q$ and $q \vee r$ on the root, then the tableau would look like:

$$\begin{array}{lcccccccc}
1. & & & & p \vee q & & & \\
2. & & & & q \vee r & & & \\
3. & & p & & & & q & \\
4. & q & & r & & q & & r
\end{array}$$

Here, we have used the rule $(\vee)$ on $p \vee q$ to get the tableau up to line 3. The path from the leaf p to the root contains $q \vee r$, the same is true for the leaf q. So, in the next stage, the tableau is extended by adding the children q and r of $q \vee r$ to both these leaves.

***EXAMPLE* 4.44** Show that $\Sigma = \{p \to (\neg q \to r), p \to \neg q, \neg(p \to r)\}$ is inconsistent.

Solution The root of the tableau contains all the three propositions in Σ. As this is our first example, we number the propositions as also document which rule has been applied where. Once you gain experience (not much is required here), you will find out the suitable rule yourself. However, do not forget to cross a closed path. Here is the tableau:

$$\begin{array}{ccccccc}
\multicolumn{6}{c}{1.\ p \to (\neg q \to r)} & \\
\multicolumn{6}{c}{2.\ p \to \neg q} & \\
\multicolumn{6}{c}{3.\ \neg(p \to r)} & \\
\multicolumn{6}{c}{4.\ p} & \text{(On 3.)} \\
\multicolumn{6}{c}{5.\ \neg r} & \\
\multicolumn{3}{c}{6.\ \neg p} & \multicolumn{3}{c}{7.\ \neg q} & \text{(On 2.)} \\
8.\ \neg p & \multicolumn{2}{c}{9.\ \neg q \to r} & 10.\ \neg p & \multicolumn{2}{c}{11.\ \neg q \to r} & \text{(On 1.)} \\
\times & 12.\ \neg\neg q & 13.\ r & \times & 14.\neg\neg q & 15.\ r & \\
 & 16.\ q & \times & & 17.\ q & \times & \\
 & \times & & & \times & &
\end{array}$$

This is a completed tableau. We check for each path. The leftmost path comprises the propositions numbered $1, 2, 3, 4, 5, 6, 8$. There are p and $\neg p$ on this path, and thus it is a closed path. The cross below the leaf ($\neg p$) indicates that the path is closed. The next left path has the propositions numbered $1, 2, 3, 4, 5, 6, 9, 12, 16$. Here also you find both p and $\neg p$, and hence it is closed. The next left path consists of the propositions $1, 2, 3, 4, 5, 6, 9, 13$. This is also a closed path as it contains $p, \neg p$. The other three paths comprise the propositions $1, 2, 3, 4, 5, 7, 10$; $1, 2, 3, 4, 5, 7, 11, 14, 17$; and $1, 2, 3, 4, 5, 7, 15$, containing the complementary literals $p, \neg p$; $q, \neg q$; and $r, \neg r$, respectively. Therefore, the tableau is a closed tableau showing that Σ is inconsistent. However, there are repetitions in the above tableau. It can be shortened as in the following:

$$\begin{array}{cccc}
\multicolumn{4}{c}{p \to (\neg q \to r)} \\
\multicolumn{4}{c}{p \to \neg q} \\
\multicolumn{4}{c}{\neg(p \to r)} \\
\multicolumn{4}{c}{p} \\
\multicolumn{4}{c}{\neg r} \\
\neg p & \multicolumn{3}{c}{\neg q} \\
\times & \neg p & \multicolumn{2}{c}{\neg q \to r} \\
 & \times & \neg\neg q & r \\
 & & \times & \times
\end{array}$$

***EXAMPLE* 4.45** Show the following by analytic tableau:

(a) $\vdash p \to (q \to p)$

(b) $\vdash (p \to (q \to r)) \to ((p \to q) \to (p \to r))$

(c) $(\neg q \to \neg p) \to (p \to q)$

(d) $\{p, p \to q\} \vdash q$

Solution To show that the proposition $p \to (q \to p)$ in (a) is a theorem in PT, we start with the proposition $\neg(p \to (q \to p))$ and then see that the tableau closes. Similarly, (b)$-$(d) are also shown. Before reading further, construct the tableaux for all of them.

(a)

$$\begin{array}{c}
\neg(p \to (q \to p) \\
p \\
\neg(q \to p) \\
q \\
\neg p \\
\times
\end{array}$$

(b)

$$\begin{array}{c}
\neg(p \to (q \to r) \to ((p \to q) \to (p \to r)) \\
p \\
p \to (q \to r) \\
\neg((p \to q) \to (p \to r)) \\
p \to q \\
\neg(p \to r) \\
p \\
\neg r \\
\begin{array}{cc}
\begin{array}{c} \neg p \\ \times \end{array} &
\begin{array}{c} q \\ \begin{array}{cc} \begin{array}{c} \neg p \\ \times \end{array} & \begin{array}{c} q \to r \\ \begin{array}{cc} \begin{array}{c} \neg q \\ \times \end{array} & \begin{array}{c} r \\ \times \end{array} \end{array} \end{array} \end{array} \end{array}
\end{array}
\end{array}$$

(c)

$$\begin{array}{c}
\neg((\neg q \to \neg p) \to (p \to q)) \\
\neg q \to \neg p \\
\neg(p \to q) \\
p \\
\neg q \\
\begin{array}{cc}
\begin{array}{c} \neg\neg q \\ \times \end{array} &
\begin{array}{c} \neg p \\ \times \end{array}
\end{array}
\end{array}$$

(d)

$$\begin{array}{ccc}
 & p & \\
 & p \to q & \\
 & \neg q & \\
\neg p & & q \\
\times & & \times
\end{array}$$

So, PT is complete with respect to PL, is it? Yes, that is what the above example shows. You can prove the soundness of PT by induction on the number of nodes in any tableau. This can easily be extended to the strong soundness and strong completeness of PT. However, we plan to prove these results for tableau without using PC.

Recollect that a proposition is inconsistent if every path in a tableau for the proposition (with the proposition at the root) is closed. But there can be many tableaux with the proposition at the root, and all the tableaux may not be closed. For example, start the tableau by taking the proposition at the root, but do not extend it further; this is clearly a tableau. Even if the proposition at the root is inconsistent (say, $p \wedge \neg p$), the tableau is not closed; inconsistency requires only some tableau to close. Similarly, a proposition would be called consistent if every tableau for the proposition remains open, i.e., in every tableau for the proposition, there is an open path. Alternatively, the business of ‘every tableau’ and ‘some tableau’ is settled well by considering a completed tableau. A proposition is inconsistent if the completed tableau for it is closed. Similarly, a proposition is **consistent** if the completed tableau is open. And a set Σ of propositions is consistent if the completed tableau with all the elements of Σ at the root is open.

***EXAMPLE* 4.46** Consider the following tableau:

$$\begin{array}{cccc}
\multicolumn{4}{c}{(p \to q) \wedge (r \to s)} \\
\multicolumn{4}{c}{s \wedge (q \to t)} \\
\multicolumn{4}{c}{\neg t} \\
\multicolumn{4}{c}{\neg(\neg p \vee \neg r)} \\
\multicolumn{4}{c}{\neg\neg p} \\
\multicolumn{4}{c}{\neg\neg r} \\
\multicolumn{4}{c}{s} \\
\multicolumn{4}{c}{q \to t} \\
\multicolumn{4}{c}{p \to q} \\
\multicolumn{4}{c}{r \to s} \\
\neg r & \multicolumn{3}{c}{s} \\
\times & \neg p & \multicolumn{2}{c}{q} \\
 & \times & \neg q & t \\
 & & \times & \times
\end{array}$$

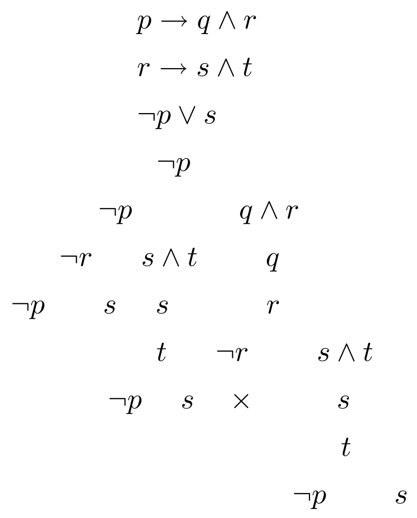
Since this completed tableau is closed, the set of propositions

$$\Sigma = \{(p \to q) \wedge (r \to s), s \wedge (q \to t), \neg t, \neg(\neg p \vee \neg r)\}$$

is inconsistent. (Why?) Of course, a completed tableau is not a must for showing inconsistency! For showing consistency, however, we must construct a completed tableau.

***EXAMPLE* 4.47** Is $\Sigma = \{p \to q \wedge r, \neg p \vee s, r \to s \wedge t, \neg p\}$ consistent?

Solution Here is the completed tableau for Σ:



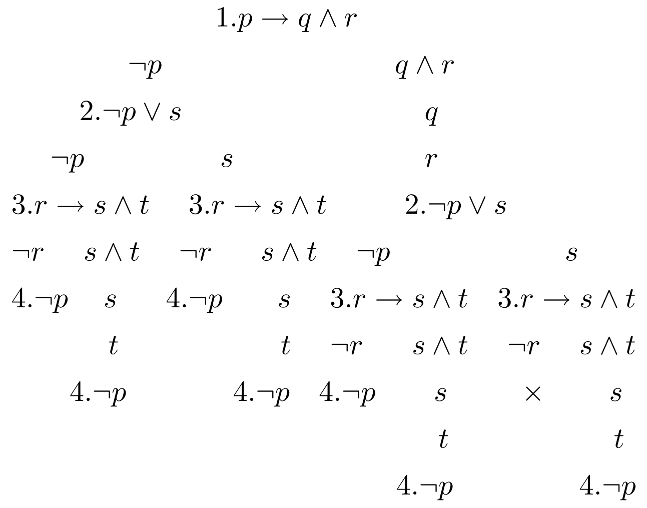
The tableau is completed but is not closed. Hence Σ is consistent.

***Exercise* 4.18** Let w be a proposition such that every path in the completed tableau for it is open. Show that w need not be valid.

The generation of a completed tableau can be made systematic by considering the set Σ of propositions as an ordered set. Since the set of all propositions is countable, Σ can be at most countable and then, we can have an enumeration of it, say, its elements are: $X_1, X_2, X_3, \ldots$. A **systematic** tableau for Σ is generated by starting with X_1 as the root. Apply the suitable tableau rule to generate its children (or a child), apply tableau rules on the children, on the children of children, and so on till you get a completed tableau. This is a completed tableau for the single proposition X_1. Suppose the leaves on the open paths are: $X_{11}, X_{12}, \ldots, X_{1m1}$. Add X_2 as a child to each of these leaves $X_{11}, X_{12}, \ldots, X_{1m1}$. Apply tableau rules on X_2 on each of these $m1$ copies of X_2 and generate the completed tableau. Call the leaves of the open paths of the completed tableau generated so far as $X_{21}, X_{22}, \ldots, X_{2m2}$. Add X_3 to each of these leaves and extend the tableau by applying tableau rules on each of these copies of X_3 to get a still bigger completed tableau. Continue this process of extending the tableau, adding in each stage a new proposition from Σ and then completing the application of rules on them. The tableau so obtained is the systematic tableau for Σ. If Σ is a finite set, then the systematic tableau for Σ, now considered as an ordered set, is a finite binary tree, a completed tableau which is also finite. If Σ is infinite, then either the tableau closes after a finite number of propositions from Σ are used, or it continues to grow to an infinite completed tableau.

***EXAMPLE* 4.48** Construct a systematic tableau for the set Σ (ordered as written) in Example 4.47.

Solution Here is the systematic tableau:


$1.p \to q \wedge r$
$\neg p$ $q \wedge r$
$2.\neg p \vee s$ q
$\neg p$ s r
$3.r \to s \wedge t$ $3.r \to s \wedge t$ $2.\neg p \vee s$
$\neg r$ $s \wedge t$ $\neg r$ $s \wedge t$ $\neg p$ s
$4.\neg p$ s $4.\neg p$ s $3.r \to s \wedge t$ $3.r \to s \wedge t$
t t $\neg r$ $s \wedge t$ $\neg r$ $s \wedge t$
$4.\neg p$ $4.\neg p$ $4.\neg p$ s $\times$ s
t t
$4.\neg p$ $4.\neg p$

In the above tableau, we have taken the first proposition $p \to q \wedge r$ of Σ as the root. We have also numbered it as 1. After applying the tableau rules, we end up at a tree with leaves $\neg p$ and r. Then the second element $\neg p \vee s$ (numbered 2) of Σ is added to the leaves. Again, the tableau rules are applied to extend the tree. The leaves of the extended tableau are $\neg p, s, \neg p, s$, four in number. To each of them is added the third element $r \to s \wedge t$. Once again the tableau is extended and the eight leaves contain the propositions $\neg r, t, \neg r, t, \neg r, t, \neg r, t$, respectively from left to right. One of them is on a closed path and it is marked $\times$. To the other leaves, we add the fourth element $\neg p$ of Σ. Since it is a literal, no tableau rule can be applied, and the tableau is completed systematically.

We know that Σ in the above example is consistent. Is it also satisfiable? It is, since you can construct a model of Σ from an open path in its systematic tableau. Take, for example, the leftmost path. The literals in this path are: $\neg p, \neg r$. This set itself is a model of Σ. To be explicit, define an interpretation $i : \{p, q, r, s, t\} \to \{0, 1\}$ by taking $i(p) = 0 = i(r)$ and $i(q) = i(s) = i(t) = 0$. You can verify that $i \models \Sigma$. It does not matter what you assign to the variables q, s, t here. Take different interpretations by assigning different values to these variables but keep both of p, r assigned to 0 and verify that they are indeed models of Σ. Take another open path, say, the second from the left. This contains the literals $\neg p, s, t$. Then the corresponding model of Σ is any interpretation that assigns p to 0, and s, t to 1. You can also verify this. Each open path in the systematic tableau gives rise to a model of Σ. Why?

Look at the leftmost path in the above tableau once again. It contains the propositions $p \rightarrow q \wedge r, \neg p, \neg p \vee s, \neg p, r \rightarrow s \wedge t, \neg r, \neg p$. The four elements of Σ are appearing in that order, but in between them come some more propositions. These extra propositions have come by application of tableau rules. The first proposition in this path is $p \rightarrow q \wedge r$, from which came the second proposition $\neg p$. This means that $p \rightarrow q \wedge r$ is true (assigned to 1 under an interpretation) whenever $\neg p$ is true. From the second element $\neg p \vee s$ of Σ comes the next proposition $\neg p$ in the path. Here again, whenever $\neg p$ is true, $\neg p \vee s$ is also true. From the third element $r \rightarrow s \wedge t$ of Σ comes the next literal $\neg r$. Here also, if an interpretation assigns $\neg r$ to 1, then it must assign 1 to $r \rightarrow s \wedge t$. Finally, $\neg p$, the fourth element of Σ, is itself a literal, and it would be assigned to 1.

What we observe is that if a literal comes from a proposition by applications of tableau rules in a path, then the proposition is true provided that all the literals are true. This fact is easy to prove by induction since each tableau rule has this property. This means that if you have a tableau rule with parent proposition X and children Y, Z, (or only Y) and i is any interpretation, then the following statements hold:

(S) If it is a stacking rule, then $i(X) = 1$ iff $i(Y) = i(Z) = 1$.

(B) If it is a branching rule, then $i(X) = 1$ iff $i(Y) = 1$ or $i(Z) = 1$.

***Exercise* 4.19** Show: each tableau rule has the properties (S) and (B).

Suppose that Σ is a finite set of propositions. Since any path in a systematic tableau contains all the elements of Σ, any model of the literals in any path is also a model of the whole path due to properties (S) and (B). Moreover, if a path in the systematic tableau is open, then it does not contain a pair of complementary literals, and then, a model can be constructed by assigning each of these literals to 1. Can you argue the same way even if Σ is infinite? Let $\mathcal{S}$ be the systematic tableau for Σ, which we assume to be an ordered infinite set. We must answer whether any open path in $\mathcal{S}$ contains all the elements of Σ.

To start with, we note that an open path $\mathcal{P}$ in $\mathcal{S}$ has to be infinite. For, if $\mathcal{P}$ is finite, then in it must have occurred a finite number of premises, and this would force us to introduce one more premise since $\mathcal{S}$ is generated systematically. Now, if an element, say, $X_m \in \Sigma$, does not occur in $\mathcal{P}$, but X_{m-1} occurs, then X_m has not been added to this path while generating $\mathcal{S}$ systematically. Why? The only possibility is that $\mathcal{P}$ must have closed after X_{m-1} has been introduced. But this is not possible since $\mathcal{P}$ is open. Therefore, any open path in a systematic tableau for a set Σ must contain all elements of Σ, even when Σ is infinite. Since such a path is open, the set of all literals occurring in this path is satisfiable. Then, from the properties (S) and (B), it follows that the set of all propositions occurring in this open path is satisfiable. But all the propositions in Σ occur in this path. Therefore, Σ is satisfiable. We have thus proved the following interesting result.

Theorem 4.23 (Strong Completeness of PT) *Every (PT-) consistent set of propositions is satisfiable.*

Conversely, if a set Σ of propositions is inconsistent, then the systematic tableau for Σ is a closed tableau. However, all the propositions in Σ might not have occurred in the tableau. Denote by Γ the set of all propositions that occur in the systematic tableau for Σ. Due to the properties (S) and (B), Γ is unsatisfiable. Since $\Gamma \subseteq \Sigma$, the set Σ is also unsatisfiable. We thus obtain the following result.

Theorem 4.24 (Strong Soundness of PT) *Every satisfiable set of propositions is (PT-) consistent.*

What can be Γ, the set of propositions occurring in the closed systematic tableau for Σ? Is it finite or infinite? Since the tableau is generated systematically, if at no stage of introducing a new proposition from Σ we get a pair of complementary literals in some path, then that path can never close. Hence, every closed path in a systematic tableau is finite. Does that mean that Γ has only a finite number of propositions? Yes, since a tableau is a binary tree.

In general, if a tree is **finitely generated**, i.e., if each node has at the most m number of children, for some $m \in \mathbb{N}$, and each path has a finite number of nodes, then the tree can have only a finite number of nodes. This result is expressed more elegantly by the following lemma.

Lemma 4.25 (König's Lemma) *Each finitely generated infinite tree has an infinite path.*

Proof Start from the root of the tree τ_0. There are finitely many, say, k (where $k \leq m$ for some $m \in \mathbb{N}$, m fixed for the tree) children. Take out the root, and you will end up in k different trees, the subtrees of τ_0 with roots as the children of τ_0. If all these subtrees are finite, then clearly, τ_0 is also finite, which is not the case. Thus, there is at least one subtree of τ_0 which is infinite.

Choose one of these subtrees and call it τ_1. Repeat the same procedure with τ_1 as you had done with τ_0. You will be able to choose one subtree τ_2 of τ_1, and hence of τ_0 which is infinite. By induction it follows that there is a (infinite) sequence of trees $\tau_0, \tau_1, \tau_2, \ldots$ such that each tree in this sequence is an infinite tree and each is a subtree of all of the preceding trees. Now the sequence of the roots of these trees in the same order is an infinite path in τ_0. ∎

***Exercise* 4.20** Show that if in a tree each node has at most m children, and each path consists of at most n nodes, then the tree consists of at most $\sum_{i=0}^{n} m^i$ many nodes. This directly proves the result we want.

Coming back to what we were doing, we see that due to König's lemma, the set of propositions in the systematic closed tableau Γ for Σ is a finite set. Then, Γ must have used only a finite number of propositions from Σ. Thus, we have proved the following statement.

Theorem 4.26 (Compactness of PT) *A set of propositions is inconsistent iff it has a finite subset which is inconsistent.*

We reformulate the above metatheorems in terms of $\vdash$ and $\models$ instead of inconsistency and unsatisfiability. Let Σ be a set of propositions and w be any propositions. Now, by definition of theoremhood in PT, we have $\Sigma \vdash w$ iff $\Sigma \cup \{\neg w\}$ is inconsistent. Due to the strong soundness and strong completeness of PT, this is equivalent to "$\Sigma \cup \{\neg w\}$ is unsatisfiable". By RAA (in PL), $\Sigma \models w$. We have thus proved the following statement.

Theorem 4.27 (Strong Adequacy of PT) *Let Σ be a set of propositions and w be any proposition. $\Sigma \vdash w$ in PT iff $\Sigma \models w$.*

Similarly, treading between inconsistency and unsatisfiability, due to the strong soundness and strong completeness of PT, we see that each set is unsatisfiable iff it has a finite unsatisfiable subset. Using RAA, we get the usual formulation of compactness for PL.

Theorem 4.28 (Compactness for PT and PL) *Let Σ be a set of propositions and w be any proposition. Then $\Sigma \vdash w$ in PT iff $\Sigma_0 \vdash w$ for some finite subset Σ_0 of Σ. Moreover, $\Sigma \models w$ iff $\Sigma_0 \models w$ for some finite subset Σ_0 of Σ.*

***Exercise* 4.21** Prove the deduction theorem and monotonicity for PT.

With this experience with PT, we go to construct a tableau system for FL. We name the tableau system for FL as **FT**, the **first order analytic tableau**. In FT, all the earlier rules for PT are taken *in toto*. Here, of course, the symbols p, q in the rules are taken as formulas and not just propositions. We now give some more rules to handle the quantifiers and the equality predicate. The following are the additional rules of inference.

For any formula X and for any term t free for the variable x in X,

$$(\forall)\quad \frac{\forall x X}{\begin{array}{c}X[x/t]\\ \forall x X\end{array}} \qquad\qquad (\neg\forall)\quad \frac{\neg\forall x X}{\neg X[x/t]}\ \text{where } t \text{ is a new term}$$

$$(\exists)\quad \frac{\exists x X}{X[x/t]}\ \text{where } t \text{ is a new term} \qquad\qquad (\neg\exists)\quad \frac{\neg\exists x X}{\begin{array}{c}\neg X[x/t]\\ \neg\exists x X\end{array}}$$

$$(\approx r)\quad \frac{\cdot}{t \approx t} \qquad\qquad (\approx s)\quad \frac{\begin{array}{c}X[x/s]\\ s \approx t\end{array}}{X[x/t]}$$

In the rules ($\forall$) and ($\neg\exists$), the formula itself is also repeated. This means that we can use these formulas again if need arises, possibly, instantiating them with different terms than t. In practice, we do not write the formulas again, but do remember that we can use these formulas again. Formulas of these two types are called **universal formulas** which can be instantiated again and again. The corresponding rules are the **universal rules**. In contrast, the other two types of formulas in the rules of ($\neg\forall$) and ($\exists$) are called **existential formulas**, and the rules are called **existential rules**. In the existential rules, the restriction of 't is a new term' is in parallel with the restriction on (UG) in FC, the 'new variable' in FND, and the eigen variable condition in GFC. We use the phrase 't is a **new term**' to mean that one of the following provisos is met:

Proviso 1 : No term that has occurred in any formula in the path from the root to the current existential formula (node on which the rule is being applied) occurs in t.

Proviso 2 : All the the following four conditions are met:

(a) t neither occurs in the current formula nor in any premise.

(b) t has not occurred in any term s that has been introduced by any existential rule in the path earlier to the current formula.

(c) no other term s which has been introduced earlier to the path by an existential rule occurs in the current formula.

(d) no term s that has been introduced by an existential rule in the path earlier to the current formula occurs in t.

For the time being, you may forget the four conditions and remember that the 'new term' means only the first alternative that it is new to the path. We will clarify the use of the second alternative later. These instantiations must be done with new terms and they can be effected only once in a path. The notion of theorems, consequences, consistency, etc. is the same as in PT. It is clear that the rules are sound. We show their completeness by proving all the quantifier rules and equality rules of FC in the following examples.

***EXAMPLE* 4.49** $\vdash \forall x X \to X[x/t]$, for any term t free for x in X.

Solution We start generating a tableau by negating the formula and see that the tableau closes.

$$\begin{array}{ll}
\neg(\forall x X \to X[x/t]) & \\
1.\ \forall x X & \\
\neg X[x/t] & \\
X[x/t] & 1.\ \forall \\
\times &
\end{array}$$

***EXAMPLE* 4.50** $\vdash \forall x(X \to Y) \to (X \to \forall xY)$, where x not free in X.

Solution

$$\begin{array}{ccl}
\multicolumn{2}{c}{\neg(\forall x(X \to Y) \to (X \to \forall xY)} & \\
\multicolumn{2}{c}{1.\ \forall x(X \to Y)} & \\
\multicolumn{2}{c}{\neg(X \to \forall xY)} & \\
\multicolumn{2}{c}{X} & \\
\multicolumn{2}{c}{\neg Y[x/t]} & t \text{ new} \\
\multicolumn{2}{c}{X \to Y[x/t]} & 1.\ \forall,\ x \text{ not free in } X \\
\neg X & Y[x/t] & \\
\times & \times &
\end{array}$$

***EXAMPLE* 4.51** Show that $\vdash (t \approx t)$

Solution

$$\begin{array}{cl}
\neg(t \approx t) & \\
t \approx t & \approx r \\
\times &
\end{array}$$

***EXAMPLE* 4.52** Show that $\vdash (s \approx t) \to (X[x/s] \to X[x/t])$, where s, t are free for x in X.

Solution

$$\begin{array}{cl}
(s \approx t) \to (X[x/s] \to X[x/t]) & \\
1.\quad s \approx t & \\
\neg(X[x/s] \to X[x/t]) & \\
2.\quad X[x/s] & \\
\neg X[x/t] & \\
X[x/t] & 1, 2, \approx s \\
\times &
\end{array}$$

***EXAMPLE* 4.53** Let x be a variable not free in any of the formulas $X_1, \ldots, X_n, X$. Show that if $\{X_1, \ldots, X_n\} \vdash X$ then $\{X_1, \ldots, X_n\} \vdash \forall xX$.

Solution We have to show that if a tableau for $\{X_1, \ldots, X_n, \neg X\}$ closes, then there is a tableau for $\Sigma = \{X_1, \ldots, X_n, \neg\forall xX\}$ which also closes. Let τ be a closed tableau for $\{X_1, \ldots, X_n, \neg X\}$. We construct the following tableau for Σ:

$$\begin{array}{cl}
X_1 & \\
\vdots & \\
X_n & \\
\neg\forall xX & \\
1.\ \neg X & \neg\forall \\
\tau & \\
\times &
\end{array}$$

Since τ uses the propositions $X_1, \ldots, X_n, \neg X$ and it closes, the above is a closed tableau. The entry of the proposition numbered 1 is justified since x is not free in any of $X_1, \ldots, X_n, \neg\forall x X$, and since $\neg X[x/x] = \neg X$.

The above example shows that the rule (UG) of FC is a derived rule of FT. We have already seen that all the axioms of FC are also theorems in FT. Since FT is sound, this completes the proof of adequacy of analytic tableau. We will give its direct proof shortly. Before that, let us see some more examples.

***EXAMPLE* 4.54** Show by the tableau method that the following argument is correct: "If all logicians are mathematicians and if there is a logician who is a computer scientist, then there must be a mathematician who is also a computer scientist".

Solution A translation into FL asks for the validity of the consequence:

$$\{\forall x(Lx \to Mx), \exists x(Lx \wedge Cx)\} \models \exists x(Mx \wedge Cx)$$

In the tableau method we construct a closed tableau for the set

$$\{\forall x(Lx \to Mx), \exists x(Lx \wedge Cx), \neg\exists x(Mx \wedge Cx)\}$$

Here we are using the completeness of FT. Recollect that in PT we have used the heuristic of applying all stacking rules before applying any branching rule whenever possible. Along with that, we use the following *heuristic* :

If possible, apply existential rules before applying universal rules.

That is, whenever possible, use the ($\exists$) and ($\neg\forall$) rules before applying the ($\forall$) and ($\neg\exists$) rules. This is because an existential rule introduces a new term which can be chosen as a term in a universal rule for instantiation. However, once a term is chosen for instantiation in a universal rule, it cannot be used in an existential rule losing flexibility. See the following tableau:

$$\begin{array}{ccccl}
 & & \forall x(Lx \to Mx) & & \\
 & & \exists x(Lx \wedge Cx) & & \\
 & & \neg\exists x(Mx \wedge Cx) & & \\
 & & La \wedge Ca & & a \text{ is a new constant} \\
 & & La & & \\
 & & Ca & & \\
 & & La \to Ma & & \\
 & & \neg(Ma \wedge Ca) & & \\
\neg La & & & Ma & \\
\times & & \neg Ma & & \neg Ca \\
 & & \times & & \times
\end{array}$$

***EXAMPLE* 4.55** Show by the tableau method that (see Example 4.42):
$\exists x \exists y \exists z(\neg Qx \wedge \neg Qy \wedge \neg Qz \wedge Qf(f(x,y),z) \models x \exists y(\neg Qx \wedge \neg Qy \wedge Qf(x,y))$

Solution Here is a tableau proof:

$$\exists x \exists y \exists z(\neg Qx \wedge \neg Qy \wedge \neg Qz \wedge Qf(f(x,y),z)$$
$$1.\ \neg \exists x \exists y(\neg Qx \wedge \neg Qy \wedge Qf(x,y))$$
$$\neg Qa \wedge \neg Qb \wedge \neg Qc \wedge Qf(f(a,b),c)$$
$$\neg Qa$$
$$\neg Qb$$
$$\neg Qc$$
$$Qf(f(a,b),c)$$
$$2.\ \neg(\neg Qa \wedge \neg Qb \wedge Qf(a,b))$$
$$\neg\neg Qa \qquad\qquad \neg(\neg Qb \wedge \neg Qf(a,b))$$
$$\times \qquad \neg\neg Qb \qquad\qquad \neg Qf(a,b)$$
$$\times \qquad 3.\ \neg(\neg Qf(a,b) \wedge \neg Qc \wedge Qf(f(a,b),c))$$
$$\neg\neg Qf(a,b) \qquad\qquad \neg(\neg Qc \wedge Qf(f(a,b),c))$$
$$\times \qquad\qquad \neg\neg Qc \qquad\qquad \neg Qf(f(a,b),c))$$
$$\times \qquad\qquad\qquad \times$$

Observe how the formula numbered 1 has been used twice in the tableau above giving rise to different children. In PT, this is not done (think of a systematic tableau). Even if it is done, it would simply repeat the same children. In the presence of quantifiers, reuse of a universal formula with different substitutions give different children. From $\forall xX$, we can have $X[x/a]$ as a child. In fact, we can have infinitely many such formulas as children of $\forall xX$, depending upon what substitution we choose. This is the reason that in the universal rules the parent formulas are repeated, whereas in existential rules, the parent formulas are not repeated, meaning that they can be used only once, and that too with a new term.

To clarify the second proviso of satisfying four conditions that go with a 'new term' in an existential rule, consider the following example and its two solutions.

***EXAMPLE* 4.56** Show that $\vdash \exists x(\exists yPy \to Px)$ in PT.

Solution 1

$$1.\ \neg \exists x(\exists yPy \to Px)$$
$$2.\ \neg \exists yPy \to Pa$$
$$3.\ \exists yPy$$
$$4.\ \neg Pa$$
$$5.\ Pb$$

6. $\neg\exists y Py \to Pb$

7. $\exists y Py$

8. $\neg Pb$

9. $\times$

Let us see how this tableau has been generated. Formula 2 comes from formula 1 by applying $(\neg\exists)$ and choosing the term t as the constant a. Formulas 3 and 4 come from formula 2 by $(\neg \to)$. Formula 5 comes from 3 by applying $\exists$. Formula 6 comes from 1 again by applying $(\neg\exists)$. And formulas 7 and 8 come from 6 by $(\neg \to)$. Let us see it carefully, there has been an irritating repetition in the tableau. We had to instantiate formula 1 with the constant a, and then once again, with b. Why could we not use a again, say, to instantiate formula 3? Had it been done, we would have obtained straight Pa, and that would have closed the tableau in six lines. The reason is that while instantiating Formula 3, we had to choose a new term, and it must be different from the already appearing a. So, we had followed the first proviso that has been stated with an existential rule. If we follow the second proviso instead, then perhaps we would be able to use the same constant a again. Is it or is it not? Look at the following alternative solution.

Solution 2

1. $\neg\exists x(\exists y Py \to Px)$

2. $\neg\exists y Py \to Pa$

3. $\exists y Py$

4. $\neg Pa$

5. Pa

6. $\times$

Is the introduction of Pa justified, say applying $\exists$ on the formula 3? We must see that all the four conditions in the second proviso of an existential rule are satisfied by the constant a. Well, first, a does not occur in the current formula, i.e., formula 3, $\exists y Py$; also, a does not occur in the premise $\exists x(\exists y Py \to Px)$. Second, a has not been introduced by an existential rule in the path earlier to the current formula, i.e., in the formulas 1 and 2. Third, (and fourth,) no other term s that has been introduced to the path (1 and 2) by an existential rule occurs in the current formula since there has been no application of an existential rule yet. Hence the formula 5 is allowed to be introduced after the formula 3 by an application of $(\exists)$. Therefore, the tableau in Solution 2 is, indeed, a proof.

Thus, it is advisable to get familiar with the second proviso. Moreover, the second proviso, though complicated, is more general than the first. Informally, the four conditions in the proviso ask us to check that the term t used for instantiating an existential formula should not have occurred in any premise, and it should not have been introduced earlier (in that path) by another existential instantiation, nor even in parts. The phrase 'nor even in parts' means that if $f(a, b)$ has been used in an existential instantiation earlier, then a cannot be used now and also $f(f(a, b), a)$ cannot be used.

Similar to the systematic tableau for propositions, how do we generate a systematic tableau for formulas? In PT, we complete a tableau for a single proposition, then add to each leaf the second proposition, complete all the tableaux thereby, add the third proposition, and so on. In FT, we may not, possibly, be successful in doing that. The reason is that if a universal formula appears as a premise, then an application of the tableau rule will introduce an instance of the formula, and along with that the formula itself is repeated. Hence the tableau for such a proposition cannot be completed in any path. The key is that we just bear with it, choose another formula and keep the universal formula for later use. In FT, a **systematic generation** of a tableau for a given set Σ of formulas is described inductively as follows.

Assume that the set $\Sigma = \{X_1, X_2, \ldots, X_m \ldots\}$ is already ordered. Since the set $\mathcal{T} = \{t_1, t_2, \ldots, t_n \ldots\}$ of all terms is enumerable, we take it as an ordered set. The construction of the systematic tableau for Σ starts with taking X_1 as the root. Here we say that the premise X_1 has been introduced to the tableau. This completes the first step of our construction. Suppose that the n-th step of our construction has been completed, and we have obtained a tableau, say, τ. If τ is a closed tableau, then we stop and consider τ as the tableau systematically generated for Σ. If a tableau rule has been applied on each compound (not a literal in FL) formula on each open path of τ, then we add X_{n+1} to each open path. If neither happens, then we choose a formula X up in the tableau on which a rule has not been applied (X has not yet been used). We choose such a node which is of minimal level (closest to the root), and if more than one such node exist, then we choose the one which is on the leftmost. We extend every open path that contains this node (formula) X, by applying the appropriate tableau rule on X, depending on what type of formula X is. If X is a stacking formula with its children (by the stacking rule) as Y, Z, then we add both Y, Z to every open path. If X is a branching formula with children Y, Z, then we extend each open path by branching out to two different paths, one with Y and the other with Z. If X is an existential formula, say $X = \exists x Y$, then we take the first term t_k from $\mathcal{T}$ which does not occur in the whole tableau, and extend each open path by adding $Y[x/t_k]$. If $X = \neg\forall x Y$, then $\neg Y[x/t_k]$ is added instead. If X is a universal formula, say, $X = \forall Y$, then we take the first t_k from $\mathcal{T}$ such that $Y[x/t_j]$ does not occur in the whole tableau, and then add $Y[x/t_j]$ and X to each open path. Similarly, if $X = \neg\exists x Y$, and t_j is the first term in $\mathcal{T}$ such that $Y[x/t_j]$ does not occur in the tableau, then we add $Y[x/t_j]$ to each open path. After the application of a tableau rule in this fashion, we finally add X_{n+1} to each open path provided there is such an X_{n+1} in Σ. Note that if Σ is finite, this addition may not be effected always. This concludes the $(n+1)$-th step of our construction.

Look at the following example for a systematic generation of a tableau.

***EXAMPLE* 4.57** Construct a systematic tableau for the set:

$$\Sigma = \{\exists x(Lx \wedge Cx), \forall x(Lx \to Mx), \neg\exists x(Mx \wedge Cx)\}$$

Solution Let $X_1 = \exists x(Lx \wedge Cx), X_2 = \forall x(Lx \to Mx), X_3 = \neg\exists x(Mx \wedge Cx)$, and $\mathcal{T} = \{t_1, t_2, \ldots\}$. Here is the systematically generated tableau:

1.	$\exists x(Lx \wedge Cx)$		X_1	(Step 1)
2.	$Lt_1 \wedge Ct_1$		$1, \exists$	(Step 2)
3.	$\forall x(Lx \to Mx)$		X_2	(Step 2)
4.	Lt_1		$2, \wedge$	(Step 3)
5.	Ct_1		$2, \wedge$	(Step 3)
6.	$\neg\exists x(Mx \wedge Cx)$		X_3	(Step 3)
7.	$Lt_1 \to Mt_1$		$3, \forall$	(Step 4)
8.	$\forall x(Lx \to Mx)$		$3, \forall$	(Step 4)
9.	$\neg(Mt_1 \wedge Ct_1)$		$6, \neg\exists$	(Step 5)
10.	$\neg\exists x(Mx \wedge Cx)$		$6, \neg\exists$	(Step 5)
11.	$\neg Lt_1$	Mt_1	$7, \to$	(Step 6)
12.	$\times$	$Lt_2 \to Mt_2$	$8, \forall$	(Step 7)
13.		$\forall x(Lx \to Mx)$	$8, \forall$	(Step 7)
14.		$\neg Mt_1 \quad \neg Ct_1$	$9, \neg\wedge$	(Step 8)
15.		$\times \quad \times$		

Compare this tableau with that in Example 4.54. A tableau generated systematically may run very lengthy, but the method guarantees that all the premises that can be used in a tableau will eventually be used. Obviously, the systematic procedure is not a good way of generating a tableau, but it is a mechanical way with book-keeping. A tableau which is systematically generated is finite after every step. However, we want to consider the net product, the infinite tree which is generated this way. This is not constructive, but is, conceptually, a tableau. Thus, we define a **systematic tableau** as one that is systematically generated and either it is infinite, or if finite, it cannot be further extended by the systematic procedure. Note that in the former case, the tableau has to be open, while in the latter case, it may be either closed or open. Moreover, it is clear that for any set of formulas Σ, there exists a closed tableau for Σ iff there exists a closed systematic tableau for Σ. This is in sharp contrast to the more creative way of generating tableaux which may sometimes fail. That is, even if there is a closed tableau for a set of formulas, you may not get it easily by arbitrarily instantiating the quantified formulas. But the systematic tableau will eventually close. On the other hand, when the systematic tableau is unmanageably large, you may be able to construct a short tableau with ingenuous instantiations.

Suppose that ρ is a path in a systematic tableau of a set of formulas Σ. Let us call ρ to be a **satisfiable path** if the set of all formulas in ρ is satisfiable. Now, if ρ is closed, is it satisfiable? Definitely not. For, if ρ is satisfiable, then there is a state in which all the formulas in ρ are assigned 1, including the two formulas of the form Z and $\neg Z$ which occur in ρ (since ρ is closed); this is impossible. Thus, a satisfiable path has to be open. A tableau is **satisfiable** if at least one of its paths is satisfiable. It follows that if a systematic tableau is satisfiable then it has at least one open path. But when is a systematic tableau satisfiable? Does satisfiability of Σ guarantee it?

Let Σ be a satisfiable set of formulas and τ be the systematic tableau for Σ. Let τ_n be the tableau generated systematically up to and including the nodes generated in step n. Since Σ is satisfiable, τ_1 is satisfiable as it contains the first premise from Σ. As for our inductive proof, we first show that if τ_n is satisfiable, then τ_{n+1} is also satisfiable. So, we assume that τ_n is satisfiable. The uninteresting case is when $\tau_{n+1} = \tau_n$; but then there is nothing to prove. Otherwise, by the induction hypothesis, τ_n has an open path. Let $Z \in \tau_n$ be the formula, in such an open path, say, ρ, which has been chosen in the step n while extending τ_n to τ_{n+1}. You are required to prove the following:

(a) If Z is a stacking formula with Z_1, Z_2 as its children (Z_2 may be absent.) then the extended path ρ, Z_1, Z_2 is satisfiable.

(b) If Z is a branching formula with Z_1, Z_2 as its children, then at least one of the extended path ρ, Z_1 and ρ, Z_2 is satisfiable.

(c) If Z is a universal formula with its children as $Z'[x/t]$, Z, then the path ρ, $Z'[x/t]$, Z is satisfiable. [*Notation*: If $Z = \forall x U$, then $Z' = U$. If $Z = \neg\exists x U$, then $Z' = \neg U$. The quantified variable is denoted by x in this case as well as in the next case.]

(d) If Z is an existential formula with its child as $Z'[x/t]$ in τ_{n+1}, then the path ρ, $Z'[x/t]$ is satisfiable.

(e) An addition of a premise from Σ to ρ does not make the extension unsatisfiable.

Note that satisfiability of the path implies the satisfiability of the extended path in each case above. This guarantees the satisfiability of τ_{n+1}. Proofs of the statements in (a)$-$(c) are straightforward. Since Σ is assumed to be satisfiable, due to the statements (a)$-$(d), the statement in (e) follows. For (d), suppose that $I_\ell = (D, \phi, \ell)$ be a state such that $I_\ell \models \rho$, where we write the set of formulas in the path ρ as ρ itself. Due to the conditions on an existential rule, there is no constraint on defining ℓ for this new term t. Extend ℓ to ℓ' by defining $\ell'(t) = \alpha$, where $\alpha \notin D$, a new symbol, and keeping the values of ℓ' as those of ℓ for all other terms. Let $D' = D \cup \{\alpha\}$. Extend the function ϕ to ϕ' by redefining $\phi(Z')$. Note that $\phi(Z')$ might have been already defined in the state I_ℓ. However, we now view $\phi(Z')$ as a relation on D' instead of D. The redefinition of this relation $\phi(Z')$ is an extension of the old $\phi(Z')$ as existing on D plus the assertion that $\phi(Z'[x/t])$ holds. For example, if $Z' = Pxa$ and we have already $\phi(P) = \{(a, b), (b, c)\}$, then $Z'[x/t] = Pta$, and the redefinition of $\phi(P)$ is $\phi'(P) = \{(a, b), (b, c), (\alpha, a)\}$, since t is assigned to α by ℓ'.

Now it is clear that the new state $I'_\ell = (D', \phi', \ell')$ is still a model of all the formulas in ρ and, in addition, it is a model of the new formula $Z'[x/t]$. Thus, the extension of ρ, i.e., the set $\rho \cup \{Z'[x/t]\}$, is satisfiable.

So we have proved that if Σ is satisfiable, then the systematic tableau τ for Σ has an open path. This means that any tableau you construct (with ingenuity, of course), is bound to remain open. Why? Because, if there is a closed tableau for Σ, then it (or one of its incarnations) would have occurred in the systematic tableau, and then the systematic tableau would close. Thus you have proved the following result.

Theorem 4.29 (Strong Soundness of FT) *Every satisfiable set of formulas is (FT-) consistent.*

In proving the soundness of FT, we have not used the properties of a systematic tableau in any essential way. We could have argued for any tableau instead, with slight modifications. But there are interesting properties of a systematic tableau. Let ρ be an open path (if it exists) in a systematic tableau for the set Σ of formulas. Let C be the set of constants and free variable appearing in ρ. Let F be the set of all function symbols appearing in ρ. Construct the set D inductively as the set of all terms generated from C by using the function symbols from F. This is the **free universe** generated from the sets C and F. Think of the Herbrand universe we had defined in Chapter 2. Let P be the set of all predicates occurring in ρ. Then ρ satisfies the following properties:

H0 : No atomic formula and its negation are both in ρ.

H1 : If $Z \in \rho$ is a stacking formula with children Z_1, Z_2, then $Z_1 \in \rho$ and $Z_2 \in \rho$. Note that Z_2 may not exist for some formulas.

H2 : If $Z \in \rho$ is a branching formula with children Z_1, Z_2, then $Z_1 \in \rho$ or $Z_2 \in \rho$.

H3 : If $Z \in \rho$ is a universal formula, then $Z'[x/t] \in \rho$ for every term $t \in D$, where either $Z = \forall x Z'$, or $Z = \neg\exists x Y$ and $Z' = \neg Y$.

H4 : If $Z \in \rho$ is an existential formula, then $Z'[x/t] \in \rho$ for at least one term $t \in D$, where either $Z = \exists x Z'$, or $Z = \neg\forall x Y$ and $Z' = \neg Y$.

Using these properties, you can show that ρ is satisfiable. How do you proceed? You must find a state model of ρ. So, start with a domain. An obvious choice is D. Choose the assignment function ℓ as the identity function, just like constructing a Herbrand interpretation. Then, following the same idea, we only have to specify which atomic formulas are declared true and which are declared false. This is accomplished by defining the function ϕ from $P - \{\approx\}$ to the set of all relations over D. Define ϕ by

$\phi(Q(t_1, t_2, \ldots, t_m)) = 1$ if $Q(t_1, t_2, \ldots, t_m) \in \rho$,

$\phi(Q(t_1, t_2, \ldots, t_m)) = 0$ if $\neg Q(t_1, t_2, \ldots, t_m) \in \rho$, and

$\phi(Q(t_1, t_2, \ldots, t_m)) = 1$ if neither happens.

For the equality predicate, define the equivalence relation E on the domain D, as was done in Section 2.8 for the construction of the Herbrand interpretation. Now, the function ϕ must satisfy the condition "$\phi(s \approx t) = 1$ iff $(s, t) \in E$". Due to the properties H0 to H4, it is clear that $(D, \phi, \ell) \models \rho$. Thus, we have shown that every open path in a systematic tableau for any set of formulas is satisfiable. That is, the tableau, in this case, is satisfiable. We note it down as a theorem below.

Theorem 4.30 (Strong Completeness of FT) *Every (tableau-) consistent set of formulas is satisfiable.*

The strong soundness and the strong completeness together can be restated in terms of the entailment relation using RAA.

Theorem 4.31 (Strong Adequacy of FT) *Let Σ be a set of formulas and w be a formula. Then $\Sigma \models w$ iff there is a closed tableau for $\Sigma \cup \{\neg w\}$.*

From the construction of a systematic tableau it also follows that if every finite subset of a set of formulas is consistent, then the set itself is consistent, which yields the compactness theorem for tableaux and also for FL due to the strong adequacy of FT. Similarly, the Skolem−Löwenheim theorem also follows from the proof of the strong soundness theorem. Can you see how? Note the way a free universe has been constructed. It also shows directly that a set of formulas is satisfiable iff the set of all existential closures of the formulas is satisfiable. Write a formal proof of this result which you have already proved in Chapter 2.

***Exercise* 4.22** Formulate and prove the compactness theorem and the Skolem−Löwenheim theorem using systematic tableaux.

The proofs above also point at another interesting fact. Whenever a systematic tableau for a set Σ of formulas closes, the set Σ is not only unsatisfiable, but its completeness says that it is enough to see whether it is unsatisfiable on a finite domain, because the free universe constructed from that systematic tableau is finite. If a systematic tableau for Σ is finite and open, then any open path in it will give rise to a finite model of Σ. What if a set is neither satisfiable nor unsatisfiable on any finite domain, but is satisfiable on an infinite domain? The systematic tableau will then have an infinite open path. But then, how to determine by the tableau method that there is an infinite open path? Obviously, the tableau method will not be able to show it, but some meta-argument might. Look at the following example.

***EXAMPLE* 4.58** Is the following set Σ of formulas satisfiable?

$$\Sigma = \{\forall x\exists yPxy, \forall x\forall y\forall z(Pxy \wedge Pyz \to Pxz), \neg\exists xPxx\}$$

Solution

$$\begin{array}{c}
\forall x\exists yPxy \\
\forall x\forall y\forall z(Pxy \wedge Pyz \to Pxz) \\
\neg\exists xPxx \\
\exists yPby \\
\neg Pbb \\
Pbc \wedge Pcd \to Pbd \\
Pbd \\
\neg Pcc \\
\neg(Pbc \wedge Pcd) \qquad Pbd
\end{array}$$

This is not a systematic tableau; you can construct one anyway. Look at the tableau carefully. We realize (Do you?) that the tableau will go on for ever by reusing the universal formulas, and it will never close. This set Σ is neither satisfiable nor unsatisfiable in any finite domain, but is satisfiable in an infinite domain. For example, you can interpret P as 'greater than' in $\mathbb{N}$ and see that Σ has a model there. As a procedure, analytic tableau fails to determine satisfiability of any arbitrary set of formulas. Does any other proof method achieve it? It seems the answer is negative. Reason? Analytic tableau is a complete proof procedure, so nothing better can be achieved. At this point, you may try with other proof procedures to determine satisfiability of Σ in Example 4.58. A somewhat convincing argument is that an algorithm is a finite procedure, and it cannot find out an answer to such an infinite construction of a model. But it needs a proof, and that is exactly the undecidability of FL, which we are not going to prove! See the summary for some references on this important result.

SUMMARY

In this chapter, you have learnt four proof methods for both PL and FL. The axiomatic system PC, the propositional calculus, has only three axioms (axiom schemes) and an inference rule, the modus ponens. Since it uses the connectives $\neg$ and $\to$ only, the other connectives and the propositional constants are introduced by definitions. The axioms have been constructed in such a way that the deduction theorem becomes easier to prove. A proof has been defined as a finite sequence of propositions where each proposition in it is either an axiom, a premise, or is obtained by an application of the inference rule on some earlier formulas. Then a conclusion of the premises is defined as the last proposition in a proof.

You have seen that PC is a strongly adequate system for PL, i.e., it is both strongly sound and strongly complete. Strong soundness means that if a proposition is a conclusion of a set of premises in PC, then the consequence is PL-valid. Strong completeness is the converse of strong soundness, i.e., every valid consequence has a proof in PC. Since finiteness of proofs is a matter of definition and the system is adequate, you have obtained the compactness theorem for PL. Compactness asserts that if a conclusion follows from an infinite number of premises, then it also follows from some finite number of those premises.

You have then learnt how to extend the system PC to FC, the first order calculus so that it becomes adequate to FL. All the metaresults hold for FC in parallel to PC, including the compactness for FL. The basic idea is to see that truth and provability are in parlance as long as our world is described by a first order language. Since the concept of provability is effective, the notion of truth becomes effective.

Though the concept of a proof is effective, construction of a proof need not be. This negative feeling is very much evidenced in both the systems PC and FC. You have seen how difficult it is to construct proofs of trivial looking tautologies. This is remedied by making informal proofs. A somewhat better choice is the natural deduction system, where there are certainly more rules of inference, but easier proofs. You have learnt the natural deduction systems PND and FND which are strongly adequate to PL and FL, respectively.

As another variation to the proof methods, you have learnt how to have a proof as a tree by way of Gentzen systems. A Gentzen system uses a different strategy than the earlier methods. It takes a consequence as its basic block in a proof rather than taking the propositions or formulas. The consequences are named as sequents in the system. This renaming is used to demarcate the line between syntactic manipulation from the semantic overload. A Gentzen system then goes on arguing with sequents by transforming them into simpler sequents. If the universal sequent is eventually generated, then the original sequent is proved, otherwise not. You have also seen that the Gentzen systems GPC and GFC are strongly adequate to PL and FL, respectively.

Finally, you came across another tree method of proof, the analytic tableaux. Though it is a tree method, it works with propositions and formulas instead of consequences. It employs *reductio ad absurdum* by starting from the negation of the conclusion as a new premise and later confirming that it leads to a contradiction. Since the proof methods have to have a syntactic way rather than the truth, it does not use the word 'contradiction', nor the notion of truth. It rather gives some syntactic rules to determine inconsistency by way of closing a tableau. You have seen that the idea of tableau-inconsistency and the semantic notion of unsatisfiability are in parlance, which is nothing but the adequacy of the tableau method. You have learnt independent (meta-) proofs of strong adequacy of analytic tableau. These proofs are instructive as they revealed directly many interesting metaresults on both PL and FL. The following bibliographic comments will be helpful to pursue the topics in greater detail.

You can find the axiomatic systems PC and FC in almost all texts in logic, though in different incarnations, see for example, [4, 17, 19, 51, 52]. The ones presented here are taken from [67]. The completeness proof for the systems follows the ideas of many logicians such as L. Henkin, J. Hintikka, G. Hasenjaeger, A. Lindenbaum, and K. Gödel; their relevant works can be found in [80]. The natural deduction systems PND and FND have been similarly presented. In particular, you may like to see [8, 41, 70] for some variations on their forms. The Gentzen systems were invented by G. Gentzen in 1935, though in a very different form. The original form of the sequents was rather close to the natural deduction systems PND and FND. The system GPC was formulated by H. Wang in the form of an algorithm while trying to write a theorem prover for PL using FORTRAN. Since then, the systems GPC and its natural extension GFC for FL have been recognized as the Gentzen systems. The texts [28, 50] provide you with complete references on Gentzen systems. The analytic tableaux had their origin in E. W. Beth's semantic trees. Two good texts on Tableaux are [22, 71]. In [71], you will also find other results concerning metamathematics. There is one more nice and useful result which we have not discussed. It is Beth's definability. It says that implicit and explicit definabilities of a concept in FL are on par. See [22, 67] for this important result; other references which deal with metamathematics or metalogic also deal with this result. I would suggest that you go through the references after solving the following problems.

PROBLEMS

1. Construct PC-proofs for the following:

(a) $\vdash ((\neg A \to \neg B) \to B) \to ((\neg A \to \neg B) \to A)$

(b) $\vdash A \to ((B \to (A \to C)) \to (B \to C))$

(c) $\vdash A \to (\neg B \to \neg\neg A)$

(d) $\vdash (A \to \neg B) \to ((B \to \neg A) \to (A \to \neg B))$

(e) $\vdash (A \to (B \to C)) \to (B \to (A \to C))$

(f) $\vdash (A \to B) \to (\neg B \to \neg A)$

(g) $\vdash \neg(A \to B) \to (C \to \neg B)$

(h) $\vdash (\neg A \to B) \to ((\neg A \to \neg B) \to A)$

(i) $\vdash (A \to B) \to ((\neg A \to B) \to B)$

(j) $\vdash \neg(\neg B \to \neg(A \to B)) \to \neg A$

(k) $\vdash A \to (\neg B \to \neg(A \to \neg(B \to (\neg A \to A))))$

2. Let Σ be any set of propositions and p, q be any propositions. Show the following in PC:

(a) if $\Sigma \vdash p$ and $\Sigma \vdash q$, then $\Sigma \vdash p \wedge q$.

(b) if at least one of $\Sigma \vdash p$ or $\Sigma \vdash q$, then $\Sigma \vdash p \vee q$.

3. Formulate and prove Craig's interpolation theorem for PC.
[*Hint*: See Problem 41 of Chapter 1.]

4. For any set of propositions Σ, show that Σ is (PC-) consistent iff there is a proposition w such that $\Sigma \nvdash w$.

5. Show that no axiom of PC can be derived from other axioms.
[*Note*: This is the *independence* of axioms.]

6. We write $Z(u, v)$ to mean that Z is a formula with free variables u, v. Construct FC-proofs for the following:

(a) $\vdash \forall x \neg X \leftrightarrow \neg \exists x X$

(b) $\vdash \exists x \neg X \leftrightarrow \neg \forall x X$

(c) $\vdash (s \approx t) \to (f(s) \approx f(t))$

(d) $\vdash (s \approx t) \to ((t \approx t') \to (s \approx t'))$

(e) $\vdash \forall x(X \to (\neg X \to Y))$

(f) $\vdash (X \to Y) \to (\forall x X \to \forall x Y)$

(g) $\vdash (X \to Y) \to (\exists x X \to \exists x Y)$

(h) $\vdash \forall x \forall y(\neg(X(x) \to \neg Y(y)) \to X(x))$

(i) $\vdash \forall x((X \to (X \to Y)) \to ((X \to X) \to (X \to Y)))$

7. Let x does not occur free in X, y does not occur free in Y, and z does not occur free in Z. Construct FC-proofs of the following:

(a) $\vdash (X \to \forall x Z) \to \forall x(X \to Z)$

(b) $\vdash (X \to \exists x Z) \to \exists x(X \to Z)$

(c) $\vdash (\exists y Z \to Y) \to \forall y(Z \to Y)$

(d) $\vdash (\forall y Z \to Y) \to \exists y(Z \to Y)$

(e) $\vdash \exists x Z \leftrightarrow \exists z Z[x/z]$

(f) $\vdash \forall x(Z \to \forall z((x \approx z) \to Z[x/z]))$

8. Show that a set Σ of formulas is consistent iff there is a formula which is not a theorem of Σ.

9. Let Σ be a (FC-) consistent set and $\Sigma \nvdash X$. Prove that $\Sigma \cup \{\neg X\}$ is consistent.

10. State and prove Craig's interpolation theorem for FC.
[*Note*: See Problem 47 of Chapter 2.]

11. Construct PND-proofs for the following:

(a) $\vdash p \to (q \to p \land q)$

(b) $\vdash (p \to q) \to (q \land r \to (p \to r))$

(c) $\vdash \neg p \to (p \to \bot)$

(d) $\vdash \neg p \to (p \to q)$

(e) $\vdash (p \lor q) \to (\neg p \land q \to q)$

(f) $\vdash (p \to q) \to ((q \to r) \to ((r \to s) \to (p \to s)))$

(g) $\{p \lor q \to r \land s, r \lor u \to \neg v \land w, v \lor x \to p \land y\} \vdash \neg v$

(h) $\{p \to \neg q, r \to s, \neg t \to q, s \to \neg u, t \to \neg v, \neg u \to w, p \land r\} \vdash \neg v \land w$

(i) $\vdash \neg(p \to (q \to r)) \lor ((p \to q) \to (p \to r))$

(j) $\{(r \to r \land s) \to t, t \to (\neg u \lor u \to p \land u), p \lor q \to (r \to r)\} \vdash p \leftrightarrow t$

12. Construct FND-proofs for the following:

(a) $\forall x(Px \land Qx) \vdash \forall xPx \to \forall xQx$

(b) $\forall x(Px \land Qx) \vdash \forall xPx \land \forall xQx$

(c) $\forall xPx \lor \forall xQx \vdash \forall x(Px \lor Qx)$

(d) $\exists x(Px \land Qx) \vdash \exists xPx \land \exists xQx$

(e) $\exists xPx \lor \exists xQx \vdash \exists x(Px \lor Qx)$

(f) $\forall x\forall y(Py \to Qx) \vdash \exists yPy \to \forall xQx$

(g) $\forall xP(x) \vdash \forall yP(y)$

(h) $\forall x(Px \to Qx) \vdash \forall x\neg Qx \to \forall x\neg Px$

(i) $\exists x\exists yP(x,y) \vdash \exists z\exists uP(z,u)$

(j) $\exists x\forall yP(x,y) \vdash \forall y\exists xP(x,y)$

(k) $\neg\forall x\neg Px \vdash \exists xPx$

(l) $Pc \vdash \forall x(x \approx c \to Px)$

(m) $\{\exists x\exists y(Pxy \lor Pyx), \neg\exists xPxx\} \vdash \exists x\exists y\neg(x \approx y)$

(n) $S \to \forall xPx \vdash \forall x(S \to Px)$, for any sentence S

(o) $X \to \forall xPx \vdash \forall x(X \to Px)$, if x is not free in X

(p) $\{\forall xPaxx, \forall x\forall y\forall z(Pxyz \to Pf(x)yf(z))\} \vdash Pf(a)af(a)$

(q) $\{\forall x Paxx, \forall x \forall y \forall z(Pxyz \to Pf(x)yf(z))\} \vdash \exists z Pf(a)zf(f(a))$

(r) $\{\forall x Pcx, \forall x \forall y(Pxy \to Pf(x)f(y))\} \vdash \exists z(Pcz \wedge Pzf(f(c)))$

13. Show that if two boxes are allowed to overlap, then the soundness of PND and FND may not be retained.

14. Using tableau rules, determine satisfiability of the following sets of propositions or formulas:

(a) $\{A \wedge B \to C, \neg A \to D, B \wedge \neg C \wedge \neg D\}$

(b) $\{A \vee B, A \vee (B \wedge D), A \to \neg C\}$

(c) $\{A \to B, (A \wedge B) \vee \neg C, B \to \neg A, C \to D, B \leftrightarrow \neg D, A \vee \neg E\}$

(d) $\{\exists x Px, \neg Pc\}$

(e) $\{\exists x Px \wedge \exists x Qx, \neg \exists x(Px \wedge Qx), \forall x Px \to Ps\}$

(f) $\{\forall x(Px \to Qx), \exists x Px, \forall x \neg Qx, \exists x Px \vee \neg Pc\}$

15. Construct GFC- and FT-proofs of all the formulas in Problems 1, 6, 7, 11, 12, and 14.

16. Are the following arguments valid? Use PND, FND, GPC, GFC, PT, and/or FT to solve this problem.

(a) Either Logic is elegant or many students like it. If Computer Science is a difficult discipline, then Logic has to be elegant. Therefore, if many students like Computer Science, then Logic is elegant.

(b) If tomorrow morning it is chilly and not so clear a sky, then we go swimming unless you do not have any special liking for boating. It isn't always the case that if the sky is not clear, then you don't go boating. Therefore, if the weather is not chilly tomorrow morning, then we go swimming.

(c) Yanka would have been at home, had he been to the club or not to the theatre while his friend was waiting for him at the college. He had been to the club premises while it was dusk only if he didn't come home. Unless the watchman saw him at the club premises, it is certain that it was dusk, since the watchman is night blind. Therefore, Yanka did not go to the theatre.

(d) If anyone is in a guest house, which is located in a city, then he is in that city. None can be in two different cities at the same time. One who snatched the necklace away must have been in the guest house at Chennai. Since Chennai and Mumbai are two different cities and Yanka was in the Mumbai guest house, he did not snatch the necklace.

(e) If all the politicians praise a bandit, then the bandit must be none other than Robin Hood. A politician praises a person if the person has indeed helped him in his career. There is a bandit who has not helped any politician. Therefore, there is a bandit other than Robin Hood.

17. Consider the proof systems PC, PND, GPC, and PT. Assume adequacy of one of them and try proving adequacy of all the others to PL.

18. Consider the proof systems FC, FND, GFC, and FT. Assume adequacy of one of them and try proving adequacy of all the others to FL.

19. Are the rules of PND (and FND) independent? If yes, show that no rule can be derived from others taken together. If no, then pick the rule(s) which can be derived from others. Resolve the problem of independence of axioms and rules for the other proof systems such as PC, FC, GPC, GFC, PT, and FT.

5

Program Verification

5.1 Introduction

What do you do when you write a program? Of course, before you start writing a program, you know the job at hand, and the computer language you would be using to write it. Typically, you would have an idea as to what are the inputs to your program and what would be the required outputs. You must also know how the program would perform the job of getting the outputs from the inputs. For example, you are asked to write a program for checking whether two strings are equal. Immediately you think of a shortcut. If the given strings are of different lengths, then they are not equal. So, a subclass of the problems is solved. Once the strings are found to be of the same length, you are going to check them character by character. So, this describes *how* the job is done. Now, how to write this procedure as a program? Well, you imagine a Boolean variable, say, *done*, which is assigned a value *true* or *false* accordingly as the given strings s_1 and s_2 are equal or not. Assume that you had already a ready-made program to compute the length of a string, denoted by $length(s)$. Suppose that the language in which you are writing your program uses '=' for comparing two natural numbers. That is, the expression '$5 = 5$' is equivalent to *true* and '$5 = 6$' is equivalent to *false*. Then you would like to write a line:

$$done := (length(s_1) = length(s_2))$$

This says that the variable *done* is assigned the value of the expression $length(s_1) = length(s_2)$ which is either true or false depending upon whether the length of the string s_1 is equal to the length of s_2 or not.

If $length(s_1)$ is not equal to $length(s_2)$, then the variable *done* is assigned *false*, and you have the required result. Otherwise, you conclude that the strings s_1 and s_2 require further treatment. In this case, the variable *done* is assigned the value *true* and $length(s_1) = length(s_2) = m$, say. Now to match the strings s_1 and s_2 character by character, suppose that you already have a program that computes the i-th character of a string; you can use it by calling the function $ch(i, s)$. That is, $ch(i, s)$ finds out the i-th character of the string s_1. Given inputs i, a positive integer and s, a string, the output of $ch(i, s)$ is the i-th character of the string s. Again, suppose that you can compare the characters by the same relation '='. That is, the expression $ch(i, s_1) = ch(j, s_2)$ is computed as *true* if the i-th character of the string s_1 is the same as the j-th character of the string s_2; else, it is evaluated to *false*. Then it would be OK to write:

$$done := (ch(i, s_1) = ch(i, s_2)) \text{ for all } i \text{ from 1 to } m$$

So, you have the program:

```
PROGRAM : StringMatching
  $done := (length(s_1) = length(s_2))$
  if $done$ then for $i = 1$ to $length(s_1)$
    do $done := (ch(i, s_1) = ch(i, s_2))$
```

If there is a stake on writing this program correctly, you would not just leave it here. You would like to test the program on some inputs. As a student of logic, you start testing the program on strings (a) which are identical, (b) which have unequal lengths, and then strings (c) which are unequal but of equal lengths. This phase of program development is usually called **debugging**. (Programmers use a face saving word, 'bug', instead of 'mistake'.) As test inputs you run the program on strings:

(a) $s_1 = mat, s_2 = mat$; $s_1 = chalk, s_2 = chalk$;
$s_1 = blackboard, s_2 = blackboard$, etc.

(b) $s_1 = mat, s_2 = chalk$; $s_1 = chalk, s_2 = blackboard$;
$s_1 = duster, s_2 = blackboard$; $s_1 = pencil, s_2 = paper$, etc.

(c) $s_1 = mat, s_2 = pad$; $s_1 = chalk, s_2 = paper$;
$s_1 = blackboard, s_2 = bluepencil$, etc.

Now, in all the runs, your program was a success and you have submitted your program to your instructor on Friday. The following Monday your instructor announces in the class that you have been awarded with zero marks as the program is incorrect. To justify himself, he says that your program evaluates *done* to *true* for the inputs $s_1 := mat, s_2 := cat$. You then verify and see that he is correct. You have lost your quiz marks. Imagine what would happen to a company where they would have invested in crores on a problem for which you have supplied the software; and that too, the software uses the string matching program you have written. Once this happened to the Intel Pentium chips having a flaw in multiplication of floating point numbers, and they had to bear the loss. The moral:

Debugging is not a foolproof method of program development.

Debugging cannot prove a correct program correct. It is not a reliable method to prove an incorrect program incorrect. If it finds, mostly by chance, a mistake in a program, then of course, the program is incorrect. To encapsulate, E. W. Dijkstra said, "Program testing can be used to show the presence of bugs, but never to show their absence".

This chapter addresses the issue of how to prove that a program does its intended job. This is referred to as proving correctness of a program or verifying a program. You will see how logic comes of help in achieving this.

5.2 Issue of Correctness

We have thus deliberated on the disadvantages of debugging a program. Though it is useful, it is not foolproof. Then how to show that a program does its intended job; how to prove the correctness of a program? For example, consider the program *StringMatching* that we have developed in Section 5.1 for matching two strings. You see that there is a clear flaw in it; it finally assigns *true* to the variable *done* when two strings s_1 and s_2 are of equal length having the same last character. You see, the keyword here is 'finally'. What happens if a program goes on an infinite loop? You can instruct a person in a treasure hunt to move to a place where he finds the instruction for moving back to the original place again. Unless you want to make a Sysiphus out of a computer, such a program may not do the intended job. However, in some circumstances you may want the computer not to come to a halt stage unless it is switched off manually; for example, a surveillance system. Otherwise, for most jobs, you may need your program to come to a finishing stage, i.e., it should terminate and give you the required output. Thus it is worthwhile to determine whether a program terminates at all with any of the possible specified inputs. The next important issue is, whether it gives the expected output after termination. Nevertheless, we are interested in having both the properties of a program, and more importantly, in proving that both the properties hold for a program that you may write.

The first issue is called the problem of **program termination**, and the second is called the **partial** or **conditional correctness** of a program. Taken together, we have the **total correctness** of a program. Given a program P, we now have the problems as:

Termination: whether P terminates for all the specified inputs?

Partial Correctness: Given that P terminates for all specified inputs, does P ends up in delivering the required output, corresponding to each input?

Total Correctness: For each specified input whether P delivers the corresponding output?

Thus P is totally correct iff P terminates and P is partially correct. Once the issues are clear, we must proceed to see how to settle them. How do we prove termination, or partial or total correctness of a program? Note that we have to take care of the 'specified inputs' and the 'corresponding outputs'.

Well, what is an input and what is an output of a program P? For example, take P as the program *StringMatching* of Section 5.1. What are the inputs to this program? A pair of strings. Means? One string such as *mat* is assigned to the **program variable** s_1, and another string, say *cat*, is assigned to the program variable s_2. And what is an expected output? It is the value *true* or *false* assigned to the program variable *done*. For the program P, the input may be described as $s_1 = mat \wedge s_2 = cat$, and its output as you know in this case, is $done = true$. In abstract terms, the program variables satisfy some property before the execution of the program. This property, called the **precondition** of the program, is described here by the (first order) formula

$$s_1 = mat \wedge s_2 = cat$$

Similarly, some property holds for the program variables after the execution of the program. This property, called the **postcondition** of the program, is described here by the formula:

$$done = true$$

Of course, we are interested in a specified or required postcondition only. The postcondition $done = false$ is not the required postcondition when we have the strings as *mat* and *cat*. In developing a program, our main intention is to meet the required postcondition by manipulating the program variables. A **state** of a program is simply described by what values are assigned to the program variables. A typical state of our program P is $s_1 = mat, s_2 = cat, done = false$. Then a precondition of P is simply a property satisfied by a state of P. Here the precondition $s_1 = mat \wedge s_2 = cat$ is such a property. Note that all program variables need not be assigned values in a precondition. That is not a constraint since an unassigned variable can be assigned each possible value in turn and then $\vee$-ed together. For example, our precondition for P can be rewritten using the equivalence:

$$s_1 = mat \wedge s_2 = cat \equiv s_1 = mat \wedge s_2 = cat \wedge (done = false \vee done = true)$$

What we learn here is that, instead of talking of inputs and outputs, we may speak of preconditions and postconditions, which are first order properties satisfied by a state of the program. Moreover, the precondition is satisfied by all states of the program initially, and the postcondition must be satisfied by all states finally, in order that the program is (conditionally) correct. To argue about programs, we introduce a notation for this triple of a precondition, a postcondition, and the program itself. Such a triple is loosely called a **specification** of a program and is also called a **Hoare triple** after the computer scientist C. A. R. Hoare. The triple has the syntax:

$$\langle Q \rangle \; S \; \langle R \rangle$$

where S is a program, a sequence of program statements, Q and R are some properties of the program variables, the variables used in the program S. The predicate Q is the precondition and R is the postcondition of the program S. The semantics of the Hoare triple is the following:

> $\langle\, Q \,\rangle\, S \,\langle\, R \,\rangle$ means that, if initially the state of the program variables satisfies the predicate Q and the program S is executed, then on termination of S, the final state of the program variables will satisfy the predicate R.

Thus the Hoare triple expresses conditional correctness of the program S. It says that if you assume the termination of S, then it is guaranteed that the postcondition R is satisfied after the execution provided that the precondition Q is (has been) satisfied before the execution. However, to argue about conditional correctness of a program, say, for the string matching program P, you will not be able to consider all inputs separately. Will you try proving that all of the following Hoare triples hold?

$$\langle\, s_1 = mat \wedge s_2 = cat \,\rangle\;\; P\;\; \langle\, done = false \,\rangle$$

$$\langle\, s_1 = chalk \wedge s_2 = duster \,\rangle\;\; P\;\; \langle\, done = false \,\rangle$$

$$\langle\, s_1 = bottle \wedge s_2 = pencil \,\rangle\;\; P\;\; \langle\, done = false \,\rangle$$

$$\vdots$$

There are certainly an infinite number of such Hoare triples to be verified to hold. We would rather like to introduce some new variables for the strings. Suppose we write all possible strings to be assigned to s_1 as $st1$ and those to s_2 as $st2$. Then a typical precondition might look like

$$s_1 = st1 \wedge s_2 = st2$$

where the required postcondition might be

$$st1 = st2 \text{ if and only if } done = true$$

that is,

$$st1 = st2 \leftrightarrow done = true$$

Thus the infinite number of Hoare triples would be represented by the specification

$$\langle\, s_1 = st1 \wedge s_2 = st2 \,\rangle\;\; P\;\; \langle\, st1 = st2 \leftrightarrow done = true \,\rangle$$

The new variables $st1$ and $st2$ are called the **ghost variables** or **logical variables** with respect to the program S. These logical variables simply relate the final values of the program variables to their initial values. The logical variables do not occur in the program; they only occur in the logical representations of the precondition and the postcondition. We repeat:

> The above specification for the program P states that if P at all terminates, and if initially, the variables s_1, s_2 are assigned the values $st1, st2$, respectively, (no matter what is assigned to the variable *done*) then finally the formula $st1 = st2 \leftrightarrow done = true$ will hold.

Our notation of a Hoare triple captures the concept of partial correctness only. We will address the issue of total correctness, but in a latter section. However, to argue about a program, we must also describe what a program is. You have already written many programs, may be, in many languages such as BASIC, FORTRAN, PASCAL, C, and PROLOG. As you know, syntax and semantics of commands vary from one language to another. How are we going to argue about them all? Our plan is to develop a methodology for arguing about programs independent of such languages. If we consider any one of these languages, then our arguments would become specialized to that language. Further, we would like to consider procedural languages only, thus excluding languages like PROLOG and LISP. For procedural languages the core may be thought of as comprising three types of commands. They are the assignment statements, the conditional statements, and the loop statements. Since Turing machines are accepted as definitions of algorithms, it is enough if you can simulate a Turing machine by using these three types of core statements. And this can be done since a Turing machine requires symbol writing, conditional movement of the read-write head, and repeating certain required sequence of moves. Thus, we take up this path of defining a core language comprising these three types of statements along with the capability of sequential execution, so that statements in a program are executed one after another for preserving compositionality of a program. We describe the syntax and semantics of our core language in the following section. All our programs will be assumed to be written in this language.

5.3 The Core Language CL

We have decided to keep the four constructs of assignment statement, conditional statement, looping statement, and the sequential control of execution in our core language. We begin with the assignment statement.

If x is a variable and E is an expression, then an **assignment statement** looks like

$$x := E$$

We use := instead of the already overused symbol = to remember that it assigns the value of the expression E to the variable x. Variables are simply memory locations in a computer. Here, in the abstract language CL, you may regard them as some names, such as $x, y, z, \ldots, x_0, x_1, x_2, \ldots$, or even as *string*, *book*, *string*1, ... just like identifiers in procedural programming languages. Further, we limit the expressions E to be integer expressions only, so that things will become easier without sacrificing much on the capabilities of CL. In fact, an integer expression is any integer from the set $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$, or any variable, or is in any of the forms: $-E1$, $(E1 + E2)$, $(E1 - E2)$, $(E1 * E2)$, or $(E1/E2)$, where $E1, E2$ are any integer expressions. This recursive definition will let us generate integer expressions such as $(5 * (-3 + (x * x)))$, $((2 * 3) * 4), \ldots$ which are usually called as the arithmetic expressions in some procedural languages.

For doing logical checks, we also keep Boolean expressions. These are simply the propositions of PL, but now have a syntax including the integer expressions. To be definite, a Boolean expression is one of the following:

$$\top, \bot, \neg B1, (B1 \wedge B2), (B1 \vee B2), (E1 < E2), \text{ or } (E1 = E2)$$

where $B1, B2$ are some Boolean expressions and $E1, E2$ are integer expressions. Note that there is some redundancy in keeping $(E1 = E2)$ since it could be defined by using the connective $\neg$ and the predicate $<$. However, allowing '$=$' will simplify long expressions, and we decide to keep it. The symbol '$-$' represents both the unary 'minus' and the binary 'subtraction' as in -3 and $(5-3)$, respectively; $+, *$ represent the operations of addition, and multiplication of integers as usual. Similarly, $=$ and $<$ mean the usual equality and 'less than' relations of integers. Note that we are deviating from our earlier notation for equality. We write (in this chapter,) the usual $=$ instead of the symbol $\approx$. The Boolean connectives $\neg, \wedge, \vee$ are the usual negation, conjunction, and disjunction, respectively. The constants $\top, \bot$ represent the truth values '*true*' and '*false*', and the elements in $\mathbb{Z}$ are the usual integers. Execution of an assignment statement of the form

$$x := E$$

proceeds as follows: the expression E is evaluated and the computed value is stored in the name (memory location) x. After the execution, the variable becomes **bound** to the value of E. Boolean expressions are used to describe conditional statements just as integer expressions are used to define the syntax of an assignment statement.

A **conditional statement** looks like

$$\texttt{if}\ C\ \texttt{then}\ \{S1\}\ \texttt{else}\ \{S2\}$$

where C is a condition, a Boolean expression, and $S1, S2$ are some statements. The curly brackets are used as punctuation marks as in C and JAVA. For example,

$$\texttt{if}\ (m < n)\ \texttt{then}\ \{m := n\}\ \texttt{else}\ \{n := m\}$$

is a conditional statement. The execution of the conditional statement

$$S\text{:}\ \ \texttt{if}\ C\ \texttt{then}\ \{S1\}\ \texttt{else}\ \{S2\}$$

proceeds as follows. First, the Boolean expression C is evaluated. If its value turns out to be $\top$, i.e., true, then the statement $S1$ is executed; and after this execution, the whole statement S is considered executed. On the other hand, if the value of C turns out to be $\bot$, i.e., false, then the statement $S2$ is executed instead of $S1$; and after this execution, execution of S is considered over. For example, consider the statement S as

$\texttt{if}\ (m < n)\ \texttt{then}\ \{m := n\}\ \texttt{else}\ \{n := m\}$

Suppose that before S is executed the variables m, n have been bound to 2 and 3, respectively. Then the condition $(m < n)$ is evaluated to $\top$, and consequently, execution of S proceeds in executing the statement $m := n$. After the execution of S, m becomes bound to 3, the value of n, and also n is bound to 3 as before. If m, n have been bound to 10, 5, respectively, before the execution of S, then after S is executed, both m and n become bound to 5, the value of m. This is so because, in that case, the condition $(m < n)$ is evaluated to $\bot$; consequently, execution of S forces execution of $n := m$, the statement that comes after `else`.

The **sequential control statement** or the **composition** has the following syntax:

$S1\,;\,S2$

where $S1$ and $S2$ are any statements or programs (program fragments). Its semantics is obvious. First, $S1$ is executed. After this execution is over, $S2$ is executed. After the execution of $S2$, the execution of S is considered over. If $S1$ does not terminate, then $S2$ is never executed, and in that case, the composition program $S1\,;\,S2$ does not terminate. Thus, termination of $S1\,;\,S2$ is guaranteed only when both $S1$ and $S2$ terminate, and in that order.

Finally, we describe the looping statement. Among many alternatives, we choose the 'while loop format'. A while statement looks like

$\texttt{while}\ C\{S\}$

where C is a condition, a Boolean expression, and S is a statement (a program or a program fragment). Its execution is done in the following way. The Boolean expression C is first evaluated. If its value is $\top$, then the statement S is executed. Upon termination of this execution, the Boolean expression C is again evaluated. If its value is $\top$ this time, then S is once again executed. The loop of 'evaluation of C' followed by 'execution of S' is repeated. This repeated execution continues till the value of C becomes $\bot$. Thus, in practice, it is often the case that execution of S changes some program variables occurring in C so that after a finite number of repeated executions (evaluation of C and then execution of S), C would eventually become bound to $\bot$. If this does not happen, then the while statement would never terminate. Thus, vacuously, if C and S do not share any program variables and C is initially bound to $\top$, then the while statement will not terminate. Moreover, at the termination of the while statement, the condition C must be bound to $\bot$. In a while statement, the condition C is called the **guard**, and the statement S is called the **body** of the while loop.

The while statement is a looping statement meant for repeated execution of its body until its guard is falsified. It is like a 'for command' found in procedural languages. The difference is that in a for command, you know beforehand how many times S would be repeatedly executed, whereas in a while statement, this is not known beforehand. A while statement is more suitable for recursion, whereas a for command is more suitable for iteration. The while statement has more direct connection to a machine scheme representation of Turing machines. However, it comes with the potential danger of nontermination. For example, consider the statement:

S: `while` $m \neq n\{m := m + 1\,;\ s := s + m\}$

Suppose that before execution of S, the variable m has been bound to 0, n has been bound to 5, and s has been bound to 0. With this initial state, execution of S begins. Now, the guard $m \neq n$ is satisfied (evaluated to $\top$) as $0 \neq 5$; consequently, the body $m := m+1\,;\ s := s+m$ is executed. After this first execution of the body, m is bound to 1, s is bound to $s+m$, i.e., to 1. As S is to be executed repeatedly until the guard $m \neq n$ becomes false, once more execution of S is initiated. Again, the guard $m \neq n$ is satisfied and consequently, the body is executed. This makes m bound to 2 and, s bound to 3. After five executions of the body, it turns out that m is bound to 5 and s is bound to 15. On the sixth execution, the guard $m \neq n$ is found to be false, as m becomes equal to 5, the value of n. Then execution of S is stopped; S terminates with the state satisfying $m = 5 \wedge n = 5 \wedge s = 15$.

In this detailed trace of execution of S, you must have noticed that there are some variables whose values have not changed while the values of some other variables have changed. The ones whose values have changed are m, s, and the value of n has not changed. As the program computes the sum of numbers $0, 1, \ldots, n$, it is this changed (final) value of s which stores the result. Though s changes, it is unlike the change in m. The variable m goes successively through the natural numbers. The change in s can be captured by a formula by looking at its pattern of change at every successive execution of the body of S. When the body of S is executed once, s contains the value 1. At the end of the second execution, it contains the value $1 + 2$. In general, at the end of the i-th execution, s contains the value $1 + 2 + \ldots + i$. Now you see that though the value of s is changing through the repeated execution of the body of the loop, the statement that

After the i-th execution of the body of S, $s = 1 + 2 + \ldots + i$

does not change. This statement holds before the execution of S, it holds after S terminates, and it even holds throughout the execution of the loop S. Such a property which holds throughout the execution of a loop is called an *invariant* of the loop. It is often the invariant of a loop that signals towards the real job undertaken by the loop. Getting the invariant(s) of a loop is not mechanical; it needs some experience, but often it comes from the requirements or the postcondition. We will see in the following section, how an invariant is used for proving correctness of programs.

5.4 Partial Correctness

Before going into proofs of partial correctness of programs, we summarize the discussions of the last two sections. It will serve as a background material for the proofs we will study in this section. All our programs will be written in the language CL. Whenever some new constructs like arrays or lists are required, we will introduce them informally, without making any hassle and sacrificing on the precise notions of syntax and semantics. You will be able to formalize the constructs with your rich experience in the syntax and semantics of logical languages like PL and FL. Once the new constructs are understood, we will assume that necessary formalities can be framed accordingly. Our attention will be focused on the control structures of CL. For a balanced view of formalism and informals, we have already introduced the integer and Boolean expressions, which, for the time being, will be all that we require to understand the control structures.

To recollect, the language CL has two types of expressions: integer and Boolean. These expressions are used in defining statements, which are of four types. The syntax of CL is summarized by the following BNF:

$$E ::= n \,|\, x \,|\, -E \,|\, (E+E) \,|\, (E-E) \,|\, (E*E) \,|\, (E/E)$$

$$B ::= \top \,|\, \bot \,|\, \neg B \,|\, (B\wedge B) \,|\, (B \vee B) \,|\, (E=E) \,|\, (E<E)$$

$$S ::= x := E \,|\, S\,;\, S \,|\, \texttt{if}\ B\ \texttt{then}\ \{S\}\ \texttt{else}\ \{S\} \,|\, \texttt{while}\ B\{S\}$$

We write $x \neq y$ as a shorthand for $\neg(x = y)$. Similarly, we write $x \to y$ for $\neg x \vee y$ and $x \leftrightarrow y$ for $(x \to y)\wedge(y \to x)$. We assume the usual properties of the operations $+, *, -, /$ and of the connectives $\neg, \wedge, \vee$, etc. Note that the m/n is the integral part of the number $m \div n$, and it presumes that n is not equal to 0. However, in the preconditions and postconditions, we (sometimes) use the symbol '$/$' for real number division instead of this integer division. You should be able to interpret it correctly from the given context. We also assume the usual precedence of operations and connectives. To argue about programs, which are statements in CL, we have introduced the Hoare triple, which has the syntax:

$$\langle\, Q\,\rangle\ S\,\langle\, R\,\rangle$$

where Q and R are FL-formulas; the precondition and the postcondition of the statement (program) S, respectively. To distinguish between the **specification** (a Hoare triple) $\langle\, Q\,\rangle\, S\,\langle\, R\,\rangle$ and the fact that '$\langle\, Q\,\rangle\, S\,\langle\, R\,\rangle$ is partially correct', we introduce a notation for the latter. We say that the specification $\langle\, Q\,\rangle\, S\,\langle\, R\,\rangle$ is **partially correct** iff for all states of the program S, which satisfy the precondition Q, the states resulting from the execution of S satisfy R, provided that S actually terminates. Whenever $\langle\, Q\,\rangle\, S\,\langle\, R\,\rangle$ is partially correct, we write $\models_p \langle\, Q\,\rangle\, S\,\langle\, R\,\rangle$. In such a case, we also say that the triple $\langle\, Q\,\rangle\, S\,\langle\, R\,\rangle$ is **satisfied** under partial correctness. We reserve the notation $\langle\, Q\,\rangle\, S\,\langle\, R\,\rangle$ for the abstract specification of our requirement which may be read as:

> When the precondition Q is satisfied by all states of the program variables of S, upon termination of S, the resulting states satisfy the postcondition R.

Then $\models_p \langle Q \rangle\, S\, \langle R \rangle$ means that $\langle Q \rangle\, S\, \langle R \rangle$ holds provided S terminates. Basically, it is the difference between a sentence A and the metasentence that 'A holds'. For arguing about correctness of programs, we simply argue about the specification of the program.

***EXAMPLE* 5.1** $\models_p \langle\, x = x_0 \,\rangle\ y := x \,\langle\, y = x_0 \,\rangle$

Solution Let s be a state of program variables, which satisfies the precondition $x = x_0$. That is, in state s, we have $x = x_0$ and y can be bound to anything whatsoever. Here x and y are the program variables and x_0 is a logical variable. Now being in state s, the statement $y := x$ is executed. After its execution, y becomes bound to the value contained in x. That is, $y = x_0$. Thus, the resulting state $\bar{s}$ satisfies $x = x_0 \wedge y = x_0$. we see that $\bar{s}$ satisfies the postcondition $y = x_0$. Hence the specification

$$\langle\, x = x_0 \,\rangle\ y := x \,\langle\, y = x_0 \,\rangle$$

is partially correct, i.e., $\models_p \langle\, x = x_0 \,\rangle\ y := x \,\langle\, y = x_0 \,\rangle$.

***EXAMPLE* 5.2** $\models_p \langle\, x = y \,\rangle\ x := x + 1\,;\, y := y + 1 \,\langle\, x = y \,\rangle$

Solution Let s be a state satisfying the precondition $x = y$. here x and y may have been bound to any (integer) value. Once the state s satisfies $x = y$, we have in state s, $x = x_0$ and $y = x_0$ for some x_0. Now the statement $x := x + 1\,;\, y := y + 1$ is executed. After the execution of $x := x+1$, the new state $\bar{s}$ satisfies $x = x_0+1 \wedge y = x_0$. Then the statement $y := y+1$ is executed. After this execution, the new state s' satisfies (with $\bar{s}$ as the initial state for $y := y+1$) the formula $x = x_0+1 \wedge y = x_0+1$. We see that the postcondition $x = y$ is satisfied by the final state s'. Therefore, $\models_p \langle\, x = y \,\rangle\ x := x + 1\,;\, y := y + 1 \,\langle\, x = y \,\rangle$.

***EXAMPLE* 5.3** $\models_p \langle\, (i = j \to k = j) \wedge (i \neq j \to k = m) \,\rangle$
$\quad$ `if` $i = j$ `then` $\{m := k\}$ `else` $\{j := k\}$
$\quad$ $\langle\, j = k \wedge j = m \,\rangle$

Solution Let s be a state satisfying the precondition. Using equivalences, we see that the precondition

$$\begin{aligned} &(i = j \to k = j) \wedge (i \neq j \to k = m) \\ &\quad\equiv\ (i \neq j \vee k = j) \wedge (i = j \vee k = m) \\ &\quad\equiv\ (i \neq j \wedge k = m) \vee (k = j \wedge i = j) \vee (k = j \wedge k = m) \end{aligned}$$

Initially, s satisfies this formula. Now, the statement

S: `if` $i = j$ `then` $\{m := k\}$ `else` $\{j := k\}$

is executed. The condition $i = j$ may or may not be satisfied by s. Thus we have two cases to consider.

In case s satisfies $i = j$, we also see that s satisfies $k = j$ (while $k = m$ may or may not be satisfied). Execution of S then proceeds to execute the statement $m := k$. Here ends the execution of S, resulting in a new state $\bar{s}$ satisfying $i = j \wedge k = j \wedge m = k$, i.e., $\bar{s}$ satisfies the postcondition $j = k \wedge j = m$.

On the other hand, when s satisfies $i \neq j$, we see that s also satisfies $k = m$. Now, since the condition $i = j$ of S is evaluated $\bot$ (in the state s), execution of S initiates the execution of the statement $j := k$ (look after the punctuation 'else'). This brings in a new and final state $\bar{s}$ satisfying $k = m \wedge j = k$. That is, $\bar{s}$ satisfies the postcondition $j = k \wedge j = m$.

Thus the specification is partially correct.

***EXAMPLE* 5.4** $\models_p \langle\, i = 0 \wedge \sigma = 0 \wedge n > 0 \,\rangle$
$\quad$ `while` $i \neq n \,\{i := i + 1\,;\, \sigma := \sigma + 1\}$
$\quad$ $\langle\, \sigma = \sum_{i=0}^{n} i \wedge i \leq n \,\rangle$

Solution Let s be any state where $i = 0, \sigma = 0, n = n_0$ with $n_0 \geq 0$. This is the only way s might satisfy the precondition. When execution of

S: `while` $i \neq n\{i := i + 1\,;\, \sigma := \sigma + i\}$

is initiated, the guard $i \neq n$ is first evaluated. If $n_0 = 0$, then this guard is not satisfied, and consequently, execution of S is terminated. The new state, in this case, s itself, obviously satisfies the postcondition as $\sigma = 0$ and $\sum_{i=0}^{n_0} i = \sum_{i=0}^{0} i = 0$. If $n_0 = 0$, then the guard $i \neq n$ is satisfied and the body $i := i + 1\,;\, \sigma = \sigma + i$ is executed to have the resulting state $\bar{s}$ which satisfies

$$i = 1,\ \sigma = 0 + 1 = \textstyle\sum_{i=0}^{n_0} i = \sum_{i=0}^{n} i$$

That is, $\bar{s}$ satisfies the postcondition. You can repeat this argument for any n_0. But this will not prove the statement. What you require is that for every natural number n_0, the resulting state $\bar{s}$ will satisfy the postcondition. So, induction? All right, for $n_0 = 0$, you have already shown the partial correctness. Now lay out the induction hypothesis that for $n_0 = m \in \mathbb{N}$, partial correctness holds. Let $n_0 = m + 1$. Let s be any state satisfying the precondition. Since $n_0 \geq 1$ and $i = 0$, the guard $i \neq n$ is evaluated $\top$, and the body is then executed. This yields a new state $\tilde{s}$ which satisfies $i = 1 \wedge \sigma = \sum_{i=0}^{1} i$. So, how to use the induction hypothesis? We are stuck! Leaving it here amounts to the hand waving remark: "proceed up to $n_0 + 1$ and similarly ... " – not a proof!

***Exercise* 5.1** First show that the number of times the body of the loop in Example 5.4 is executed is n_0. Then use induction on the number of times the body of the loop is executed to complete the proof of partial correctness.

We have not completed the proof of partial correctness in Example 5.4 because you have to slow down a bit. This example shows that we cannot always argue the way we used to do in the previous examples. It is like proving the validity of a first order formula using semantics. It is better to have some proof system. To have one such, we will abstract away the peculiarities in the above examples and, instead, arrive at a consensus on how to handle the basic four types of statements of CL.

We take up the assignment statement first. Its specification looks like

$$\langle\, Q \,\rangle\, x := E \,\langle\, R \,\rangle$$

where x is a program variable, E is an arithmetic expression and Q, R are FL-formulas. When can we assert that $\models_p \langle\, Q \,\rangle\, x := E \,\langle\, R \,\rangle$? By definition, if s is a state which satisfies Q, after x is assigned the value of E, s must satisfy R. In Example 5.1, you have seen such a specification. Since the value of x is changed to the value of E, you must see that either $Q[x/E]$ happens to be R or $R[x/E]$ happens to be Q. Which way should it happen? Whenever $\langle\, Q \,\rangle\, x := E \,\langle\, R \,\rangle$ is partially correct, you see that R is satisfied for a state $s[x \rightarrowtail E]$. The state $s[x \rightarrowtail E]$ is the same as in FL; it is the environment obtained from s by fixing the variable x to E. Now, $\bar{s} = s[x \rightarrowtail E]$ satisfies R. That means $R[x/E]$ must have been satisfied by s. Thus, Q must be equal to $R[x/E]$ or at least that Q must imply $R[x/E]$. We will break these two alternatives into two rules. The latter case will be taken care of by a more general rule. The former rule will be taken as the rule of assignment, or the assignment axiom. The **Assignment Axiom** is then expressed as

$$\models_p \langle\, R[x/E] \,\rangle\, x := E \,\langle\, R \,\rangle$$

Note that $\models_p \langle\, Q \,\rangle\, x := E \,\langle\, Q[x/E] \,\rangle$ is not correct. For example, take $Q \equiv x = 5, E = 2$, then had it been correct, we would have obtained the specification $\langle\, x = 5 \,\rangle\, x := 2 \,\langle\, (x = 5)[x/2] \,\rangle$, which is the same as $\langle\, x = 5 \,\rangle\, x := 2 \,\langle\, 2 = 5 \,\rangle$ as a partially correct specification. But this is not so; why? Suppose that s is a state of the program variables satisfying the precondition $x = 5$. If $x := 2$ is executed, then the resulting state satisfies $x = 2$. But it can never satisfy the formula $2 = 5$, as $(2 = 5) \equiv \bot$. However, the assignment axiom allows $\models_p \langle\, 2 = 2 \,\rangle\, x := 2 \,\langle\, x = 2 \,\rangle$, the truth of which can be seen clearly, since $(2 = 2) = (x = 2)[x/2]$.

The assignment axiom demands backward reasoning, from the postcondition to the precondition. We will see shortly how to use the axiom in proving partial correctness of programs. To see that it is all right, argue with the specification $\langle\, x = 5 \,\rangle\, x := 2 \,\langle\, x = 2 \,\rangle$. This is partially correct since for any state s satisfying $x = 5$ before the execution of the statement $x := 2$, the resulting state after the execution will satisfy $x = 2$. It does not really matter whether the precondition is $x = 5$; it could have been any Q. The reason is: $(x = 5) \rightarrow (2 = 2)$ is a valid FL-formula.

This suggests the rule that "if $Q \models R[x/E]$, then $\models_p \langle\, Q \,\rangle\, x := E \,\langle\, R \,\rangle$ holds". Similarly, if P is any formula such that $R \models P$, then also we have $\models_p \langle\, R[x/E] \,\rangle\, x := E \,\langle\, P \,\rangle$. For example, $\models_p \langle\, 2 = 2 \,\rangle\, x := 2 \,\langle\, x^2 = 2^2 \,\rangle$ since $(x = 2) \models (x^2 = 2^2)$ is a valid consequence in FL. These observations are true for any type of statements, and not just for assignment. We may express it as

If $P \models Q, \models_p \langle\, Q \,\rangle\, S \,\langle\, R \,\rangle$, and $R \models U$, then $\models_p \langle\, P \,\rangle\, S \,\langle\, U \,\rangle$.

We call this rule as the **Rule of Implication**.

***Exercise* 5.2** Using the assignment axiom and the rule of implication, show the partial correctness of the following specifications:

(a) $\langle\, x = y \,\rangle\, x := x + 1 \,\langle\, x = y + 1 \,\rangle$

(b) $\langle\, x = y + 1 \,\rangle\, y := y + 1 \,\langle\, x = y \,\rangle$

Next, we consider the composition or the sequential execution. When can we assert that $\langle\, Q \,\rangle\, S1\,;\, S2 \,\langle\, R \,\rangle$ is partially correct? Note that we are reasoning backward, from the postcondition to the precondition. If after the execution of $S1\,;\, S2$, a state satisfies R, then what would have happened before the execution? Let us first think about the execution of $S2$. Before $S2$ has been executed, there must have been some state that satisfies its precondition so that R is satisfied by the resulting state after the execution of $S2$. Call the precondition P. That is, we hypothesize the partial correctness of the specification

$\langle\, P \,\rangle\, S2 \,\langle\, R \,\rangle$

Then, obviously, P must have been satisfied by a resulting state after the execution of $S1$, which was initiated by the state s satisfying, say, Q. That is, the partial correctness of the specification

$\langle\, Q \,\rangle\, S1 \,\langle\, P \,\rangle$

could have been obtained earlier. This argument is encapsulated in the **Rule of Sequential Execution**, or the **Rule of Composition**, which says that

If $\models_p \langle\, Q \,\rangle\, S1 \,\langle\, P \,\rangle$, $\models_p \langle\, P \,\rangle\, S2 \,\langle\, R \,\rangle$, then $\models_p \langle\, Q \,\rangle\, S1\,;\, S2 \,\langle\, R \,\rangle$.

Note that for proving $\models_p \langle\, Q \,\rangle\, S1\,;\, S2 \,\langle\, R \,\rangle$ some such P will do provided that the other two specifications are partially correct. Look back at Example 5.2 now. There, you have obtained the following:

$\models_p \langle\, x = y \,\rangle\, x := x+1 \,\langle\, x = y+1 \,\rangle$, $\models_p \langle\, x = y+1 \,\rangle\, y := y+1 \,\langle\, x = y \,\rangle$

From these two partially correct specifications, we derive, by the rule of sequential execution, that $\models_p \langle\, x = y \,\rangle\, x := x + 1\,;\, y := y + 1 \,\langle\, x = y \,\rangle$.

***Exercise* 5.3** By using assignment axiom and the rule of sequential composition show that

$\models_p \langle\, \sigma = \sum_{m=0}^{i} m \wedge i \neq n \,\rangle\, i := i + 1\,;\, \sigma := \sigma + i \,\langle\, \sigma = \sum_{m=0}^{i} m \,\rangle$

Consider the conditional statement. We want to decide when does

$$\models_p \langle\, Q \,\rangle \ \texttt{if}\ B\ \texttt{then}\ \{S1\}\ \texttt{else}\ \{S2\}\, \langle\, R \,\rangle$$

hold. Go back to Example 5.3. There, we had to break the proof into two cases depending on whether the Boolean expression B is evaluated to $\top$ or $\bot$. If s is a state that satisfies Q and B is evaluated to $\top$, then in this state $S1$ is executed. After such an execution the resulting state must satisfy R. This suggests the partial correctness of the specification $\langle\, Q \wedge R \,\rangle\, S1\, \langle\, R \,\rangle$. On the other hand, when B is evaluated to $\bot$, i.e., when $\neg B$ holds, the initial state s satisfies $Q \wedge \neg B$. In this case, $S2$ is executed and the resulting state satisfies R. This suggests, similarly, the partial correctness of the specification $\langle\, Q \wedge \neg B \,\rangle\, S2\, \langle\, R \,\rangle$. Remember that we are going from the postcondition to the precondition. Summarizing, we have the **Rule of Conditional statement** as

If $\models_p \langle\, Q \wedge B \,\rangle\, S1\, \langle\, R \,\rangle$ and $\models_p \langle\, Q \wedge \neg B \,\rangle\, S2\, \langle\, R \,\rangle$,
then $\models_p \langle\, Q \,\rangle$ $\texttt{if}\ B\ \texttt{then}\ \{S1\}\ \texttt{else}\ \{S2\}\, \langle\, R \,\rangle$.

In Example 5.3, we had, first of all,

$$\models_p \langle\, (i = j \rightarrow k = j) \wedge (i \neq j \rightarrow k = m) \wedge i = j \,\rangle\, m := k\, \langle\, j = k \wedge j = m \,\rangle$$

$$\models_p \langle\, (i = j \rightarrow k = j) \wedge (i \neq j \rightarrow k = m) \wedge i \neq j \,\rangle\, j := k\, \langle\, j = k \wedge j = m \,\rangle$$

By using the rule of conditional statement, we obtain

$$\models_p \langle\, (i = j \rightarrow k = j) \wedge (i \neq j \rightarrow k = m) \,\rangle$$
$$\texttt{if}\ i = j\ \texttt{then}\ \{m = k\}\ \texttt{else}\ \{j := k\}$$
$$\langle\, j = k \wedge j = m \,\rangle$$

***Exercise* 5.4** By using the assignment axiom and the rule of implication, show that the following statements hold:

(a) $\models_p \langle\, (i = j \rightarrow k = j) \wedge (i \neq j \rightarrow k = m) \wedge i = j \,\rangle\, m := k\, \langle\, j = k \wedge j = m \,\rangle$

(b) $\models_p \langle\, (i = j \rightarrow k = j) \wedge (i \neq j \rightarrow k = m) \wedge i \neq j \,\rangle\, j := k\, \langle\, j = k \wedge j = m \,\rangle$

Finally, we consider the while statement. Here the task is to find out sufficient conditions (which should be reasonably weak) for determining the partial correctness of the specification

$$\langle\, Q \,\rangle\ \texttt{while}\ B\{S\}\, \langle\, R \,\rangle$$

Look back at Example 5.4. We have seen that there are properties that change during repeated executions of the loop body, and there are some which do not change. Recollect that the properties that do not change are called the **invariants** of the loop. An invariant remains true (not false, for convenience) before, during, and after the execution of the loop. The invariant, in some way, comes out of the job that the loop does. The invariant in Example 5.4 is the 'current sum up to i', where i is the number of times the loop body has been executed (till that instant). That is, the invariant is the formula: $\sigma = \sum_{j=0}^{i} j$. Though the value of the variable i changes, and the value of the variable σ changes, the truth (value) of the statement $\sigma = \sum_{j=0}^{i} j$ does not change.

Now, suppose that we know what the invariant of a while statement is. In order that $\langle\, Q \,\rangle$ `while` $B\{S\}\,\langle\, R \,\rangle$ holds, we know that Q must contain the invariant, and R must also contain the invariant. Moreover, after the execution, assuming that the loop terminates, B must become false. Otherwise, the statement S is executed again contrary to the termination of the loop. That is, R must also contain $\neg B$. Denoting the loop invariant by I, we seek the conditions so that the specification

$$\langle\, I \,\rangle\ \texttt{while}\, B\{S\}\,\langle\, I \wedge \neg B \,\rangle$$

is partially correct. Let us look at its execution once again. Here, s is a state of the program variables satisfying the invariant I. In such a state, execution of the while statement `while` $B\{S\}$ is initiated. Suppose that s satisfies the guard B. Then S is executed. After this execution, the invariant is satisfied by the resulting state. If B is also satisfied by the resulting state, then once again S is executed, and once again it results in a state that satisfies the invariant. Hence, in order that $\langle\, I \,\rangle$ `while` $B\{S\}\,\langle\, I \wedge \neg B \,\rangle$ is partially correct, we would have had the partial correctness of $\langle\, I \wedge B \,\rangle\, S\,\langle\, I \,\rangle$. We thus have the **Rule of While** as

$$\text{If} \models_p \langle\, I \wedge B \,\rangle\, S\,\langle\, I \,\rangle, \text{ then} \models_p \langle\, I \,\rangle\ \texttt{while}\, B\{S\}\,\langle\, I \wedge \neg B \,\rangle\,.$$

Here I is an invariant of the while loop. In fact, an invariant may be defined as any formula so that the above statement holds. Once more, look at Example 5.4. Pretend that we want to discover an invariant of the loop, i.e., a formula which would make the above statement hold. We start with the postcondition. The postcondition is $\sigma = \sum_{i=0}^{n} i \wedge i \le n$. Since negation of the guard will be satisfied after termination, we will have $i = n$. This will of course imply the formula $i \le n$. Then, matching with $I \wedge \neg B$, we may consider I to be something that would express $\sigma = \sum_{i=0}^{n} i = \sum_{m=0}^{n} m$ upon termination. Note that upon termination, we have $i = n$. Moreover, the running sum of the i-th execution of the loop is $\sum_{m=0}^{i}$; and when $i = n$, we would get σ. So, let us start with $I \equiv \sigma = \sum_{m=0}^{i} m$ as an invariant. Due to the identity

$$\sigma = \textstyle\sum_{m=0}^{i} m \wedge i = m \models \sigma = \sum_{i=0}^{n} i \wedge i \le n$$

and the rule of implication, it is enough to show that

$$\begin{array}{l} \models_p \langle\, i = 0 \wedge \sigma = 0 \wedge n \ge 0 \,\rangle \\ \quad \texttt{while}\, i \ne n\{i := i + 1\,;\, \sigma := \sigma + i\} \\ \quad \langle\, \sigma = \sum_{m=0}^{i} m \wedge i = n \,\rangle \end{array}$$

Look at the postcondition again. It is $I \wedge \neg B$. Then to apply the while rule, we require $\models_p \langle\, I \wedge B \,\rangle\, i := i+1\,;\, \sigma := \sigma + i \,\langle\, I \,\rangle$, i.e.,

$$\models_p \langle\, \sigma = \textstyle\sum_{m=0}^{i} m \wedge i \neq n \,\rangle\, i := i+1\,;\, \sigma := \sigma + i \,\langle\, \sigma = \textstyle\sum_{m=0}^{i} m \,\rangle$$

To complete the proof, use the rule of implication, the consequence

$$(i = 0 \wedge \sigma = 0 \wedge n > 0) \models (\sigma = \textstyle\sum_{m=0}^{i} m \wedge i \neq n)$$

and Exercise 5.3.

Here is a summary of the rules for arguing about programs in CL; the names of the rules are self-explanatory: 'A' the for the assignment axiom, 'S' for the sequential execution, 'C' for the conditional statement, 'W' for the while statement, and 'I' for implication.

(RA) $$\frac{\cdot}{\langle\, R[x/E] \,\rangle\, x := E \,\langle\, R \,\rangle}$$

(RS) $$\frac{\models_p \langle\, Q \,\rangle\, S1 \,\langle\, P \,\rangle \qquad \models_p \langle\, P \,\rangle\, S2 \,\langle\, R \,\rangle}{\models_p \langle\, Q \,\rangle\, S1\,;\, S2 \,\langle\, R \,\rangle}$$

(RC) $$\frac{\models_p \langle\, Q \wedge B \,\rangle\, S1 \,\langle\, R \,\rangle \qquad \models_p \langle\, Q \wedge \neg B \,\rangle\, S2 \,\langle\, R \,\rangle}{\models_p \langle\, Q \,\rangle\ \texttt{if}\, B\, \texttt{then}\, \{S1\}\ \texttt{else}\, \{S2\} \,\langle\, R \,\rangle}$$

(RW) $$\frac{\models_p \langle\, I \wedge B \,\rangle\, S \,\langle\, I \,\rangle}{\models_p \langle\, I \,\rangle\ \texttt{while}\, B\{S\} \,\langle\, I \wedge \neg B \,\rangle}$$

(RI) $$\frac{P \models Q \qquad \models_p \langle\, Q \,\rangle\, S \,\langle\, R \,\rangle \qquad R \models U}{\models_p \langle\, P \,\rangle\, S \,\langle\, U \,\rangle}$$

Remember that $R[x/E]$ is the result of substitution of all free occurrences of the variable x by the expression E in R. We assume that the symbols when used in a context are well defined. For example, $R[x/E]$ must be well defined, meaning that the types of x and E must match. Note that in FL, this discussion was unwarranted, since variables there could only be substituted by terms. Similarly, sequential composition of programs or statements is left associative, i.e., $S1\,;\, S2\,;\, S3$ is to be read as $(S1\,;\, S2)\,;\, S3$. At this point go back to the rules as stated earlier and match them with the above written as fractions. These rules say that if you have already proved the partial correctness of the specification in the numerator, then the denominator follows. The denominator of RA does not need the partial correctness of any specification; it is an axiom.

You can use these rules to define and construct proofs in the style of an axiomatic system or as a tree. In the first style, a **proof of partial correctness** of a specification is a finite sequence of specifications, and possibly some FL-formulas. The formulas must be valid (or provable in any proof system), each specification is either an axiom (RA), or follows from earlier specifications/formulas by an application of one of the other four rules. Further, RA terminates a path, in the sense that no more justification is required along that path, just like Gentzen systems. The logic defined by these rules is obviously an extension of FL. Occasionally we will have to prove the valid FL-formulas that may be used in a proof. But we will refrain from doing so here; rather we give more attention to the proofs of specifications. The proof system so obtained is known as the **Hoare Logic** for CL. If you take a different language, say, PASCAL, then the corresponding Hoare logic will be different. The proofs in any Hoare logic is called a **Hoare proof**.

We follow the three-column style of writing a proof; the first column keeps track of the entries by giving each a serial number, the second column is the actual proof, and the third column documents the proof by giving justification. As justifications we will mention the rules by their names RA, RS, RC, RW, or RI as appropriate, and 'FL' for the valid FL-formulas. Occasionally, we may add a separate proof of the valid formulas. If a rule is applied on the preceding line(s), then we will not mention the line numbers; otherwise we will mention the remote line numbers.

***EXAMPLE* 5.5** Give a Hoare proof of

$$\begin{aligned}&\models_p \langle\, 1 + a - a^2 = 0 \wedge a^n = b + c * a \,\rangle\\ &\quad n := n + 1\,;\; m := b + c\,;\; b := c\,;\; c := m\\ &\quad \langle\, a^n = b + c * a \,\rangle\end{aligned}$$

Solution Remember that we have to proceed from the postcondition to the precondition. In this specification only assignment statements are sequentially composed. We may only require the rules RA,RS, and RI. The last statement in the program is $c := m$ with the postcondition $a^n = b+c*a$. So, what should be the precondition? By RA, it is $(a^n = b+c*a)[c/m]$ which equals $a^n = b + m * a$. This is taken as a postcondition for the statement $b := c$, preceding the last statement. Again, by RA, the precondition would be $(a^n = b + m * a)[b/c]$ which equals $a^n = c + m * a$. This is, essentially, the rule RS. Read the following proof backward first to understand how it is constructed:

1.	$\langle\, a^n = b + m * a \,\rangle\, c := m \,\langle\, a^n = b + c * a \,\rangle$	RA
2.	$\langle\, a^n = c + m * a \,\rangle\, b := c \,\langle\, a^n = b + m * a \,\rangle$	RA
3.	$\langle\, a^n = c + (b + c) * a \,\rangle\, m := b + c \,\langle\, a^n = c + (b + c) * a \,\rangle$	RA
4.	$\langle\, a^{n+1} = c + (b + c) * a \,\rangle\, n := n + 1 \,\langle\, a^n = c + (b + c) * a \,\rangle$	RA
5.	$\langle\, a^{n+1} = c + (b + c) * a \,\rangle$ $n := n + 1\,;\; m := b + c \,\langle\, a^n = c + m * a \,\rangle$	RS
6.	$\langle\, a^{n+1} = c + (b + c) * a \,\rangle$ $n := n + 1\,;\; m := b + c\,;\; b := c \,\langle\, a^n = b + m * a \,\rangle$	2, RS
7.	$\langle\, a^{n+1} = c + (b + c) * a \,\rangle$	

$n := n + 1\,;\ m := b + c\,;\ b := c\,;\ c := m\,\langle\, a^n = b + c * a\,\rangle$ 1, RS

8. $1 + a - a^2 = 0 \wedge a^n = b + c * a \rightarrow a^{n+1} = c + (b + c) * a$ FL

9. $\langle\, 1 + a - a^2 = 0 \wedge a^n = b + c * a\,\rangle$
$n := n + 1\,;\ m := b + c\,;\ b := c\,;\ c := m\,\langle\, a^n = b + c * a\,\rangle$ RI

A proof of validity of the formula in line 8 goes as follows:

$$\begin{aligned}
&1 + a - a^2 = 0 \wedge a^n = b + c * a\\
&\quad\Rightarrow\ a^{n+1} = a * (b + c * a) \wedge 1 + a = a^2\\
&\quad\Rightarrow\ a^{n+1} = a * b + a^2 * c \wedge 1 + a = a^2\\
&\quad\Rightarrow\ a^{n+1} = a * b + (1 + a) * c \wedge 1 + a = a^2\\
&\quad\Rightarrow\ a^{n+1} = a * b + c + a * c \wedge 1 + a = a^2\\
&\quad\Rightarrow\ a^{n+1} = c + (b + c) * a \wedge 1 + a = a^2\\
&\quad\Rightarrow\ a^{n+1} = c + (b + c) * a
\end{aligned}$$

A proof tree, called a **Hoare proof tree** can be defined by using the rules from denominators to the numerators using branching of the tree for the rules RS, RC, and RI, and stacking for the rules RA and RW. Note that unlike Gentzen systems or the analytic tableaux, the rules are written here upside down. The leaves of a Hoare proof tree must be axioms of the

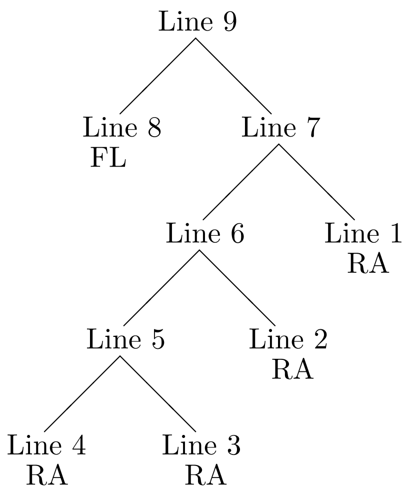


Figure 5.1 Hoare proof tree for Example 5.5

Hoare logic for CL, i.e., instances of the assignment axiom or the valid formulas of FL. The root of the proof tree must be the specification for which a proof is sought.

The Hoare proof tree for the statement of Example 5.5 is shown in Figure 5.1. In this tree we use the specifications and formulas used in the Hoare proof of Example 5.5 by mentioning the line numbers. Rewrite the whole tree first by inserting the expression at the right place where only a line number is given. Look at the tree and try to understand the phrase: 'from denominator to numerator'.

5.5 Hoare Proof and Proof Summary

Though both the styles are inconvenient, the axiomatic style looks easier than the tree style, and we follow it here. The only inconvenience in this style is the application of the assignment axiom. But nothing better is expected, not even by writing the postcondition on the left and the precondition on the right. We will rather gain experience in working out more proofs. It is often desirable to omit the small details in a proof and only sketch the important steps so that a formal proof may be constructed out of the sketch. To develop such a sketch, called a **proof summary**, let us go through the Hoare proof given in Example 5.5. A proof summary is as follows. We number the lines below for further reference, though they are not part of the proof summary. Read the proof summary from bottom to top and compare with the original Hoare proof.

Proof summary for Example 5.5

1. $\langle\, 1 + a - a^2 = 0 \wedge a^n = b + c * a \,\rangle$
2. $\langle\, a^{n+1} = c + (b + c) * a \,\rangle$
3. $n := n + 1\,;$
4. $\langle\, a^n = c + (b + c) * a \,\rangle$
5. $m := b + c\,;$
6. $\langle\, a^n = b + m * a \,\rangle$
7. $b := c\,;$
8. $\langle\, a^n = b + m * a \,\rangle$
9. $c := m$
10. $\langle\, a^n = b + c * a \,\rangle$

What do you find? The last three lines of the proof summary match with line 1 of the original Hoare proof. Lines 6, 7 and 8 match with line 2 of the original. Lines 4, 5 and 6 correspond to line 3, and lines 2, 3 and 4 correspond to line 4 of the Hoare proof. The other lines in the original Hoare proof have been omitted, where the rule of sequential execution has worked implicitly in the proof summary. In the proof summary, lines 1 and 2 comprise a **verification condition** which corresponds to the FL-consequence in line 8 of the (detailed) original proof. In the proof summaries, we will not mention the line numbers, but sometimes, we will mention the rule that has been applied at a particular stage to go for the next. Note that the verification conditions are written as

$\langle P \rangle \; \langle Q \rangle$

written one below the other. This is suggestive as it says that P is the precondition and Q is the postcondition of the 'empty code'. A real fun! It means that the Hoare logic is an extension of FL, where $P \models Q$ is rewritten as $\langle P \rangle \; \langle Q \rangle$. Recollect that for a specification for the sequential execution of the type $\langle P \rangle \, S1 \, \langle Q \rangle$, $\langle Q \rangle \, S2 \, \langle R \rangle$, the proof summary is written easily mentioning Q only once instead of repeating it twice in the middle, i.e., as $\langle P \rangle \, S1 \, ; \; \langle Q \rangle \, S2 \, \langle R \rangle$. How should we abbreviate the specification for the conditional execution? The rule of the conditional (RC) looks like

From $\langle Q \wedge B \rangle \, S1 \, \langle R \rangle$ and $\langle Q \wedge \neg B \rangle \, S2 \, \langle R \rangle$,
derive $\langle Q \rangle$ `if` B `then` $\{S1\}$ `else` $\{S2\}$ $\langle R \rangle$.

All the notations of the three specifications occurring in the above must be true in the abbreviation, and we must be able to read it through. The proof summary for this fragment would look like

$\langle Q \rangle$ `if` B `then` $\{ \langle Q \wedge B \rangle \, S1 \, \langle R \rangle \}$ `else` $\{ \langle Q \wedge \neg B \rangle \, S2 \, \langle R \rangle \} \, \langle R \rangle$

This is because, for the partial correctness of the specification

$\langle Q \rangle$ `if` B `then` $\{S1\}$ `else` $\{S2\}$ $\langle R \rangle$

we require partial correctness of both the specifications $\langle Q \wedge B \rangle \, S1 \, \langle R \rangle$ and $\langle Q \wedge \neg B \rangle \, S2 \, \langle R \rangle$. Henceforth, we use these abbreviations in constructing the proof summaries. Note that there are repetitions of the postcondition inside the braces following `then` and `else`. See the following example.

***EXAMPLE* 5.6** Construct a proof summary for showing that

$\models_p \langle \top \rangle$ `if` $i < j$ `then` $\{$ `if` $j < k$ `then` $\{m := k\}$ `else` $\{m := j\}\}$
`else` $\{$ `if` $i < k$ `then` $\{m := k\}$ `else` $\{m := i\}\} \, \langle m \geq i \wedge m \geq j \wedge m \geq k \rangle$.

Solution For this specification, we start with the proof summary as:

```
⟨ ⊤ ⟩ if $i < j$ then
{ ⟨ ⊤ ∧ $i < j$ ⟩ if $j < k$ then {$m := k$} else {$m := j$}
  ⟨ $m \geq i \wedge m \geq j \wedge m \geq k$ ⟩ }
 else { ⟨ ⊤ ∧ $i \not< j$ ⟩ if $i < k$ then {$m := k$} else {$m := i$}
   ⟨ $m \geq i \wedge m \geq j \wedge m \geq k$ ⟩ }
⟨ $m \geq i \wedge m \geq j \wedge m \geq k$ ⟩
```

But this is not a proof summary because the two conditional statements themselves involve other statements, and necessary preconditions and postconditions are not yet supplied. We work through them to obtain a complete proof summary. In the following proof summary, we use indentations to logically group the conditional statements together. Still there are repetitions of the postconditions of conditional statements in it. We will not omit the repetitions for better readability.

```
$\langle \top \rangle$ if $i < j$ then
$\{ \langle \top \wedge i < j \rangle$ if $j < k$ then
    $\{ \langle \top \wedge i < j \wedge j < k \rangle \, m := k \, \langle m \geq i \wedge m \geq j \wedge m \geq k \rangle \}$
    else $\{ \langle \top \wedge i < j \wedge j \not< k \rangle \, m := j \, \langle m \geq i \wedge m \geq j \wedge m \geq k \rangle \}$
  $\langle m \geq i \wedge m \geq j \wedge m \geq k \rangle \}$
else $\{ \langle \top \wedge i \not< j \rangle$ if $i < k$ then
    $\{ \langle \top \wedge i \not< j \wedge i < k \rangle \, m := k \, \langle m \geq i \wedge m \geq j \wedge m \geq k \rangle \}$
    else $\{ \langle \top \wedge i \not< j \wedge i \not< k \rangle \, m := i \, \langle m \geq i \wedge m \geq j \wedge m \geq k \rangle \}$
  $\langle m \geq i \wedge m \geq j \wedge m \geq k \rangle \}$
$\langle m \geq i \wedge m \geq j \wedge m \geq k \rangle$
```

Read the proof summary from the bottom to the top and see how the rules are working and how the abbreviations help us for better reading of a Hoare proof. To complete the proof, however, you must do the following exercise.

***Exercise* 5.5** Show that the following specifications are partially correct by constructing Hoare proofs:

(a) $\langle \top \wedge i \not< j \wedge i \not< k \rangle \, m := i \, \langle m \geq i \wedge m \geq j \wedge m \geq k \rangle$

(b) $\langle \top \wedge i \not< j \wedge i < k \rangle \, m := k \, \langle m \geq i \wedge m \geq j \wedge m \geq k \rangle$

(c) $\langle \top \wedge i < j \wedge j \not< k \rangle \, m := j \, \langle m \geq i \wedge m \geq j \wedge m \geq k \rangle$

(d) $\langle \top \wedge i < j \wedge j < k \rangle \, m := k \, \langle m \geq i \wedge m \geq j \wedge m \geq k \rangle$

A proof summary shows applications of all the rules except the assignment axiom. Thus, the only thing that remains in a proof summary are the assignment statements and the verification conditions. Once you verify that the precondition and postcondition pairs correctly specify the assignments and the verification conditions (the FL-consequences) hold, you get a complete proof from a proof summary. Now, what about the while statement? The while rule looks like

From $\langle Q \wedge B \rangle \, S \, \langle Q \rangle$, derive $\langle Q \rangle$ `while` $B\{S\} \, \langle Q \wedge \neg B \rangle$.

Note that in this notation, Q is the invariant of the while loop. It is instructive to keep this special property and mark it as the invariant, for it requires a lot of ingenuity to discover this. Of course, we keep all information on the specifications of the premise $\langle Q \wedge B \rangle \, S \, \langle Q \rangle$ and of the conclusion $\langle Q \rangle$ `while` $B\{S\} \, \langle Q \wedge \neg B \rangle$ so that we can easily read through the proof summary. The fragment of the proof summary for the while statement thus looks like:

$$\langle \text{Invariant: } Q \rangle \ \texttt{while}\, B\{ \langle Q \wedge B \rangle \, S \, \langle Q \rangle \} \langle Q \wedge \neg B \rangle$$

Mark the movement of the curly brackets '{' and '}' above from $\{S\}$ to $\{ \langle Q \wedge B \rangle \, S \, \langle Q \rangle \}$. We take up this movement of the curly brackets in order that $\langle Q \rangle$ and $\langle Q \wedge \neg B \rangle$ would not come together; for, that would mean $Q \models Q \wedge \neg B$, which is completely unwarranted. The word 'Invariant' inside the precondition, $\langle \text{Invariant: } Q \rangle$, is a reminder for us that Q is not only the precondition but also the invariant of the while loop.

An invariant of the while statement $\texttt{while}\, B\{S\}$ having guard B and body S is an FL-formula I such that $\models_p \langle I \wedge B \rangle \, S \, \langle I \rangle$ holds, i.e., if I and B are both satisfied by any state of the program variables (occurring in S) and S is executed, then upon termination (assumed here) of S, the resulting state will satisfy the formula I. All that is required of I is that it holds both before and after execution; it may not be satisfied by some intermediate state while execution is in progress.

The disturbing fact is that there are always many invariants of a while loop. This is simple to see since both $\top$ and $\bot$ are invariants of every while statement. (Prove it!) But then these might be the useless invariants. The useful invariants express a relationship between the program variables that are manipulated in the body of the loop. If we have the specification

$$\langle P \rangle \ \texttt{while}\, B\{S\} \, \langle R \rangle$$

then in order to be able to prove it (by RW and RI), we must look for an invariant Q such that all of

$$\models P \to Q, \ \models Q \wedge \neg B \to R, \text{ and } \models_p \langle Q \rangle \ \texttt{while}\, B\{S\} \, \langle Q \wedge \neg B \rangle$$

hold. It requires ingenuity and knowledge on the intended job the while statement is supposed to perform. The while statement is unlike any other statements which are purely mechanical. So, you need some experience in working with the while statement and this may guide you for discovering the useful invariants.

***EXAMPLE* 5.7** Construct a proof summary for the specification

$$\langle 0 \le n \rangle \, y := 1 \, ; \, z := x \, ; \, k := n \, ;$$
$$\texttt{while}\, 0 \ne k\{k := k - 1 \, ; \, y := y * z\} \, \langle y = x^n \rangle$$

Solution What is the (useful) invariant of the while loop here? Since the postcondition of the while statement will be in the form $Q \wedge \neg B$, we must look for an invariant Q so that $Q \wedge \neg B \equiv y = x^n$ or $Q \wedge \neg B \models y = x^n$, or even the weaker consequence $Q \wedge \neg B \Rightarrow y = x^n$, where B is the guard $0 \ne k$ of the loop. We must also have a formula P such that

$$\models_p \langle 0 \le n \rangle \, y := 1 \, ; \, z := x \, ; \, k := n \, \langle P \rangle, \text{ and } P \models Q.$$

Note also that the knowledge of what the loop does will come of help in hypothesizing the invariant(s). What does the loop do? Suppose that $k = 2$. The guard $0 \ne k$ is satisfied. Now, $k := k - 1$ is executed, so $k = 1$.

Next, $y := y * z$ is executed, thus in place of y, we now have the value of $y * z$. Once again, the guard is satisfied, and then k becomes 0, and y becomes bound to $(y * z) * z$, i.e., to $y * z^2$. Therefore, finally, y becomes bound to $y * z^k$. The postcondition is $y = x^n$. Combining these two, we would obtain an invariant which looks like

$$y * z^k = x^n$$

All these suggestions implicitly assume that $0 \le k$. To make it explicit, we start with the invariant Q as

$$0 \le k \land y * z^k = x^n$$

Then our specification is pushed one step closer to a proof summary, which, at this stage, appears as

$$\begin{array}{l}
\langle\, 0 \le n \,\rangle \\
y := 1\,;\ z := x\,;\ k := n\,; \\
\langle\, P \,\rangle\ \langle\, Q \,\rangle \\
\texttt{while}\ 0 \ne k\{k := k - 1\,;\ y := y * z\} \\
\langle\, 0 = k \land Q \,\rangle\ \langle\, y = x^n \,\rangle
\end{array}$$

We do not know yet what P is. Trying with $P = Q$ (instead of $P \models Q$), the specification would be simplified. Further, using the specification of a proof summary for the while statement, it would look like

$$\begin{array}{l}
\langle\, 0 \le n \,\rangle \\
y := 1\,;\ z := x\,;\ k := n\,; \\
\langle\, \text{Invariant: } Q \,\rangle \\
\texttt{while}\ 0 \ne k\{\ \langle\, Q \land 0 \ne k \,\rangle\, k := k - 1\,;\ y := y * z\, \langle\, Q \,\rangle\ \} \\
\langle\, 0 = k \land Q \,\rangle\ \langle\, y = x^n \,\rangle
\end{array}$$

We now focus our attention on the first part:

$$\langle\, 0 \le n \,\rangle\, y := 1\,;\ z := x\,;\ k := n\, \langle\, Q \,\rangle$$

Pushing the postcondition towards the precondition through the assignments, we see that a proof summary of this fragment starts with

$$\langle\, 0 \le n \,\rangle\, y := 1\,;\ z := x\, \langle\, Q[k/n] \,\rangle\, k := n\,;\ \langle\, Q \,\rangle$$

And, finally, the fragment will be

$$\begin{array}{l}
\langle\, 0 \le n \,\rangle\ \langle\, Q[k/n][z/x][y/1] \,\rangle\, y := 1\,;\ \langle\, Q[k/n][z/x] \,\rangle \\
z := x\,;\ \langle\, Q[k/n] \,\rangle\, k := n\,;\ \langle\, Q \,\rangle
\end{array}$$

Taking Q as the formula $0 \le k \land y * z^k = x^n$, the substitutions above will be simplified. With that, use the fragment for the while statement as done earlier to obtain the required proof summary as:

$$
\begin{array}{l}
\langle\, 0 \le n \,\rangle \\
\langle\, 0 \le n \vee a * x^n = x^n \,\rangle \\
y := 1\,; \\
\langle\, 0 \le n \wedge y * x^n = x^n \,\rangle \\
z := x\,; \\
\langle\, 0 \le n \wedge y * z^n = x^n \,\rangle \\
k := n\,; \\
\langle\, \text{Invariant: } 0 \le k \wedge y * z^k = x^n \,\rangle \\
\texttt{while}\, 0 \ne k \{ \\
\langle\, 0 \le k \wedge y * z^k = x^n \wedge 0 \ne k \,\rangle \\
\langle\, 0 \le k - 1 \wedge (y * z) * z^{k-1} = x^n \,\rangle \\
k := k - 1\,; \\
\langle\, 0 \le k \wedge (y * z) * z^k = x^n \,\rangle \\
y := y * z \\
\langle\, 0 \le k \wedge y * z^k = x^n \,\rangle\, \} \\
\langle\, 0 \le k \wedge y * z^k = x^n \wedge 0 = k \,\rangle \\
\langle\, y = x^n \,\rangle
\end{array}
$$

***Exercise* 5.6** Prove the three verification conditions met in the proof summary in Example 5.7. They are

(a) $0 \le n \models 0 \le n \wedge 1 * x^n = x^n$

(b) $0 \le k \wedge y * z^k = x^n \wedge 0 \ne k \models 0 \le k - 1 \wedge (y * z) * z^{k-1} = x^n$

(c) $0 \le k \wedge y * z^k = x^n \wedge 0 = k \models y = x^n$

Develop a complete Hoare proof from the proof summary.

***EXAMPLE* 5.8** Let $odd(k)$ be a predicate expressing the fact that k is odd. Formally, $odd(k) \equiv \exists m(m \in \mathbb{N} \wedge k = 2 * m + 1)$. Construct a proof summary with the necessary verification condition(s) for the following specification:

$\langle\, 0 \le n \,\rangle\, k := n\,;\, y := 1\,;\, z := x\,;$ `while` $0 \ne k \{$ `if` $odd(k)$ `then`

$\{k := k - 1\,;\, y := y * z\}$ `else` $\{y := y\}\,;\, k := k/2\,;\, z := z * z\}\, \langle\, y = x^n \,\rangle$

Solution To give a hint, first check that $Q(y, z, k) \equiv y * z^k = x^n \wedge 0 \le k$ is an invariant of the while statement. Solve it yourself before reading further. Prove the verification conditions in the following proof summary.

Writing $Q(y, z, k)$ instead of Q will be helpful, in the sense that it will allow us to write $Q(x, z, k)$ in place of $Q(y, z, k)[y/x]$. Introduce the predicate $even(k) \equiv \exists m(m \in \mathbb{N} \wedge k = 2 * m)$. Clearly, $\neg odd(k) \equiv even(k)$. The proof summary now looks like

```
$\langle\, 0 \le n \,\rangle$
$\langle\, 1 * x^n = x^n \wedge 0 \le n \,\rangle$
$k := n\,;\; y := 1\,;\; z := x\,;$
$\langle\,$Invariant: $Q(y,z,k)\,\rangle$
while $0 \ne k$ {
  $\langle\, 0 \ne k \wedge Q(y,z,k)\,\rangle$
  if $odd(k)$ then {
    $\langle\, odd(k) \wedge 0 \ne k \wedge Q(y,z,k)\,\rangle$
    $\langle\, even(k-1) \wedge Q(y*z, z, k-1)\,\rangle$
    $k := k-1\,;\; y := y*z$
    $\langle\, even(k) \wedge Q(y,z,k)\,\rangle$ }
  else { $\langle\, \neg odd(k) \wedge 0 \ne k \wedge Q(y,z,k)\,\rangle$
    $\langle\, even(k) \wedge Q(y,z,k)\,\rangle$
    $\langle\, Q(y, y*z, k/2)\,\rangle$
    $k := k/2\,;\; z := z*z$
    $\langle\, Q(y,z,k)\,\rangle$ }
  $\langle\, Q(y,z,k)\,\rangle$ }
$\langle\, y = x^n \,\rangle$
```

***EXAMPLE* 5.9** Construct a proof summary for the specification

$\langle\, \top \,\rangle\, y := 1\,;\; z := 0\,;\;$ `while` $z \ne x\; \{z := z+1\,;\; y := y*z\}\,\langle\, y = x!\,\rangle$

where $k! = 1*2*3*\cdots*k$, the 'k factorial', with $0! = 1$.

Solution The proof summary follows:

```
$\langle\, \top \,\rangle$
$\langle\, 1 = 0!\,\rangle\, y := 1\,;\; \langle\, y := 0!\,\rangle\, z := 0\,;$
$\langle\,$Invariant: $y = z!\,\rangle$
while $z \ne x$ {
  $\langle\, y = z! \wedge z \ne x\,\rangle$
  $\langle\, y*(z+1) = (z+1)!\,\rangle\, z := z+1\,;\; \langle\, y*z = z!\,\rangle\, y := y*z\,\langle\, y = z!\,\rangle$ }
$\langle\, y = z! \wedge z = x\,\rangle$
$\langle\, y = x!\,\rangle$
```

***Exercise* 5.7** Annotate the proof summaries in Examples 5.7-5.9 with the names of the rules where they have been applied.

5.6 Total Correctness

In Section 5.5, you have learnt how to develop a proof of partial correctness of specifications for programs in CL. The partial correctness is conditional, the condition being the implicit assumption that the (execution of the) program actually terminates. It may happen otherwise, and then the partial correctness follows trivially, thus its proof becomes useless. Total correctness requires partial correctness and a proof that the program actually terminates. Formally, given a specification $\langle Q \rangle\, S\, \langle R \rangle$, we say that it is **totally correct**, written as $\models_t \langle Q \rangle\, S\, \langle R \rangle$, iff for any state s of the program variables of S, if s satisfies the precondition Q, then S terminates the resulting state $\bar{s}$ satisfies the postcondition R.

But what are the cases that a program may not terminate? Surely, not for those programs which do not have loops. Only in the presence of a while statement, a program in CL may not terminate. For a proof of total correctness, our Hoare logic rules remain the same except possibly for the while rule. Thus, we must focus our attention on the while statement in order to have an appropriate modification of it.

The proof of termination of a while statement usually has the following form: We identify an integer expression, obviously related to the program variables whose value decreases as the body of the while statement is repeatedly executed. Further, this expression must have a lower bound, typically 0, so that it cannot be decremented arbitrary number of times. Such an expression is called a **variant** of the loop. Once the variant achieves its lower bound, the loop terminates. For example, consider the loop in Example 5.9. The while statement starts with the value bound to the variable x, and z being bound to 0. When the loop body is executed once, the value of z is incremented to 1. The guard of the loop holds for all values of z from 0 to 'the value of x minus 1'. Once z becomes bound to the value of x, the loop terminates. We see that the value of the expression $x - z$ is decremented by 1 each time the loop is executed 'once more'. Therefore, the variant of the loop there is $x - z$.

Suppose that we have identified a variant E for the while statement `while` $B\{S\}$. This means that the value of the expression E decreases with each (repeated) execution of S. That is, if the value of E is E_0 before the loop is executed, then after the execution, the value of E is strictly less than E_0. Moreover, to code the fact that E has a lower bound, we will put a restriction on the value of E such as $0 \leq E_0$. Now, we have to incorporate a variant with these properties into the while rule. Recollect that the while rule for partial correctness was

$$\frac{\models_p \langle Q \wedge B \rangle\, S\, \langle Q \rangle}{\models_p \langle Q \rangle\, \texttt{while}\, B\{S\}\, \langle Q \wedge \neg B \rangle}$$

where Q was the invariant of the while statement. Now, since the value of the variant is always non-negative, we have an additional condition that $0 \le E$. Also, the variant E satisfies $E = E_0$ before the execution, and after the execution of S, it is decremented, i.e., the property $E < E_0$ is satisfied. Thus the required additional condition is the correctness of the specification

$$\langle\, Q \wedge B \wedge 0 \le E = E_0 \,\rangle\, S \,\langle\, Q \wedge - \le E < E_0 \,\rangle\,.$$

Moreover, the new condition $0 \le E$ may be added to the precondition of the while statement. Though this will also hold after the execution terminates, it may not be required to add it to the postcondition. It is because the condition $0 \le E$ is not a part of the *goal* of the while statement. The goal, i.e., the postcondition is usually fixed even before the program is written, whereas we only invented the condition $0 \le E$ for the proof of correctness. Hence, the **rule of total while**, written as a fraction, is

$$\textbf{(TW)}\qquad \frac{\models_t \langle\, Q \wedge B \wedge 0 \le E = E_0 \,\rangle\, S \,\langle\, Q \wedge 0 \le E < E_0 \,\rangle}{\models_t \langle\, Q \wedge 0 \le E \,\rangle \ \texttt{while}\, B\{S\} \,\langle\, Q \wedge \neg B \,\rangle}$$

Note that in all the other rules, for total correctness, we have to replace the symbol $\models_p$ by $\models_t$. However, we will refer to them with the same names, even after this replacement.

In a proof summary, along with the invariant, we will document the variant E also. The proof summary fragment for total correctness of a while statement will look like

$\langle$ Invariant: $Q \wedge$ Variant: $0 \le E\,\rangle$

$\texttt{while}\, B\,\{\ \langle\, Q \wedge B \wedge 0 \le E = E_0 \,\rangle\, S \,\langle\, Q \wedge 0 \le E < E_0 \,\rangle\ \}$

$\langle\, Q \wedge \neg B \,\rangle$

It is written in such a way that by omitting the strings 'Invariant:' and 'Variant:', you would get the clear-cut application of the total-while rule. Let us redo Example 5.9 for proving its total correctness.

***EXAMPLE* 5.10** Construct a proof summary to show that

$$\models_t \langle\, x \ge 0 \,\rangle\, y := 1\,;\ z := 0\,;\ \texttt{while}\, z \ne x\{z := z+1\,;\ y := y * z\}\,\langle\, y = x!\,\rangle$$

Solution We know already that the variant $E = z - x$. Now, compare the following proof summary for total correctness with the one for partial correctness given in Example 5.9.

$\langle\, x \ge 0 \,\rangle$

$\langle\, 1 = 0! \wedge 0 \le x - 0 \,\rangle\, y := 1\,;\ \langle\, y = 0! \wedge 0 \le x - 0 \,\rangle\, z := 0\,;$

$\langle$ Invariant: $y = z! \wedge$ Variant: $0 \le x - z\,\rangle$

$\texttt{while}\, x \ne z\,\{$

  $\langle\, y = z! \wedge x \ne z \wedge 0 \le x - z = E_0 \,\rangle$

  $\langle\, y * (z+1) = (z+1)! \wedge 0 \le x - (z+1) < E_0 \,\rangle\, z := z+1\,;$

$$\langle\, y * z = z! \land 0 \le x - z < E_0 \,\rangle\; y := y * z\,;$$
$$\langle\, y = z! \land 0 \le x - z < E_0 \,\rangle\,\}$$
$$\langle\, y = z! \land x = z \,\rangle$$
$$\langle\, y = x! \,\rangle$$

***Exercise* 5.8** Show that the precondition $x \ge 0$ cannot be replaced in the above proof summary by $\top$ if total correctness is required. Construct total correctness proofs for the specifications in Examples 5.7-5.9.

Note that the choice of a variant depends upon the nature of the statements in the while loop. Discovering an appropriate variant requires ingenuity just like the case of an invariant. As E. W. Dijkstra pointed out, "understanding a while loop is tantamount to discovering its variants and invariants". We will have some more illustrations with a hope that this experience will help you to discover the variants and invariants appropriately.

***EXAMPLE* 5.11** [*Binary Search*] Let a denote an array of n integers in which the elements are already ordered, say, in ascending order. The problem is to split the array into two parts such that all the elements in the first part and none of the numbers in the second part precede a given integer m.

Solution Any portion of the array can be represented by an ordered pair of integers, denoting the indices of the first and last numbers minus 1. For example, the array a which is written as $a[0], a[1], \cdots, a[n-1]$ is represented as the ordered pair $(0, n-1)$. Similarly, the portion $a[i], \cdots, a[j-1]$ is represented as the ordered pair $(i, j-1)$. Note that $0 \le i, j \le n-1$ and if $i \ge j$, then the portion of the array is empty. Our procedure is as follows.

Throughout our operation on the array, we will maintain three decks (imagine the array elements written on cards). The left deck contains array elements that are known to (at a certain instant during execution) precede m, in an ordered fashion. The right deck contains array elements which are all known 'not to precede' m, again in an ordered fashion, in ascending order, of course. The middle deck contains array elements, in ascending order, which are yet unknown whether to precede or not to precede m. Thus, initially, both the left and the right decks are empty, and the middle deck contains the full array. In our representation, we need just two integers to represent the three decks. Call the two integers l and r.

The representation of the three decks is as follows: The left deck is the array segment $a[0], \cdots, a[l-1]$, the middle deck is the array segment $a[l], \cdots, a[r-1]$, and the right deck is the array segment $a[r], \cdots, a[n-1]$. Initially, $l = 0$ and $r = n$. This suggests the initial assignments

$$l := 0\,;\; r := n$$

During execution, we may need to move the first i elements from the middle deck to the left; this will be effected by the assignment

$l := i + 1$

Similarly, when we require to move all array elements $a[i], \cdots, a[n]$ to the right deck, we have to use the assignment

$r := i$

Then comes the question, as to wherefrom we make the split. Assume that after some steps, we have the left, middle and the right decks, which may contain some array elements. The middle deck now has the array segment $a[l], \cdots, a[r-1]$. We want to split the middle deck into two parts by finding the middle element in this segment, say, it is the i-th. Now, if $a[i] < m$, then all of $a[l], \cdots, a[i]$ will be moved to the left deck. And if $a[i] \geq m$, then all of $a[i], \cdots, a[r-1]$ will be moved to the right deck. Since we plan to choose this i as the middle element of the middle deck containing $a[l], \cdots, a[r-1]$, i would be equal to $(l+r-1)/2$ or may be $(l+r-1)/2+1$. We choose the former. That is, we have the assignment

$i := (l + r - 1)/2$

Then, with our representation of array segments as ordered pairs, the program of **Binary Search** looks like

```
$l := 0\,;\ r := n\,;$
while $l < r$ {
  $i := (l + r - 1)/2\,;$
  if $a[i] < m$ then {$l := i + 1$} else {$r := i$}}
```

What is the precondition and what is the postcondition of this program? Since we do not want an empty array, we may have the condition $n > 0$. The array is given to be in ascending order, which means that if $j \leq k$ then $a[j] \leq a[k]$. This constitutes the precondition

$Q \equiv \forall j \forall k(0 \leq j \leq k < n \to a[j] \leq a[k])$

Similarly, after the execution of the program, our requirement was that the left deck would contain all array elements that precede m and the remaining part of the array would contain all array elements which do not precede m. Also, both l and r must be within the range 0 to n. Thus the required postcondition is

$$R \equiv 0 \leq l \leq n \land \forall j(0 \leq j < l \to a[j] < m) \land \forall j(l \leq j < n \to a[j] \geq m)$$

Note that we do not need both l and r to formalize our requirement. You can try to write the postcondition using only r, and then go for a proof of total correctness of the above program.

For the partial correctness of Binary Search, we must find an invariant for the while loop. The invariant, in general, relates the program variables. It looks as though it would be enough to have such a relation between the variables l and r only, since the decks are completely determined by the ordered pair (l, r). The important property of l is that all the array elements $a[0], \cdots, a[l-1]$ precede m. Similarly, the property of r is that none of the array elements $a[r], \cdots, a[n-1]$ precedes m. Moreover, $l \le r$. Thus, we try an invariant in the form

$$\begin{aligned} I(l,r) \equiv 0 \le l \le r \le n \wedge \forall j(0 \le j < l \to a[j] < m) \\ \wedge \forall j(r \le j < n \to a[j] \ge m) \end{aligned}$$

For total correctness, we also need a variant, whose value may decrease as the body of the loop is executed 'once more'. In the beginning, the middle deck is the full array and after the execution, we expect it to become empty. Since the number of array elements of the middle deck is $r - l$, this is a natural variant. With Q as the precondition, R as the postcondition, $I(l, r)$ as the invariant of the while loop, and $r - l$ as the variant of the loop, we have the following proof summary for total correctness:

```
$\langle Q \rangle$
$l := 0\,;\ r := n\,;$
$\langle$ Invariant: $I(l,r) \wedge$ Variant: $u - l \le E_0 \rangle$
while $l < r$ {
  $\langle I(l,r) \wedge l < u \rangle$
  $i := (l + r - 1)/2\,;$
  $\langle I(l,r) \wedge l \le i < u \rangle$
  if $a[i] < m$ then {
    $\langle I(l,r) \wedge l \le i < r \wedge a[i] < m \rangle$
    $l := i + 1 \ \langle I(l,r) \rangle$ }
   else { $\langle I(l,r) \wedge l \le i < u \wedge a[i] \ge m \rangle$ $r := i$} $\langle I(l,r) \rangle$ }
$\langle I(l,r) \wedge l \ge u \rangle$
$\langle R \rangle$
```

***Exercise* 5.9** Develop a complete Hoare proof from the above summary for the binary search.

5.7 The Predicate Transformer *wp*

The Hoare logics are not the only way a program may be verified for total correctness. In this section we will describe another way to go for correctness proofs; historically this came prior to Hoare logics. The idea is that programming is a goal-oriented activity. That is, given inputs, we want to develop a program which would give us the required output. However, when we prove a program correct, we go backward. We ask: if we require certain output, with the given program, what should have been the input?

We have seen how the assignment axiom works well, when seen this way. Now we can extend the approach to encompass all types of statements. Think of RI, the rule of implication. Suppose that we want to prove the correctness of the specification $\langle Q \rangle\, S \,\langle R \rangle$. RI states that if we have a formula P such that both $Q \models P$ and $\models_t \langle P \rangle\, S \,\langle R \rangle$ hold, then $\models_t \langle Q \rangle\, S \,\langle R \rangle$ holds. Looking at computing a required P for given S and R poses the problem of finding the weakest P. That means, if we have a program S and a required postcondition R, the formula P must be such that the specification $\langle P \rangle\, S \,\langle R \rangle$ must be totally correct. Moreover, if Q is any formula such that $\langle Q \rangle\, S \,\langle R \rangle$ is correct, then Q must entail P. That is, P is the weakest of all preconditions of the program S with respect to the postcondition R. Let us use the notation $wp\,(S, R)$ for the weakest precondition of a program S with respect to the postcondition R.

***EXAMPLE* 5.12** What is $wp\,(t := x, t = 2)$?

Solution The statement $t := x$ assigns the value of x to t. It is required that after this execution, the formula $t = 2$ must be satisfied. This can only happen provided x has the value 2 before execution. Hence

$$wp\,(t := x, t = 2) \equiv (x = 2).$$

Here is a formal definition of the weakest precondition.

Definition 5.1 Let S be a program (a statement) and R be any FL-formula. Then the **weakest precondition** of S with respect to R, denoted by $wp\,(S, R)$, is an FL-formula which describes the set of all initial states such that the execution of S in any one (or more) of the states is guaranteed to terminate in a state satisfying R.

In this language of 'a formula describing a set of states', the formula $\top$ describes the set of all states, as any state satisfies $\top$. Similarly, $\bot$ describes the empty set of states, or no states, as no state satisfies the formula $\bot$.

***Exercise* 5.10** Prove: if $Q \equiv R$, then $wp\,(S, Q) \equiv wp\,(S, R)$.

From the definition of the weakest precondition, it is obvious that wp satisfies two properties. First, $wp\,(S, Q)$ is such a precondition that the specification $\langle\, wp\,(S, R) \,\rangle\, S \,\langle R \rangle$ is totally correct. Second, if $\langle Q \rangle\, S \,\langle R \rangle$ is totally correct, then Q must entail $wp\,(S, R)$. That is,

$$\models_t \langle\, wp\,(S, R) \,\rangle\, S \,\langle R \rangle \tag{1}$$

$$\text{If } \models_t \langle Q \rangle\, S \,\langle R \rangle, \text{ then } Q \models wp\,(S, R). \tag{2}$$

These two properties are, in fact, the defining properties of wp; together they define what a wp is. Further, from the assignment rule (see also Example 5.12), it is obvious that

$$wp\,(x := E, R) \equiv R[x/E] \tag{3}$$

for any expression E, matching types with x. The rule of implication takes the form

If $wp\,(S, R) \equiv Q$ and $P \models Q$, then $\models_t \langle\, P\,\rangle\, S\, \langle\, R\,\rangle$ (4)

Similarly, the rule of sequential execution appears as

$$wp\,(S1\,;\, S2, Q) \equiv wp\,(S1, wp\,(S2, Q)) \tag{5}$$

Imagine pushing the precondition up through the program for obtaining the required weakest precondition.

***Exercise* 5.11** Show the following equivalences:
$wp\,(S1\,;\, S2\,;\, S3, Q) \equiv wp\,(S1\,;\, S2, wp\,(S3, Q)) \equiv wp\,(S1, wp\,(S2\,;\, S3, Q))$.

For the conditional statement `if` B `then` $\{S1\}$ `else` $\{S2\}$, we can identify two possibilities: (a) executing $S1$, and (b) executing $S2$. The requirements for (a) is that B must be satisfied. Now, if the postcondition of the execution of S is R, in this case, it is also the postcondition of $S1$, and then $wp\,(S1, R)$ must have been satisfied before the execution. But we know that B has been satisfied, i.e., B must guarantee the termination of $S1$ in a state that satisfies R. Referring to (4), we see that the requirement is $B \models wp\,(S1, R)$.

Similarly, if $S2$ has been executed, then under the satisfiability of $\neg B$, $wp\,(S2, R)$ is satisfied so that the postcondition R is guaranteed to be satisfied after the execution. That is, we must have $\neg B \models wp\,(S2, R)$.

These two observations together give us (see also RC)

$$\begin{aligned} &wp\,(\,\texttt{if}\ B\ \texttt{then}\ \{S1\}\ \texttt{else}\ \{S2\}, R) \\ &\quad \equiv\ (B \to wp\,(S1, R)) \wedge (\neg B \to wp\,(S2, R)) \end{aligned} \tag{6}$$

Since $(X \to Y) \wedge (\neg X \to Z) \equiv (X \wedge Y) \vee (\neg X \wedge Z)$ (Show it.), the equivalence in (6) can be rewritten as

$$\begin{aligned} &wp\,(\,\texttt{if}\ B\ \texttt{then}\ \{S1\}\ \texttt{else}\ \{S2\}, R) \\ &\quad \equiv\ (B \wedge wp\,(S1, R)) \vee (\neg B \wedge wp\,(S2, R)) \end{aligned} \tag{7}$$

Alternatively, if you accept the Hoare logic rule for the conditional, RC, for total correctness, by using the properties of wp, you can derive (6) and then (7). See the following theorem in that light.

Theorem 5.1 *The formula* $P \equiv (B \to wp\,(S1, R)) \wedge (\neg B \to wp\,(S2, R))$ *satisfies the following properties:*

(a) $\models_t \langle\, P\,\rangle$ `if` B `then` $\{S1\}$ `else` $\{S2\}\,\langle\, R\,\rangle$

(b) if $\models_t \langle\, Q\,\rangle$ `if` B `then` $\{S1\}$ `else` $\{S2\}\,\langle\, R\,\rangle$*, then* $Q \models P$

Proof Let $P, Q, S1, S2, R$ be as given in the above statement. Then,

$$P \wedge B \models wp\,(S1, R)$$

(Prove it!) By the definition of wp, we have $\models_t \langle\, P \wedge B\,\rangle\, S1\,\langle\, R\,\rangle$. Similarly, $P \wedge \neg B \models wp\,(S2, R)$. (Prove it!) Now the rule RC proves (a).

For (b), suppose that $\models_t \langle\, Q\,\rangle$ `if` B `then` $\{S1\}$ `else` $\{S2\}\,\langle\, R\,\rangle$. Then $\models_t \langle\, Q \wedge B\,\rangle\, S1\,\langle\, R\,\rangle$, and also $\models_t \langle\, Q \wedge \neg B\,\rangle\, S2\,\langle\, R\,\rangle$, as both $S1, S2$ always terminate. By the definition of wp, we obtain

$$Q \wedge B \models wp\,(S1, Q) \text{ and } Q \wedge \neg B \models wp\,(S2, Q).$$

But these are respectively equivalent to

$$Q \models (B \rightarrow wp\,(S1, Q)) \text{ and } Q \models (\neg B \rightarrow wp\,(S2, Q)).$$

Therefore, $Q \models (B \rightarrow wp\,(S1, Q)) \wedge (\neg B \rightarrow wp\,(S2, Q))$. ∎

Since properties (a) and (b) in Theorem 5.1 are the defining properties of wp, this can be used as an alternative to deriving (6).

***Exercise* 5.12** We had given an argument to show (6) and then used an equivalence for deriving (7). Give a direct argument to show (7).

The following theorem states some logical properties of the predicate transformer wp.

Theorem 5.2 *The predicate transformer wp has the following properties:*

(a) Excluded Miracle: $wp\,(S, \bot) \equiv \bot$.

(b) Termination: $wp\,(S, \top) \equiv$ *'S terminates'.*

(c) $\wedge$*-distributivity:* $wp\,(S, Q \wedge R) \equiv wp\,(S, Q) \wedge wp\,(S, R)$.

(d) $\vee$*-distributivity:* $wp\,(S, Q \wedge R) \equiv wp\,(S, Q) \vee wp\,(S, R)$, *provided that S is a deterministic program.*

(e) $\neg$*-distributivity:* $wp\,(S, \neg Q) \models \neg wp\,(S, Q)$.

(f) $\rightarrow$*-distributivity:* $wp\,(S, Q \rightarrow R) \models wp\,(S, Q) \rightarrow wp\,(S, R)$.

(g) $\models$*-distributivity:* *If* $Q \models R$, *then* $wp\,(S, Q) \models wp\,(S, R)$.

Proof (a) The postcondition $\bot$ describes no states. Thus the precondition $wp\,(S, \bot)$ describes the states which guarantee, after S terminates, no states. If s is a state that satisfies $wp\,(S, \bot)$, then after termination of S, the resulting state is a 'no state'. This is impossible. Hence there cannot be any such state s satisfying $wp\,(S, \bot)$, which means that $wp\,(S, \bot) \equiv \bot$. This is called the law of the excluded miracle, since it would be a miracle if there is a state which would be satisfied before S is executed, and after the execution, S would terminate in no states.

(b) $wp\,(s, \top)$ is a formula that describes all states s such that after termination of S, the resulting state, say $\bar{s}$, satisfies $\top$. All states $\bar{s}$ vacuously satisfy $\top$, provided S terminates. Hence $wp\,(S, \top)$ is simply a formula that guarantees termination of S. Note that $wp\,(S, \top)$ need not be equivalent to $\top$; a counter example would be a program S which does not terminate.

(c) $wp\,(S, Q \wedge R)$ describes the set of all states s that guarantee the termination of S in a state satisfying both Q and R. Any such state s guarantees termination of S in a state satisfying Q. Hence, $wp\,(S, Q \wedge R) \models wp\,(S, Q)$. Similarly, $wp\,(S, Q \wedge R) \models wp\,(S, R)$. Together they give

$$wp\,(S, Q \wedge R) \models wp\,(S, Q) \wedge wp\,(S, R).$$

Conversely, let s be a state satisfying $wp\,(S, Q) \wedge wp\,(S, R)$. Then s is a state that guarantees termination of S resulting in a state $\bar{s}$ that satisfies Q, and also R. That is,

$$wp\,(S, Q) \wedge wp\,(S, R) \models wp\,(S, Q \wedge R).$$

(d) Let s be a state that satisfies at least one of $wp\,(S, Q)$ or $wp\,(S, R)$. Then, after S terminates, the resulting state satisfies at least one of Q or R. That is, $wp\,(S, Q) \vee wp\,(S, R) \models wp\,(S, Q \vee R)$.

Conversely, suppose that S is a **deterministic program**. This means that if s is a state in which execution of S is initiated, then, upon termination of S, the resulting state is s' is unique. In contrast, one execution of a non-deterministic program can lead to one state, and another execution may drive the same initial state to another state. Now, suppose that s is a state that satisfies $wp\,(S, Q \vee R)$ before S is executed. After S terminates, let s' be the resulting state. Then, s' satisfies $Q \vee R$, i.e., s' satisfies Q or s' satisfies R. If s' satisfies Q, then by the definition of wp, s satisfies $wp\,(S, Q)$. On the other hand, if s' satisfies R, then s must satisfy $wp\,(S, R)$. Hence s satisfies $wp\,(S, Q) \vee wp\,(S, R)$. This shows that $wp\,(S, Q \vee R) \models wp\,(S, Q) \vee wp\,(S, R)$.

(e) $wp\,(S, \neg Q) \wedge wp\,(S, Q) \equiv wp\,(S, \neg Q \wedge Q) \equiv wp\,(S, \bot) \equiv \bot$, using (a) and (c). Thus, $wp\,(S, \neg Q) \wedge wp\,(S, Q) \models \bot$. By RAA and double negation, we have $wp\,(S, \neg Q) \models \neg wp\,(S, Q)$.

(f) $$\begin{aligned} & wp\,(S, Q \to R) \to (wp\,(S, Q) \to wp\,(S, R)) \\ \equiv\ & (wp\,(S, Q \to R) \wedge wp\,(S, Q)) \to wp\,(S, R) \\ \equiv\ & wp\,(S, (Q \to R) \wedge Q) \to wp\,(S, R), \text{ by (c)} \\ \equiv\ & wp\,(S, Q \wedge R) \to wp\,(S, R), \text{ as } (Q \to R) \wedge Q \equiv Q \wedge R \\ \equiv\ & wp\,(S, Q) \wedge wp\,(S, R) \to wp\,(S, R), \text{ by (c) again} \\ \equiv\ & \top \end{aligned}$$

Thus, $wp\,(S, Q \to R) \models wp\,(S, Q) \to wp\,(S, R)$.

(g) Let $Q \models R$. Then $Q \wedge R \equiv Q$. And, $wp\,(S, Q \wedge R) \equiv wp\,(S, Q)$ or with (c), $wp\,(S, Q) \wedge wp\,(S, R) \equiv wp\,(S, Q)$. Thus, $wp\,(S, Q) \models wp\,(S, R)$. ∎

Note that the three laws mentioned in (e)-(g) in Theorem 5.2 are one sided distributivity laws.

***EXAMPLE* 5.13** The converse of the 'one sided' laws in Theorem 5.2 are not valid for arbitrary programs and arbitrary postconditions Q. For this, show that

(a) $\neg wp\,(S, Q) \not\models wp\,(S, \neg Q)$

(b) $wp\,(S, Q) \to wp\,(S, R) \not\models wp\,(S, Q \to R)$

(c) $wp\,(S, Q) \models wp\,(S, R)$ does not imply $Q \models R$.

Solution (a) Let S be a program that does not terminate. We see that

$$wp\,(S,\neg Q) \equiv \neg wp\,(S,\top) \equiv \neg\top \equiv \bot, \text{ and}$$

$$wp\,(S,\neg Q) \equiv wp\,(S,\top) \equiv \text{ `}S\text{ terminates'}.$$

Now, if $\neg wp\,(S,Q) \models wp\,(S,\neg Q)$, then we would have $\top \models$ 'S terminates', contradicting the assumption that S does not terminate.

(b) We see that

$$\begin{aligned} & wp\,(S,Q) \rightarrow wp\,(S,R) \\ &\quad \equiv\ wp\,(S,\bot) \rightarrow wp\,(S,\top) \\ &\quad \equiv\ \bot \rightarrow \text{ `}S\text{ terminates'} \\ &\quad \equiv\ \top \end{aligned}$$

Also, $wp\,(S,Q \rightarrow R) \equiv wp\,(S,\bot \rightarrow \top) \equiv wp\,(S,\top) \equiv$ 'S terminates'.

Now, if $wp\,(S,Q) \rightarrow wp\,(S,R) \models wp\,(S,Q \rightarrow R)$, then as in (a), we would have 'S always terminates', which does not hold for a non-terminating program.

(c) Let S be a non-terminating program and $Q \equiv \top, R \equiv \bot$. Now,

$$wp\,(S,\top) \equiv \text{ `}S\text{ terminates'}.$$

Since S does not terminate, 'S terminates' $\models \bot$. But,

$$wp\,(S,R) \equiv wp\,(S,\bot) \equiv \bot.$$

Hence, 'S terminates' $\models wp\,(S,R)$. That is, the consequence

$$wp\,(S,Q) \models wp\,(S,R)$$

is valid. But, $Q \models R$ does not hold since $\top \not\models \bot$.

***Exercise* 5.13** Following the lines of proof in Theorem 5.2 (e-g), try proofs for their converse statements. See where the proofs break down.

The one left out is the while statement. We must see how wp works on that. Let W be the while statement `while` $B\{S\}$. Let R be a postcondition for W. We want to compute $wp\,(W,R)$. We, of course, do not know it beforehand, nor do we know whether W terminates. We want to guarantee termination of W, i.e., the loop is executed a finite number of times. Now, if the body S of the loop is never executed, then, before the execution of W is initiated, we have a state s that satisfies $\neg B$. After this execution of W (by skipping its body), the postcondition R must be satisfied by the same state s. Thus the required precondition is $\neg B \wedge R$.

Denote by P_m the weakest precondition of W with respect to the postcondition R, where the body S of W is executed exactly m times. Then, the above discussion boils down to the equivalence

$$P_0 \equiv \neg B \wedge R.$$

What happens if S is executed exactly once? In that case, S is executed once and then the guard B is not satisfied, and the body S is skipped. That is, the postcondition for the first execution is P_0. Moreover, B had been satisfied for initiating the first execution. Thus, the required weakest precondition is

$$P_1 \equiv B \wedge wp\,(S, P_0)$$

In general, you can think of the repeated execution of S, as a sequence of statements, and then push the postcondition up to the precondition where the 0-th execution of S is done. Suppose that the k-th execution of S is over, and then the $(k+1)$-th execution is initiated. Then B must have been satisfied by any resulting state of the k-th execution. Further, such a state must also have satisfied the weakest precondition for the k-th execution. Hence,

$$P_{k+1} \equiv B \wedge wp\,(S, P_k)$$

Since we require W to terminate, the repeated execution must stop somewhere. That is, there must exist a natural number, say, n such that the body S of W is executed exactly n times. In our notation, it is simple to express since this means that for some k, P_k holds. This yields, for $m \geq 0$,

$$P_0 \equiv \neg B \wedge R,\ P_{m+1} \equiv B \wedge wp\,(S, P_m),$$
$$wp\,(\,\texttt{while}\, B\{S\}, R) \equiv \exists k (k \in \mathbb{N} \wedge P_k) \tag{8}$$

Though this is enough for capturing the weakest precondition, we must also look at the invariants of a while loop. Recollect that I is an invariant of W means that if S is executed with the precondition I and S terminates, then I is also a guaranteed postcondition. Now, with I as a postcondition, we have $wp\,(S, I)$ as the weakest precondition. Thus, a state that satisfies I, B, and 'termination of S', must also satisfy $wp\,(S, I)$. This means that an invariant I of a while statement: `while` $B\{S\}$, satisfies the property

$$I \wedge B \wedge wp\,(S, \top) \models wp\,(S, I) \tag{9}$$

***Exercise* 5.14** Prove the *Fundamental Invariance Theorem*: Let I be an invariant of `while` $B\{S\}$. Then

$$I \wedge wp\,(\,\texttt{while}\, B\{S\}, \top) \models wp\,(\,\texttt{while}\, B\{S\}, I \wedge \neg B).$$

[*Hint*: Define predicates $P_0(Q) \equiv Q \wedge \neg B, P_k(Q) \equiv B \wedge wp\,(S, P_{k-1}(Q))$. Prove, by induction, that $I \wedge P_k(\top) \equiv P_k(I \wedge \neg B)$.]

***Exercise* 5.15** Relate the result in Exercise 5.14 to the Hoare logic rule RW. This will tell you why the result is fundamental.

We use the properties (1)-(9) of the predicate transformer wp for proving total correctness of programs in CL.

***EXAMPLE* 5.14** Use wp to show that (compare with Example 5.5)

$$\models_t \langle\, 1 + a + a^2 = 0 \wedge a^n = b + c * a \,\rangle$$
$$n := n + 1\,;\ m := b + c\,;\ b := c\,;\ c := m\,\langle\, a^n = b + c * a \,\rangle$$

Solution We simply compute wp from the postcondition and then see that the precondition actually entails the wp. Then,

$$
\begin{aligned}
& wp\,(n := n+1\,;\; m := b+c\,;\; b := c\,;\; c := m, a^n = b + c * a) \\
&\quad \equiv\; wp\,(n := n{+}1, wp\,(m := b{+}c, wp\,(b := c, wp\,(c := m, a^n{+}b{+}c{*}a)))), \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{using (5)} \\
&\quad \equiv\; wp\,(n := n+1, wp\,(m := b+c, wp\,(b := c, a^n = b + m * a))), \text{ by (3)} \\
&\quad \equiv\; wp\,(n := n+1, wp\,(m := b+c, a^n = c + m * a)) \\
&\quad \equiv\; wp\,(n := n+1, a^n = c + (b+c) * a) \\
&\quad \equiv\; a^{n+1} = c + (b+c) * a
\end{aligned}
$$

As in Example 5.5, $1 + a - a^2 = 0 \wedge a^n = b + c * a \models a^{n+1} = c + (b + c) * a$. By (4), we get the required total correctness.

***EXAMPLE* 5.15** Show by computing the weakest precondition that

$$
\begin{aligned}
&\models_t \langle\, m = i * j + k + 1 \,\rangle \\
&\qquad \texttt{if}\; j := k+1 \;\texttt{then}\; \{i := i+1\,;\; k := 0\} \;\texttt{else}\; \{k := k+1\} \\
&\qquad \langle\, m = i * j + k \,\rangle
\end{aligned}
$$

Solution Denote the required wp by Q. Then, Q is equivalent to

$$
\begin{aligned}
& wp\,(\;\texttt{if}\; j := k+1 \;\texttt{then}\; \{i := i+1\,;\; k := 0\} \;\texttt{else}\; \{k := k+1\}, \\
&\qquad m = i * j + k) \\
&\quad \equiv\; ((j = k+1) \to wp\,(i := i+1\,;\; k := 0, m = i * j + k) \\
&\qquad \wedge ((j \neq k+1) \to wp\,(k := k+1, m = i * j + k))
\end{aligned}
$$

But,

$$
\begin{aligned}
& wp\,(i := i+1\,;\; k := 0, m = i * j + k) \\
&\quad \equiv\; wp\,(i := i+1, wp\,(k := 0, m = i * j + k)) \\
&\quad \equiv\; wp\,(i := i+1, m = i * j) \\
&\quad \equiv\; m = (i+1) * j
\end{aligned}
$$

And,

$$
wp\,(k := k+1, m = i * j + k) \equiv m = i * j + k + 1
$$

Hence,

$$
\begin{aligned}
Q \;&\equiv\; ((j = k{+}1) \to (m = (i{+}1){*}j)) \wedge ((j \neq k{+}1) \to (m = i{*}j{+}k{+}1)) \\
&\equiv\; ((j = k+1) \wedge (m = (i+1) * j)) \vee ((j \neq k+1) \wedge (m = i * j + k + 1)) \\
&\equiv\; ((j = k+1) \wedge (m = i * j + j)) \vee ((j \neq k+1) \wedge (m = i * j + k + 1)) \\
&\equiv\; ((j = k{+}1) \wedge (m = i{*}j{+}k{+}1)) \vee ((j \neq k{+}1) \wedge (m = i{*}j{+}k{+}1)) \\
&\equiv\; ((j = k+1) \vee (j \neq k+1)) \wedge (m = i * j + k + 1) \\
&\equiv\; \top \wedge (m = i * j + k + 1) \equiv m = i * j + k + 1
\end{aligned}
$$

The total correctness now follows from (4).

***EXAMPLE* 5.16** By computing wp show that

$$\begin{aligned}&\models_t \langle\, \exists k(k \in \mathbb{N} \land i = n - 2 * k \land s = 0)\\ &\qquad \lor \exists k(k \in \mathbb{N} \land i = n - 2 * k - 1 \land s = k)\,\rangle\\ &\qquad \texttt{while}\, i \neq n\{k := -k\,;\, s := s + k\,;\, i := i + 1\}\,\langle\, s = 0\,\rangle\,.\end{aligned}$$

Solution Here the guard is $B \equiv i \neq n$; the postcondition is $R \equiv s = 0$, and the body of the while statement S is $k := -k\,;\, s := s + k\,;\, i := i + 1$. Now,

$$\begin{aligned}&P_0 \equiv \neg B \land R \equiv i = n \land s = 0\\ &P_1 \equiv B \land wp\,(S, P_0) \equiv i \neq n \land wp\,(S, i = n \land s = 0)\\ &\quad\equiv i \neq n \land wp\,(S, i = n) \land wp\,(S, s = 0)\end{aligned}$$

But,

$$\begin{aligned}&wp\,(S, i = n)\\ &\quad\equiv\ wp\,(k := -k, wp\,(s := s + k' wp\,(i := i + 1, i = n)))\\ &\quad\equiv\ wp\,(k := -k, wp\,(s := s + k, i + 1 = n))\\ &\quad\equiv\ wp\,(k := -k, i + 1 = n) \equiv i + 1 = n\end{aligned}$$

And,

$$\begin{aligned}&wp\,(S, s = 0) \equiv wp\,(k := -k, wp\,(s := s + k, wp\,(i := i + 1, s = 0)))\\ &\quad\equiv\ wp\,(k := -k, wp\,(s := s + k, s = 0))\\ &\quad\equiv\ wp\,(k := -k, s + k = 0) \equiv s - k = 0\end{aligned}$$

Therefore,

$$\begin{aligned}&P_1 \equiv i \neq n \land i + 1 = n \land s - k = 0 \equiv i + 1 = n \land s - k = 0,\\ &P_2 \equiv i \neq n \land wp\,(S, i + 1 = n) \land wp\,(S, s - k = 0) \equiv i + 2 = n \land s = 0.\end{aligned}$$

If you look at the job the loop does, the above formulas suggest that

$$P_{2k} \equiv i = n - 2 * k \land s = 0,\ P_{2k+1} \equiv i = n - 2 * k - 1 \land s = k \tag{10}$$

Now you see that the responsibility of discovering an invariant for the Hoare logic proof of a while statement takes the form of discovering a formula for P_k. Moreover, you will have to use induction to show that your conjecture on the form of P_k is, indeed, correct. This involves ingenuity, whereas computation of wp for other statements is quite mechanical. Once you have discovered P_k, your job is almost over. With P_k as mentioned,

$$\begin{aligned}&wp\,(\,\texttt{while}\, i \neq n\{k := -k\,;\, s := s + k\,;\, i := i + 1\}, s = 0)\\ &\quad\equiv\ \exists k(k \in \mathbb{N} \land i = n - 2 * k \land s = 0)\\ &\qquad\ \lor \exists k(k \in \mathbb{N} \land i = n - 2 * k - 1 \land s = k)\end{aligned}$$

***Exercise* 5.16** Prove that P_k in Example 5.16 satisfies (10).

***Exercise* 5.17** Since a while statement W : $\texttt{while}\, B\{S\}$ terminates in a state satisfying $\neg B$, show that $wp\,(W, \top) \equiv wp\,(W, \neg B)$.

***EXAMPLE* 5.17** Compute $wp\,(\texttt{while}\, n \neq m\{S\}R)$, where

S is $i := i + 2\,;\ s := s + n * i + k\,;\ k := k + i\,;\ n := n + 1$, and

$R \equiv s = m^3 \wedge i = 2 * m \wedge k = m * (m + 1) + 1.$

Solution In the solution below, many steps are omitted, but you should be able to fill in those so that you construct a full computation of wp.

$$\begin{aligned} P_0 &\equiv n = m \wedge s = m^3 \wedge i = 2 * m \wedge k = m * (m + 1) + 1 \\ &\equiv n = m \wedge s = n^3 \wedge i = 2 * n \wedge k = n * (n + 1) + 1 \end{aligned}$$

And

$$\begin{aligned} & wp\,(S, s = n^3 \wedge i = 2 * n \wedge k = n * (n + 1) + 1) \\ &\equiv wp\,(i := i + 2\,;\ s := s + n * i + k\,;\ k := k + i, s = (n + 1)^3 \\ &\qquad \wedge i = 2 * (n + 1) \wedge k = (n + 1)8(n + 2) + 1) \\ &\equiv s = n^3 \wedge i + 2 = 2 * (n + 1) \wedge k = n * (n + 1) + 1 \\ &\equiv s = n^3 \wedge i = 2 * n \wedge k = n * (n + 1) + 1 \end{aligned}$$

Can you see that it is an invariant of the while statement? Now,

$$\begin{aligned} P_1 &\equiv n \neq m \wedge wp\,(S, P_0) \\ &\equiv n \neq m \wedge n + 1 = m \wedge s = n^3 \wedge i = 2 * n \wedge k = n * (n + 1) + 1 \end{aligned}$$

By induction, we obtain

$$\begin{aligned} P_k &\equiv n \neq m \wedge n + k = m \wedge s = n^3 \wedge i = 2 * n \wedge k = n * (n + 1) + 1 \\ &\equiv n = m - k \wedge s = n^3 \wedge i = 2 * n \wedge k = n * (n + 1) + 1 \end{aligned}$$

Thus,

$$wp\,(W, R) \equiv n \leq m \wedge s = n^3 \wedge i = 2 * n \wedge k = n * (n + 1) + 1.$$

Now that you know what program verification is, you can think of some generalities. You are presented with an informal description D of a problem in an application domain. Your aim is to develop a program to solve the problem, and then verify that the program, indeed, does the job. As a first step, you represent D into a formula X_D in some logic. We have chosen here the first order logic FL. For some problem domains, FL may not be that appropriate and you may have to use other logics for problem representation. (You will learn some more logics in Chapter 6.)

The next step is to write a program P which would realize the formula X_D, i.e., it should meet the specifications as declared by X_D. This is the phase we have not discussed at all. We have only given some hints as to how to program in the core language CL. 'How to program' is an art, and it must be mastered thoroughly. It would be nice if certain principles are followed, which we have not discussed. You will find some pointers mentioned in the summary to this chapter. Moreover, programs in CL may not be acceptable to your customers. So, you must be able to carry out similar 'principle following' activity in any language, which might be offered in your company environment, or which might be required by your customer.

The last step is to *prove* that the program P meets the requirements; it satisfies the specification X_D. Ideally, total correctness of the specification must be shown. This last issue has only been dealt with in this chapter, an that too, only partly. You must see how the art of proving programs correct helps you to specify and develop programs. This clearly involves many rounds of discussion with your customer, as the informal description D might have inaccuracy, which you cannot certainly include in your formal description X_D.

A word of caution: our definitions of $\models_p$ and $\models_t$ are semantic, in the sense that for a specification Γ we write $\models_p \Gamma$ or $\models_t \Gamma$, according as something happens for the initial state and the final state. However, the proof rules of Hoare logic or of *wp* are syntactic. They are syntactic, in the sense that once you accept to follow the rules, you have no need to go back to the states for constructing a proof of correctness of Γ. A proof then shows that the specification Γ is provable (with partial or total correctness). So, we must have encoded these provabilities by some different symbols, say, $\vdash_p$ or $\vdash_t$ accordingly. Then, it is our responsibility to show adequacy, i.e., we must show that the metastatements '$\vdash_p$ iff $\models_p$' and '$\vdash_t$ iff $\models_t$' hold. This has not been attempted here. Try it!

SUMMARY

In this chapter you have learnt to argue about correctness of programs. Programs are written to get some job done. In conventional programming languages such as Basic, Fortran Pascal, or C, you just write how to get the job done. These programs contain only one part of the whole work of program development; it is the 'how' part. They are silent about what is required to be done. Neither they say anything about why it is correct to do the job that way. In this chapter we have tried to use logic for developing a program including all the three aspects of what, how and why of a program.

For an easy presentation, we have defined the core language CL. It has all the basic features of a programming language such as assignment statement, sequential execution, conditional statement, and the while statement. Since our aim has been to integrate all the three aspects of what, how and why, we have introduced the notion of a program specification. A specification addresses the what of a program by way of writing the 'what is required of a program' in terms of a precondition and a postcondition. The program itself is written in between them and it codes 'how the job is done'.

In order to show that the program, indeed, does the required job, the specification is to be proved correct. You have learnt how to prove partial correctness of a program by developing the Hoare logic for CL. In partial correctness, the crucial assumption was that the program always terminates. You have also learnt how to extend the Hoare logic to include a proof of termination. This resulted in the rules for total correctness. The proof summary so developed combines all the three aspects of a program, where you do not need extra documentation. Finally, you have learnt how to use the weakest precondition of a program with respect to a given post-condition in proving the total correctness of programs.

This chapter is only a short note on program verification focussing rather on a useful application of logic in computer science. The following bibliographic hints will be helpful for pursuing your interest in this area.

The presentation of the topics in this chapter is largely influenced by the works of C. A. R. Hoare [38], E. W. Dijkstra [10, 11], D. Gries [33, 34], and R. C. Backhouse [3]. The motivating example of the string matching problem is taken from [3]. For more details on non-deterministic executions, see [10]. The ideas developed here can be used for program constructions also; the details can be found in [3, 33]. You will find plenty of educative and entertaining examples and exercises in these texts. Other recommended texts are [2, 24, 37, 42, 60], where you would find a relatively complete exposition of program verification including extended features such as writing to arrays, array cell aliasing, procedure calls and parallelism. For a systematic extension of the core language to include other advanced features and then their verification, consult [64, 76]. For verification of functional programming languages, you may start with [78]. For the freely available functional programming language ML, [54] is a good text. You can also explore the web resources for newer presentations of the subject. The journal *Science of Computer Programming* is a good source for interesting new problems and their algorithmic solutions. But, before you go on a hunt for material, attempt the following problems.

PROBLEMS

1. When does $\texttt{if}\ C\ \texttt{then}\ \{S1\}\ \texttt{else}\ \{S2\}$ fail to terminate?

2. In many languages a for-loop is used instead of a while loop. For example, to sum the elements of an array a, whose elements are denoted by $a[i]$, and having 100 elements, you may write the following program:

$$s := 0\,;\ \texttt{for}(i = 0, i \le 100, i := i + 1)\{s := s + a[i]\}$$

The program first initializes s to 0, then starts the for-loop. In executing the for-loop, it first initializes i to 0, then executes its body $s := s + a[i]$, and then increments i to $i + 1$. It continues doing this repeatedly till the guard $i \le 100$ holds, and stops doing it when i becomes greater than 100. How do you write a program in CL to implement the for-loop? Try for the more general for-loop: $\texttt{for}(S1, S2, S3)\{S4\}$.

3. A repeat until loop looks like: `repeat`$\{S1\}$`until`$S2$. Execution of such a loop means:

(a) $S1$ is executed in the current state,

(b) $S2$ is evaluated in the resulting state,

(c) if $S2$ is false, the program resumes with (a) and continues, otherwise, the program terminates.

Define this loop in CL. Can you define this loop through for-loop?

4. Determine the preconditions Q and the postconditions R such that the following hold:

(a) $\models_p \langle\, Q \,\rangle$ `while` $\top\{i := 0\}\,\langle\, R \,\rangle$

(b) $\models_t \langle\, Q \,\rangle$ `while` $\top\{i := 0\}\,\langle\, R \,\rangle$

5. Check whether the following specifications are partially and/or totally correct. What are the programs supposed to compute?

(a) $\langle\, \top \,\rangle\, x := m + 1\,;$ `if` $x = 1$ `then` $\{n := 1\}$ `else` $\{n := x\}$
$\langle\, n := m + 1 \,\rangle$

(b) $\langle\, m \geq 0 \,\rangle\, n := 1\,;\, k := 0\,;$ `while` $k \neq m\{k := k + 1\,;\, n := n * k\}$
$\langle\, n := m! \,\rangle$

(c) $\langle\, \top \,\rangle\, n := 1\,;\, k := 0\,;$ `while` $k \neq m\{k := k{+}1\,;\, n := n{*}k\}\,\langle\, n := m! \,\rangle$

(d) $\langle\, m \geq 0 \,\rangle\, n := 1\,;$ `while` $m \neq 0\{n := n{*}m\,;\, m := m{-}1\}\,\langle\, n := m! \,\rangle$

(e) $\langle\, m = m_0 \wedge m \geq 0 \,\rangle\, n := 1\,;$
`while` $m \neq 0\{n := n * m\,;\, m := m - 1\}\,\langle\, n := m_0! \,\rangle$

(f) $\langle\, m = m_0 \wedge m \geq 0 \,\rangle\, k := 0\,;$
`while` $m > 0\{k := k + m\,;\, m := m - 1\}\,\langle\, k = m * (m + 1)/2 \,\rangle$

6. Which of the following are assignment axioms?

(a) $\models_p \langle\, 5 = 5 \,\rangle\, m := 5\,\langle\, m = 5 \,\rangle$

(b) $\models_t \langle\, 5 = 6 \,\rangle\, m := 5\,\langle\, m = 6 \,\rangle$

(c) $\models_p \langle\, n = 5 \,\rangle\, m := 5\,\langle\, m = n \,\rangle$

(d) $\models_t \langle\, 5 > 3 \,\rangle\, m := 5\,\langle\, m > 3 \,\rangle$

(e) $\models_t \langle\, m + 1 = n \,\rangle\, m := m + 1\,\langle\, m = n \,\rangle$

(f) $\models_p \langle\, m + 1 > 0 \wedge n > 0 \,\rangle\, m := m + 1\,\langle\, m > 0 \wedge n > 0 \,\rangle$

7. Develop proofs summaries and then the complete Hoare proofs of partial and total correctness (if possible) for the following specifications:

(a) $\langle\, m > 0 \,\rangle\, n := m + 1\,\langle\, n > 1 \,\rangle$

(b) $\langle\, \top \,\rangle\, n := m\,;\, n := 2 * m + n\,\langle\, n = 3 * x \,\rangle$

(c) $\langle\, \top \,\rangle$ `if` $n < m$ `then` $\{k := n\}$ `else` $\{k := m\}\,\langle\, k = \min{(m, n)} \,\rangle$

(d) $\langle\, m \geq 0 \,\rangle\, x := m\,;\, n := 0\,;$

$\texttt{while}\, x \neq 0\{n := n + 1\,;\, x := x - 1\}\,\langle\, m := n \,\rangle$

(e) $\langle\, \top \,\rangle\, x := m\,;\, n := 0\,;$

$\texttt{while}\, x \neq 0\{n := n + 1\,;\, x := x - 1\}\,\langle\, m := n \,\rangle$

(f) $\langle\, n \geq 0 \,\rangle\, x := 0\,;\, k := 0\,;$

$\texttt{while}\, x \neq n\{k := m + k\,;\, x := x + 1\}\,\langle\, k := m * n \,\rangle$

(g) $\langle\, n = n_0 \wedge n \geq 0 \,\rangle\, k := 0\,;$

$\texttt{while}\, n \neq 0\{k := m + k\,;\, n := n - 1\}\,\langle\, k := m * n_0 \,\rangle$

(h) $\langle\, m \geq 0 \,\rangle\, n := 0\,;\; \texttt{while}\, n \neq m\{n := n + 1\}\,\langle\, m = n \,\rangle$

(i) $\langle\, n \neq 0 \,\rangle\, r := m\,;\, d := 0\,;$

$\texttt{while}\, r \geq n\{r := r - n\,;\, d := d + 1\}\,\langle\, (m := d * n + r) \wedge (r < n) \,\rangle$

(j) $\langle\, m \geq 0 \,\rangle\, x := m\,;\, n := 1\,;$

$\texttt{while}\, x \geq 0\{n := x * n\,;\, x := x - 1\}\,\langle\, n = m! \,\rangle$

8. Write programs P so that the following specifications are totally correct; also prove the correctness.

(a) $\langle\, \top \,\rangle\, P\,\langle\, n = m + 6 \,\rangle$

(b) $\langle\, \top \,\rangle\, P\,\langle\, k < m + n + 0.5 * k \,\rangle$

(c) $\langle\, \top \,\rangle\, P\,\langle\, k = \max(l, m, n) \,\rangle$

(d) $\langle\, \top \,\rangle\, P\,\langle\, ((x = 4) \rightarrow (y = 3)) \wedge ((x = 7) \rightarrow (y = 5)) \,\rangle$

9. *The minimal sum section* [41]: Let $a[1], \ldots, a[n]$ be the elements of an integer array a. A section of a is a continuous piece $a[i], a[i+1], \ldots, a[j]$, where $1 \leq i \leq j \leq n$. A minimal sum section is a section $a[i], \ldots, a[j]$ such that the sum $S_{ij} = a[i] + a[i+1] + \cdots + a[j]$ is minimal over all sections. Note that a is not necessarily ordered. To write a program which computes the minimal sum section of a given array, we store two values: the minimal sum seen so far (s) and the minimal sum seen so far of *all* the sections which end at the current element of the array (t). We also assume that we know how to compute $\min(x, y)$. Prove the partial correctness of the following program that does the job with precondition $\top$ and postcondition $\forall i \forall j (i \leq j \leq n \rightarrow s \leq Sij)$:

$k := 2\,;\, t := a[1]\,;\, s := a[1]\,;$

$\texttt{while}\, k \neq n + 1\{t := \min(t + a[k], a[k])\,;\, s := \min(s, t)\,;\, k := k + 1\}$

[*Hint*: Use the invariant: $\forall i \forall j (i \leq j < k \rightarrow s \leq S_{ij}) \wedge \forall i (i < k \rightarrow t \leq S_{ij})$ and develop a proof summary.]

10. Give Hoare proofs and also *wp*-proofs of correctness for the following specifications, prefixing to them one of $\models_p$ or $\models_t$ as appropriate:

(a) $\langle\, m = n^k \,\rangle\, k := k+1\,;\; m := m * n\, \langle\, m = n^k \,\rangle$

(b) $\langle\, n = m^2 \wedge s = m * (m+1) * (2*m+1)/6 \,\rangle$

$m := m+1\,;\; n = n + 2*m - 1\,;\; s := n + s$

$\langle\, n = m^2 \wedge s = m * (m+1) * (2*m+1)/6 \,\rangle$

(c) $\langle\, j = m^n \wedge s = (m^{n+1} - 1)/(m-1) \,\rangle$

$j := j * m\,;\; s := s + j\,;\; n := n+1$

$\langle\, j = m^n \wedge s = (m^{n+1} - 1)/(m-1) \,\rangle$

(d) $\langle\, s = n^3 \wedge i = 2*n \wedge k = n*(n+1)+1 \,\rangle$

$i := i+2\,;\; s := s + n*i + k\,;\; k := k + i\,;\; n := n+1$

$\langle\, s = n^3 \wedge i = 2*n \wedge k = n*(n+1)+1 \,\rangle$

(e) $\langle\, k^2 = k+1 \wedge k^m = a*k+b \,\rangle$

$m := m+1\,;\; t := a+b\,;\; b := a\,;\; a := t$

$\langle\, k^m = a*k+b \,\rangle$

(f) $\langle\, 0 \le s < n \,\rangle$

$q := s/(n-1)\,;\; p := q+1\,;\; t := s + 1 - q*(n-1)$

$\langle\, 1 \le t \le n \wedge q \ge 0 \wedge p = q+1 \wedge s = p*(t-1) + q*(n-t) \,\rangle$

(g) Let $swap(x, y)$ be a procedure which interchanges the values of x and y. Then $wp\,(swap(x,y), R) \equiv R[x/y, y/x]$. Recall that the substitution $[x/y, y/x]$ is not equal to $[x/y][y/x]$. Develop a Hoare proof for

$\langle\, ((y \ge z) \to (y \ge x)) \wedge ((y < z) \to (z \ge x)) \,\rangle$

`if` $y \ge z$ `then` $\{swap(x,y)\}$ `else` $\{swap(x,z)\}$

$\langle\, (x \ge y) \wedge (x \ge z) \,\rangle$

(h) $\langle\, \top \,\rangle$ `if` $i < j$ `then` $\{swap(i,j)\}$ `else` $\{i := i\}\,;$

`if` $j < k$ `then` $\{swap(j,k)\}$ `else` $\{j := j\}\,;$

`if` $i < j$ `then` $\{swap(i,j)\}$ `else` $\{i := i\}$ $\langle\, i \ge j \ge k \,\rangle$

(i) With P as $(odd(p) \to (x = y = 1)) \wedge (\neg odd(p) \to (x = 0 \vee y = 0))$,

$\langle\, p = m*n \,\rangle$ `if` $odd(m)$ `then` $\{x := 1\}$ `else` $\{x := 0\}\,;$

`if` $odd(n)$ `then` $\{y := 1\}$ `else` $\{y := 0\}$ $\langle\, P \,\rangle$

(j) With P as in (i),

$\langle\, P \,\rangle$ `if` $odd(p)$ `then` $\{z := 1\}$ `else` $\{z := 0\}$ $\langle\, z = x*y \,\rangle$

(k) Use (i) and (j) to show that

$\langle\, p = m*n \,\rangle$ `if` $odd(m)$ `then` $\{x := 1\}$ `else` $\{x := 0\}\,;$

`if` $odd(n)$ `then` $\{y := 1\}$ `else` $\{y := 0\}\,;$

`if` $odd(p)$ `then` $\{z := 1\}$ `else` $\{z := 0\}$ $\langle\, z = x*y \,\rangle$

(l) $\langle\, \exists k(k \geq 0 \wedge i = n-2*k \wedge s = 0) \vee \exists k(k \geq 0 \wedge i = n-2*k-1 \wedge s = k)\,\rangle$
$\texttt{while}\, i \neq n\{k := -k\,;\ s := s + k\,;\ i := i + 1\}\,\langle\, s = 0\,\rangle$

(m) $\langle\, c = x * y\,\rangle\ \texttt{while}\, \neg odd(x)\{y := 2 * y\,;\ x := x/2\}\,\langle\, c = x * y\,\rangle$

(n) $\langle\, x \geq 0 \wedge c = x * y\,\rangle\ \texttt{while}\, x \neq 0\,\{$
$\texttt{while}\, \neg odd(x)\{y := 2 * y\,;\ x := x/2\}\,;\ c := c - y\,;\ x := x - 1\}$
$\langle\, c = x * y\,\rangle$

11. Let S be the statement $\texttt{if}\, B\, \texttt{then}\, \{S1\}\, \texttt{else}\, \{S2\}$. Suppose that $P \wedge B \wedge wp\,(S1, \top) \models wp\,(S1, Q)$ and $P \wedge \neg B \wedge wp\,(S2, \top) \models wp\,(S2, Q)$. Show that $P \wedge wp\,(S, \top) \models wp\,(S, Q)$.

12. Construct a proof summary along with the necessary verification conditions for the following specification:

$\langle\, 0 \leq n\,\rangle\, k := n\,;\ y := 1\,;\ z := x\,;$
$\langle$ Invariant: $I(y, z, k) \equiv y * z^k = x^n \wedge 0 \leq k\,\rangle\ \texttt{while}\, 0 \neq k\{$
$\texttt{if}\, odd(k)\, \texttt{then}\, \{k := k - 1\,;\ y := y * z\}\, \texttt{else}\, \{k := k\}$
$\langle\, even(k) \wedge I(y, z, k)\,\rangle\, k := k/2\,;\ z := z^2\ \}\,\langle\, y = x^n\,\rangle$

13. Let $a[i]$ denote the i-th element of an integer array a. Construct proof summaries for the following programs that sum up the array elements.

(a) $\langle\, 0 \leq n\,\rangle\, s := 0\,;\ m := 0\,;$
$\langle$ Invariant: $s := \sum_{i=0}^{m-1} a[i]$, Variant: $n - m\,\rangle$
$\texttt{while}\, m \neq n\{s := s + a[m]\,;\ m := m + 1\}\,\langle\, s = \sum_{i=0}^{n-1} a[i]\,\rangle$

(b) $\langle\, 0 \leq n\,\rangle\, s := 0\,;\ m := n\,;$
$\langle$ Invariant: $s = \sum_{i=m}^{n-1} a[i]$, Variant: $m\,\rangle$
$\texttt{while}\, m \neq 0\{m := m - 1\,;\ s := s + a[m]\}\,\langle\, s = \sum_{i=0}^{n-1} a[i]\,\rangle$

14. Construct a proof summary for the following program for evaluating the power x^m for given numbers x and m:

$\langle\, 0 \leq m\,\rangle\, k := m\,;\ y := 1\,;\ \langle$ Invariant: $y * x^k = x^m$, Variant: $k\,\rangle$
$\texttt{while}\, k \neq 0\{k := k - 1\,;\ y := y * x\}\,\langle\, y = x^m\,\rangle$

15. Suppose that the binary representation of a natural number m is stored in an array a. Construct a proof summary for the following algorithm that computes x^m:

$\langle\, 0 \leq m = \sum_{i=0}^{n-1} a[i] * 2^i\,\rangle$
$y := 1\,;\ z := x\,;\ j = 0\,;$
$\langle$ Invariant: $y * z^k = x^m \wedge k = \sum_{j=i}^{n-1} a[i] * 2^i$, Variant: $n - j\,\rangle$
$\texttt{while}\, j \neq n\{\ \texttt{if}\, a[j] = 1\, \texttt{then}\, \{y := y * z\}\, \texttt{else}\, \{y := y\}$
$j := j + 1\,;\ z = z * z\}\,\langle\, y = x^m\,\rangle$

16. Construct a proof summary for the following algorithm that evaluates a polynomial with integer coefficients which are stored in an array a:

```
$\langle\, 0 \le n \,\rangle\; s := 0\,;\; k := n\,;$
$\langle\,$Invariant: $s * x^k = \sum_{i=k}^{n-1} a[i] * x^i$, Variant: $k\,\rangle$
while $k \ne 0\{k := k - 1\,;\; s + s * x + a[k]\}\;\langle\, s = \sum_{i=0}^{n-1} a[i] * x^i \,\rangle$
```

17. Construct a proof summary for the following specification written for computing the remainder of dividing p by q in binary arithmetic:

```
$\langle\, 0 \le p \wedge 0 < q \,\rangle$
$r := p\,;\; m := q\,;$
$\langle\,$Invariant: $\exists i(i \ge 0 \wedge m = 2^i * q)$, Variant: $r - m\,\rangle$
while $r \ge m\{m := 2 * m\}$
$\langle\,$Invariant: $0 \le r < m \wedge \exists d(p := q * d + r)$
            $\wedge\, \exists i(i \ge 0 \wedge m = 2^i * q)$, Variant: $m\,\rangle$
while $m \ne q\{m := m/2\,;$ if $r \le m$ then $\{r := r - m\}$ else $\{r := r\}\,\}$
$\langle\, 0 \le r < q \wedge \exists d(p = q * d + r) \,\rangle$
```

18. Let a be an array and you are to search for an occurrence of an item, say, x in the array. You define a predicate is by $is(l, p) \equiv \exists i(l \le i < p \wedge a[i] = x)$. Then, you write the following algorithm to do the (linear) search:

```
$\langle\, 0 \le n \,\rangle\; a[n] := x\,;\; k := 0\,;$
$\langle\,$Invariant: $0 \le k \le n \wedge (is(0, n) \leftrightarrow is(k, n)) \wedge a[n] = x$, Variant: $n - k\,\rangle$
while $a[k] \ne x\{k := k + 1\}$
$\langle\, a[k] = x \wedge 0 \le k \le n \wedge (is(0, n) \leftrightarrow k < n) \,\rangle$
```

Explain what exactly the algorithm does and then develop a proof summary.

6

Modal Logics

6.1 Introduction

Let p and q be two propositions. Consider the following argument:

p is valid. $p \to q$ is valid. Therefore, q is valid.

Now, how do you symbolize this argument in PL? Here we have three atomic sentences:

A : p is valid.

B : $p \to q$ is valid.

C : q is valid.

Agreeing that 'therefore' means the consequence relation, the argument is symbolized as

$$\{A, B\} \models C$$

Though we know that the argument is correct, there is no way to prove it *in* PL.

What about FL? Here, we can go a bit deeper into analyzing the argument. Suppose that we denote 'is valid' as a unary predicate $P(\cdot)$. Then the argument would be symbolized as

$$\{P(p), P(p \to q)\} \models P(q)$$

But it is wrong. Why? Because $P(p \to q)$ is not a formula; the symbol $\to$ cannot occur in the argument of a predicate, only a term is allowed over there. So, symbolization fails. But suppose we write '$-$ is valid' as an operator, just like the unary connective $\neg$, and try to construct a new logic, then? Well, let us write 'x is valid' as $\Box x$ (read as box x). Then, the symbolization would lead to the consequence:

$$\{\Box p, \Box(p \to q)\} \models \Box q$$

It looks feasible, but we have to do so many things afresh, if we accept this symbolization. We have to allow such constructions syntactically and then give meaning to these constructs. Let us see one more example:

Anyone whose parents are dead is an orphan. Yanka believes that his parents are dead. Therefore, Yanka believes that he is an orphan.

The argument seems to be correct, but how do you symbolize it in FL? With $B(x, z)$ for 'x believes z' and a as the constant 'Yanka', you would get something like

$$\{\forall x(D(x) \to O(x)), B(a, p)\} \models B(a, o)$$

OR

$$\{\forall x(D(x) \to O(x)), B(a, D(a))\} \models B(a, O(a))$$

***Exercise* 6.1** What are the constants p, o and what do the predicates D and O stand for?

In the former case, we cannot show the argument to be valid, and the latter case is syntactically wrong as the predicates D and O cannot occur as arguments of another predicate B. Further, the symbols D and O cannot be taken as functions since $D(x) \to O(x)$ would then be syntactically wrong.

When an old theory is inadequate, we create new theories. Let us try writing 'Yanka believes' as an operator, say, $\Box$. Then our symbolization would look like

$$\{\forall x(Dx \to Ox), \Box Dy\} \models \Box\, Oy$$

To take some more examples, consider the following:

Yanka is bold.

Yanka is necessarily bold.

Yanka is possibly bold.

Yanka is made to be bold.

Yanka must not be bold.

Yanka ought to be bold.

Yanka is supposed to be bold.

Yanka is known to be bold.

Yanka is believed to be bold.

Yanka appears to be bold.

Yanka is allowed to be bold.

Yanka is not free not to be bold.

Except the first sentence all the others are modal sentences. They simply modify the first sentence in some way or the other. This is why the name 'modal logics', and we will see later why this name is in plural. Out of all the above modalities (and possibly many more not written above), we take up two catch words 'necessarily' and 'possibly' as our basics. Other modalities will be treated as different readings of these basic ones. Further, the different interpretations and readings of these modalities may give rise to altogether different logics. As you progress through this chapter, you will understand how this is done.

6.2 Syntax and Semantics of K

The logic K is an extension of PL in the sense that it has two more connectives $\Box$ (read as box) for *necessity* and $\Diamond$ (read as diamond) for *possibility.* The language of K has the alphabet consisting of all propositional variables, $p_0, p_1, \ldots$, also called, **atomic modal propositions**, or **atomic modal formulas**, unary connectives $\neg, \Box, \Diamond$, binary connectives $\wedge, \vee, \rightarrow, \leftrightarrow$, the propositional constants $\top, \bot$, and the punctuation marks), (. As in PL, you may choose a shorter list of connectives and propositional constants or a longer list including all the binary truth functions such as $\uparrow, \downarrow$, etc. Henceforth, for convenience, we regard the propositional constants $\top$ and $\bot$ as atomic modal propositions, and work with the usual connectives. **Modal propositions** or, **mp** for short, are defined recursively by the following formulation rules:

1. Each atomic modal proposition is a modal proposition.
2. If p is a modal proposition, then $\neg p, \Box p, \Diamond p$ are modal propositions.
3. If p, q are modal propositions, then $(p \wedge q), (p \vee q), (p \rightarrow q), (p \leftrightarrow q)$ are modal propositions.
4. Each modal proposition is generated by applying one or more of the formation rules 1 to 3 above.

Using the symbol $p \in \{p_i : i \in \mathbb{N}\}$, the grammar of modal propositions in the Bacus−Naur form can be given as

$$X ::= \top \,|\, \bot \,|\, p \,|\, \neg X \,|\, \Box X \,|\, \Diamond X \,|\, (X \wedge X) \,|\, (X \vee X) \,|\, (X \rightarrow X) \,|\, (X \leftrightarrow X)$$

The parse tree of an mp (a modal proposition) is a binary tree with branch points (non-leaf nodes) as the connectives and leaves as atomic mps. It is straightforward to see that the unique parsing holds in K.

***EXAMPLE* 6.1** The parse tree of $\Diamond(\Box(p \leftrightarrow \neg p) \wedge (\Diamond(q \rightarrow p) \vee \neg q))$ is given in Figure 6.1.

Theorem 6.1 (Unique Parsing in K) *Any modal proposition has a unique parse tree. That is, if X is a modal proposition, then X is equal to exactly one among $\top, \bot, p_i, \neg Y, \Box Y, \Diamond Y, (Y \wedge Z), (Y \vee Z), (Y \rightarrow Z)$, and $(Y \leftrightarrow Z)$, where Y and Z are unique modal propositions.*

***Exercise* 6.2** Prove Theorem 6.1. The proof follows the same line of argument as in the case of PL.

The expression $\Diamond(\Box(p \leftrightarrow \neg p)\Box(\Diamond(q \rightarrow p) \vee \neg q))$ is not an mp, because $\Box$ is a unary connective. $\Box(p \leftrightarrow \neg p) \wedge (\Diamond(q \rightarrow p) \vee \neg q)$ is not an mp, since outer brackets are missing. However, we now make it a convention to omit the outer parentheses. We also follow the precedence rules of PL, giving the unary connectives $\neg, \Box, \Diamond$ the same and the highest precedence. Next in the hierarchy are $\wedge$ and $\vee$. The connectives $\rightarrow, \leftrightarrow$ receive the least precedence.

Sometimes we override the precedence rules and use extra parentheses for easy reading. Further, we will not be rigid about writing the atomic mps as $p_0, p_1, \ldots$; we will rather use any other symbol, say $p, q, \ldots$, but mention that it is an atomic mp. You can define sub-mps just like sub-propositions in PL or subformulas in FL. They are well defined due to unique parsing. Once the syntax is clear, we must discuss the semantics.

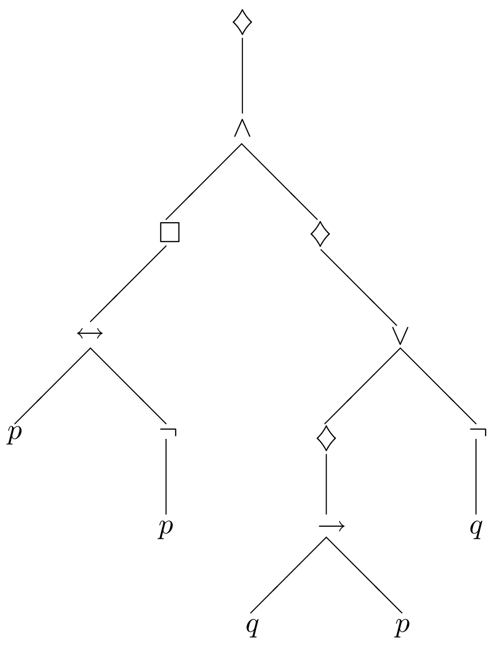


Figure 6.1 A parse tree.

The connectives $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$ have the same meanings in K, as in PL, though the meanings of $\Box$ and $\Diamond$ complicate their use a bit. Notice the various interpretations of the symbols $\Box$ and $\Diamond$ that we have mentioned in Section 6.1. Let us start with the reading of the symbols in the metalogic of PL. That is, we want to read $\Box$ as 'it is valid that' and $\Diamond$ as 'it is satisfiable that'. A proposition p in PL is valid iff it is evaluated to 1 (true) under each interpretation. Looking at the truth values of the atomic propositions occurring in p, we know how to check the truth of p. In the language of modal logic, we would call each of these interpretations a **world** (sometimes, a point, a situation, a state, etc.). Now, as we agreed, 'p is valid' is symbolized as $\Box p$, and 'p is valid' holds, or that $\Box p$ is true if p is true under every world. In this reading of $\Box$ as validity, we take a world as an interpretation, a function that takes each (atomic) proposition to the set $\{0, 1\}$. While considering other modalities, e.g., read $\Box$ as "Yanka believes that", we may have to define a world differently. A world then would not be an interpretation. In abstract terms, a world would be taken as any object, a primary object. Note that this abstraction does not prevent us in taking a world as an interpretation; it allows us to consider other structures besides interpretations. In giving meanings to modal propositions, we would start with a set of worlds.

In this particular reading of $\Box$ as validity, it is now clear that $\Box p$ holds, when p is true in every world. What about the iterated modalities, say, the truth of $\Box\Box p$? This is certainly an mp, but the reading of $\Box$ as validity would not yield to any good sense. Though we can still make out the meaning of $\Box\Box p$, the truth of $\Box\Box p$ would mean that $\Box p$ is true in every world, and then finally, it would mean that "it is true in every world that p is true in every world". Since a world here is taken as a PL-interpretation, we lose nothing by asserting that p is true in every world. Suppose that we read $\Box$ differently, say, as "Yanka knows that", then $\Box\Box p$ may be read more meaningfully as "Yanka knows that he knows p". Clearly, this sentence is different from "Yanka knows p", which is the translation of $\Box p$. So? The point is that in some readings of the modal connectives the iterated modalities may not yield anything new, while other readings may still distinguish between the iterated modalities. And then, we must not only say how to define '$\Box p$ is true' but also see how '$\Box p$ is true in a world'.

We answer this problem with a relation defined over the set of worlds. This relation is, again, a primitive concept; it should *come* along with the set of worlds. That is, we require not only a set but a relational structure for giving meanings to the modal connectives. This relation is called an accessibility relation, an agreed phrase taken for convenience. With this relation on the set of worlds, the worlds themselves become accessible from each other. The accessible relation on a particular set of worlds need not be symmetric; so we must distinguish between *accessible from* and *accessible to*. Now $\Box p$ would be true in a world w if p is true in every world accessible to w. In the case of $\Box$ as validity, each world (an interpretation) is accessible to each other world, and also to itself. Thus, $\Box\Box p$ will be forcefully read as $\Box p$ hinting at self assertion of utterances in natural languages. (Do you see the hint?) If we do not agree that $\Box\Box p$ be read as $\Box p$, then, of course, the accessibility relation must be different. As a starting point for the semantics of K, we define a relational structure of worlds and the accessibility relation among them.

Definition 6.1 A **frame** is an ordered pair (W, R), where W is a set, called the set of worlds, and R is a binary relation on W (a subset of $W \times W$), called the **accessibility relation**. For worlds $u, w \in W$, read wRu as u is accessible from w.

A frame is not sufficient to give meanings (of truth and falsity) to all modal propositions. We must specify when an mp is true at a world. Thus each world w may be considered as a mapping from the set of atomic propositions to the set $\{0, 1\}$ of truth values. Then we would define p **is true at the world** w iff $w(p) = 1$. We also invent a shorthand. We write $w \Vdash p$ whenever p is true at w. Recollect that we had alternate semantics for

PL, where an interpretation was taken as a set of literals, or of propositional variables. Similarly, we may regard each world w as a subset of the set of all atomic propositions which happen to be true at w. That is, we can associate the world w with the mapping $f_w : \{p_i : i \in \mathbb{N}\} \to \{0, 1\}$, or with the set $S_w = \{p_i : i \in \mathbb{N} \text{ and } f_w(p_i) = 1\}$. This association can be brought in by another mapping ϕ, which would take each world to f_w or to S_w, depending upon which way we want to go about. Nonetheless, they are equivalent. In the former case, we have $\phi(w) = f_w$, where $\phi : W \to$ the set of all functions from $\{p_i : i \in \mathbb{N}\}$ to $\{0, 1\}$. And in the latter case, we have $\phi(w) = S_w$, where $\phi : W \to$ the power set of $\{p_i : i \in \mathbb{N}\}$. Alternatively, we can associate with each p_i, the set of worlds where p_i is true. One more alternative: ϕ can be defined as a map from $W \times \{p_i : i \in \mathbb{N}\}$ to $\{0, 1\}$ so that $\phi(w, p_i) = 1$ means that p_i is true in the world w. All these approaches are, any way, equivalent. Why? We remember these alternatives and choose the second one for defining a model.

A model here, unlike PL, is simply an interpretation where a modal proposition may or may not hold. It would have been better to call these as interpretations rather than as models, or it could have been better to define a model in PL this way. But current practice uses the word 'interpretation' for PL, and a 'model' for modal logics. A 'model' in PL is taken as a model of a proposition which is an interpretation where the proposition is true. Note that in this terminology, a model in modal logics is defined for a logic (as the concept of an interpretation is defined for PL), and we cannot say 'model of an mp'. We should constantly remember this abuse of language. (Which one is an abuse, PL-models or modal models?)

Definition 6.2 A **model** of K is a triple (W, R, ϕ), where the pair (W, R) is a frame, and ϕ is a mapping from W to the power set of $\{p_i : i \in \mathbb{N}\}$ associating each world $w \in W$ to a subset $\phi(w)$ of atomic propositions. The mapping ϕ is called the **world truth mapping**. The model $M = (W, R, \phi)$ is said to be **based upon the frame** $F = (W, R)$, and the frame F is said to be an **underlying frame** of the model M.

Note that the set $\phi(w)$ consists of all the atomic propositions which are true at the world w. Moreover, each world w is *propositional*, in the sense that all the laws of PL hold in each world. For example, if $p \in \phi(w)$, i.e., if p is true at the world w, then $\neg p$ cannot be true at w, i.e., $\neg p \notin \phi(w)$. The new connectives $\Box$ and $\Diamond$ play roles in navigating across the worlds via the accessibility relation. Now we must define formally how a modal proposition becomes true at (in) a world w. We use the notation $w \Vdash p$ for satisfaction at a world w, a shorthand for "the modal proposition p is true in the world w", or equivalently, for "the world w satisfies (or verifies) the modal proposition p". We also read it as "p **is true at** (in) the world w", and also as, "the world w **verifies** the modal proposition p." When a world does not satisfy an mp, we say that it **falsifies** the mp. The following is a formal definition of the relation of 'satisfaction at a world'.

Definition 6.3 Let $M = (W, R, \phi)$ be a model of K. Let x, y denote arbitrary modal propositions. Let $w \in W$ be a world. The relation $w \Vdash x$, read as w **satisfies** x, is defined recursively:

(a) $w \Vdash \top$, $w \nVdash \bot$.

(b) $w \Vdash p$ iff $p \in \phi(w)$ for any atomic proposition p.

(c) $w \Vdash \neg x$ iff $w \nVdash x$.

(d) $w \Vdash x \wedge y$ iff $w \Vdash x$ and $w \Vdash y$.

(e) $w \Vdash x \vee y$ iff $w \Vdash x$ or $w \Vdash y$.

(f) $w \Vdash x \to y$ iff $w \nVdash x$ or $w \Vdash y$.

(g) $w \Vdash x \leftrightarrow y$ iff either $w \Vdash x$ and $w \Vdash y$, or $w \nVdash x$ and $w \nVdash y$.

(h) $w \Vdash \Box x$ iff for each world $u \in W$ with wRu, $u \Vdash x$.

(i) $w \Vdash \Diamond x$ iff for some world $u \in W$ with wRu, $u \Vdash x$.

We may read the meaning of the mp $\Box x$ as "$\Box x$ is true at (in) a world w if the mp x is true in every world accessible from w." Similarly, the mp $\Diamond x$ is true in the world w if the mp x is true in some world accessible from w. Since satisfaction or true at a world w depends not only on the world w, but also on the model M, we should better write $w \Vdash x$ as $M, w \Vdash x$. However, if the model M is clear from a given context, then we would continue writing $w \Vdash x$. If we feel that confusion might arise in a certain context due to involvement of many models, we can write the full form $M, w \Vdash x$. Due to unique parsing, the satisfaction relation is well defined.

***EXAMPLE* 6.2** Let $M = (W, R, \phi)$ be a model where $W = \{w, u\}$, $R = \{(w, u), (u, u)\}$, $\phi(w) = \emptyset$, and $\phi(u) = \{p\}$, for an atomic mp p. Are the mps $p, \Box p, \Box\Box p, \Box p \to \Box\Box p, \Box\Box p \to p$ true at the world w?

Solution Look at the relation R of the model M. It says that the world w is accessible from u, the world u is accessible from itself, and that is all about this accessibility relation. Since every binary relation can be depicted as a graph, we first represent our model as a graph. each node of this graph is a world. A node corresponding to the world w will be written as $\boxed{\text{w}}$. If $(w, u) \in R$, i.e., if wRu, then we draw a directed edge (line segment) from the node $\boxed{\text{w}}$ to the node $\boxed{\text{u}}$. That is, 'u is accessible from w' is drawn as

$$\begin{tikzcd}\boxed{\text{w}} \arrow[r] & \boxed{\text{u}}\end{tikzcd}$$

The mapping ϕ which gives details about which atomic propositions (mp) are true in which world can be depicted by writing the satisfaction symbol, the double turnstile ($\Vdash$) as a superscript to (or just following) the world and then writing out all those atomic propositions which are true at that world. For example, since in the world u, the atomic proposition p is true, this fact can be depicted as $\boxed{\text{u}}^{\Vdash p}$. With this **convention for drawing a model**, the model M can be drawn as

$$\begin{tikzcd} \boxed{\text{w}} \arrow[r] & \boxed{\text{u}} \arrow[loop below] \arrow[r, phantom, "\Vdash p"] & {} \end{tikzcd}$$

Since $\phi(w) = \emptyset$, nothing is labelled with the world w. Moreover, u is accessible from itself; hence the loop. We will often specify a model by depicting its graph.

Coming back to satisfaction, $w \nVdash p$. What about $\Box p$? $w \Vdash \Box p$ iff $v \Vdash p$ for every world v accessible from w. Here, u is the only world accessible from w, and $u \Vdash p$. Hence $w \Vdash \Box p$. Does $w \Vdash \Box\Box p$ hold? This would be so provided $u \Vdash \Box p$ as u is the only world accessible from w. Again, $u \Vdash \Box p$ if $u \Vdash p$ as u is the only world accessible from u. This last satisfaction, $u \Vdash p$ holds. Thus, $w \Vdash \Box\Box p$.

By induction, it follows that $w \Vdash \Box^n p$, for every positive integer n; we abbreviate the iterated modality $\Box\Box \cdots (n\text{times})\Box p$ to $\Box^n p$, for any $n \geq 0$, with the convention that $\Box^0 p = p$. Further, since $w \nVdash p$ and $w \Vdash \Box^n p$, for any $n \geq 1$, we see that $w \nVdash \Box p \to p$. What about $p \to \Box p$? Clearly, $w \Vdash p \to \Box p$. In general, $w \Vdash \Box^m p \to \Box^n p$ for any $m, n \in \mathbb{N}$ with $n \neq 0$, and $w \Vdash p \to p$.

***EXAMPLE* 6.3** Consider the model M: $\boxed{\text{w}} \longrightarrow \boxed{\text{u}}^{\Vdash p}$. Which of the following mps are true at the world w?

(a) $\Box p \to \Box\Diamond p$ (b) $\Diamond p \to \Box\Diamond p$ (c) $\Box p \to \Diamond\Box p$ (d) $\Diamond p \to \Diamond\Box p$

Solution (a) $w \Vdash \Box p \to \Box\Diamond p$ iff either $w \nVdash \Box p$ or $w \Vdash \Box\Diamond p$ (or both). Now, $w \nVdash \Box p$ iff for some world accessible from w, p is false at that world. There is only one world, namely, u which is accessible to w and $u \Vdash p$. Hence $w \nVdash \Box p$ does not hold. For the other alternative, $w \Vdash \Box\Diamond p$ iff $u \Vdash \Diamond p$. But there is no world accessible from u. Thus, $u \nVdash \Diamond p$. Summing up, $w \nVdash \Box p \to \Box\Diamond p$.

(b) $w \Vdash \Diamond p$ as u is a world accessible from w and $u \Vdash p$. We have seen in (a) that $w \nVdash \Box\Diamond p$. Hence $w \nVdash \Diamond p \to \Box\Diamond p$.

(c) In (a), we have already seen that $w \Vdash \Box p$. What about $\Diamond\Box p$? Since u is the only world accessible from w, for $w \Vdash \Diamond\Box p$, we require $u \Vdash \Box p$. This latter satisfaction would hold provided every world accessible from u satisfies p. As there is no world which is accessible from u, the condition is satisfied vacuously. So, $w \Vdash \Diamond\Box p$; consequently, $w \Vdash \Diamond p \to \Diamond\Box p$.

***EXAMPLE* 6.4** Take the same model M of Example 6.3. Which of the mps in Example 6.3 are true at the world u?

Solution Here, $u \Vdash \Box p$, as every world accessible from u (there is none, indeed) satisfies p. Similarly, $u \Vdash \Box\Diamond p$. Again, as there is no world accessible from u, we have $u \nVdash \Diamond p$. Similarly, $u \nVdash \Diamond\Box p$. Therefore, u satisfies the mps $\Box p \to \Box\Diamond p$, $\Diamond p \to \Box\Diamond p$ and $\Diamond p \to \Diamond\Box p$ but it falsifies $\Box p \to \Diamond\Box p$.

***Exercise* 6.3** Do Examples 6.2 and 6.3 replacing $\Box$ by $\Diamond$.

In the examples above you see that both the worlds u, w satisfy the mp $\Diamond p \to \Diamond\Box p$. Moreover, these are the only worlds in the model M. In such a case, we say that the model M satisfies the mp $\Diamond p \to \Diamond\Box p$.

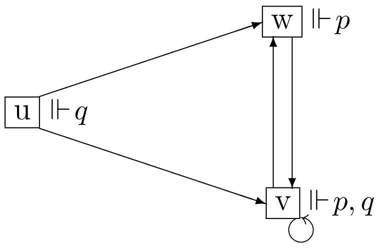
Definition 6.4 Let $M = (W, R, \phi)$ be a model of K. Let x be a modal proposition. We say that the model M **satisfies** or **verifies** x, and write it as $M \models x$, iff for every world $w \in W$, $w \Vdash x$.

You have already an example at hand, the model M of Example 6.3 satisfies the modal proposition $\Diamond p \to \Diamond\Box p$. Besides, for the same model M, you see that $M \not\models \Box p \to \Box\Diamond p$ as $w \nVdash \Box p \to \Box\Diamond p$. As $w \nVdash \Diamond p \to \Box\Diamond p$ and $u \nVdash \Box p \to \Diamond\Box p$, you see that $M \not\models \Diamond p \to \Box\Diamond p$ and $M \not\models \Box p \to \Diamond\Box p$ hold.

***EXAMPLE* 6.5** Let $M = (W, R, \phi)$, where $W = \{u, v, w\}, R = \{(u, w),$ $(u, w), (v, v), (v, w), (w, v)\}$, and $\phi(u) = \{q\}, \phi(v) = \{p, q\}, \phi(w) = \{p\}$. Which of the following hold?

(a) $M \models \Box(p \wedge q) \to (\Box p \wedge \Box q)$ (b) $M \models \Box p \wedge \Box q \to \Box(p \wedge q)$

Solution When depicted as a graph, the model looks like:



Let us make a table of satisfaction showing which mps are true at which world. We have three worlds u, v, w in our model M and mps such as $p, q, p \wedge q, \Box p, \Box q, \Box(p \wedge q), \Box p \wedge \Box q$. We know already that $u \Vdash q, v \Vdash p, v \Vdash q, w \Vdash p$ and $u \nVdash p, w \nVdash q$. Now, is it that $u \Vdash \Box p$? Well, the worlds accessible from u are v and w, and both $v \Vdash p, w \Vdash p$ hold. Hence, $u \Vdash \Box p$. But, $u \nVdash \Box q$ as the world w is accessible from u but $w \nVdash q$. What about v? All the worlds accessible from v are v and w. Both $v \Vdash p$ and $w \Vdash p$ hold. So, $v \Vdash \Box p$. However, $v \nVdash \Box q$ as w is accessible from v but $w \nVdash q$. Similarly, $w \Vdash \Box p$ and $w \Vdash \Box q$ as the only world accessible from w is v and $v \Vdash p, v \Vdash q$. Write r for $\Box(p \wedge q)$ and s for $\Box p \wedge \Box q$. $u \nVdash \Box(p \wedge q)$ as w is accessible from u but $w \nVdash p \wedge q$. $v \nVdash \Box(p \wedge q)$ as $w \nVdash p \wedge q$. $w \Vdash \Box(p \wedge q)$ as $v \Vdash p \wedge q$. Then, as in PL, we have the following table for satisfaction:

	p	q	$p \wedge q$	$\Box p$	$\Box q$	r	s	$r \to s$	$s \to r$
u	$\nVdash$	$\Vdash$	$\nVdash$	$\Vdash$	$\nVdash$	$\nVdash$	$\nVdash$	$\Vdash$	$\Vdash$
v	$\Vdash$	$\Vdash$	$\Vdash$	$\Vdash$	$\nVdash$	$\nVdash$	$\nVdash$	$\Vdash$	$\Vdash$
w	$\Vdash$	$\nVdash$	$\nVdash$	$\Vdash$	$\Vdash$	$\Vdash$	$\Vdash$	$\Vdash$	$\Vdash$

Look at the last two columns in the above table. They show that all the worlds (in M) satisfy both the mps $r \to s$ and $s \to r$, which are, respectively, $\Box(p \wedge q) \to \Box p \wedge \Box q$ and $\Box p \wedge \Box q \to \Box(p \wedge q)$. Therefore, the model M satisfies both these mps. That is,

$$M \models \Box(p \wedge q) \to \Box p \wedge \Box q, \; M \models \Box p \wedge \Box q \to \Box(p \wedge q)$$

***EXAMPLE* 6.6** For the model given below, determine whether it satisfies $\Box(p \vee q) \to \Box p \vee \Box q$ and $\Box(p \wedge q) \to \Box p \wedge \Box q$.

$$\begin{tikzcd}
 & \boxed{\text{w}} \Vdash q \\
M : \boxed{\text{u}} \arrow[ur] \arrow[dr] & \\
 & \boxed{\text{v}} \Vdash p
\end{tikzcd}$$

Solution Here, $u \Vdash \Box(p \vee q)$, as both the worlds v, w accessible from u satisfy $p \vee q$. But $u \nVdash \Box p$ as the world w accessible from u does not satisfy p. Again, $u \nVdash \Box q$ as $v \nVdash q$. Therefore, $u \nVdash \Box p \vee \Box q$ and consequently, $M \nVdash \Box(p \vee q) \to \Box p \vee \Box q$.

What about the second? $u \nVdash \Box p$, $u \nVdash \Box q$ but $v \Vdash \Box p$, $v \Vdash \Box q$ vacuously as there is no world accessible from v. You also see that $w \Vdash \Box p$ and $w \Vdash \Box q$. Thus, $u \nVdash \Box p \wedge \Box q$, $v \Vdash \Box p \wedge \Box q$, $w \Vdash \Box p \wedge \Box q$. As $v \nVdash p \wedge q$, a world accessible from u, you see that $u \nVdash \Box(p \wedge q)$. You also have $v \Vdash \Box(p \wedge q)$ and $w \Vdash \Box(p \wedge q)$. So, all the worlds satisfy $\Box(p \wedge q) \to \Box p \wedge \Box q$; consequently, $M \models \Box(p \wedge q) \to \Box p \wedge \Box q$.

***Exercise* 6.4** Let M be a model and w be a world in M, i.e., $w \in W$, where $M = (W, R, \phi)$. Show that

(a) $w \Vdash \Diamond\top$ iff there is at least one world accessible from w.

(b) $w \Vdash \Box\bot$ iff no world is accessible from w.

Conclude from (a) and (b) that $M \models \top$ but M may not satisfy $\Diamond\top$, and that each world in M falsifies $\bot$, though it may happen that $M \models \Box\bot$.

In Example 6.6, you have seen that $M \not\models \Box(p \vee q) \to \Box p \vee \Box q$. But in both the Examples 6.5 and 6.6, the respective models satisfy the mp $v(p \wedge q) \to \Box p \wedge \Box q$. Such an mp which is satisfied by every model (not just these two) is called a valid modal proposition.

Definition 6.5 Let x be a modal proposition in K. We say that x is **valid** (i.e., x is K-valid or valid in K), and write it as $\models x$ (or for precision $\models_K x$) iff for every model M, $M \models x$.

Note that all the above relations of satisfaction such as $w \Vdash x$, $M \models x$, and $\models x$ should be written as $w \Vdash_K x$, $M \models_K x$, and $\models_K x$, respectively. Unless other logics are involved in the same context, we will suppress the subscript K.

***EXAMPLE* 6.7** Show that $\models_K \Box(p \wedge q) \to \Box p \wedge \Box q$.

Solution We are not going to draw any model here since we are supposed to argue with any arbitrary model. So, let $M = (W, R, \phi)$ be any model (any K-model), and let $w \in W$. If $w \nVdash \Box(p \wedge q)$, then by the definition of satisfaction, $w \Vdash \Box(p \wedge q) \to \Box p \wedge \Box q$. Otherwise, suppose that $w \Vdash \Box(p \wedge q)$. Our aim is to show that $w \Vdash \Box p \wedge \Box q$. For this purpose, let u be a world accessible from w, i.e., $u \in W$ and uRw. Since $w \Vdash \Box(p \wedge q)$, each accessible world (from w) satisfies $p \wedge q$, i.e., $u \Vdash p \wedge q$. Then $u \Vdash p$ and $u \Vdash q$. This holds for any arbitrary world u that is accessible from w. Hence, $w \Vdash \Box p$ and $w \Vdash \Box q$. Therefore, $w \Vdash \Box p \wedge \Box q$. Again, w is any arbitrary world in W; thus we have shown that each world w (in the model M) satisfies the mp $\Box(p \wedge q) \to \Box p \wedge \Box q$. That is, $M \models \Box(p \wedge q) \to \Box p \wedge \Box q$. This holds for any arbitrary model M; consequently, $\models \Box(p \wedge q) \to \Box p \wedge \Box q$.

***Exercise* 6.5** Show that $\models_K \Box p \wedge \Box q \to \Box(p \wedge q)$. Is the modal proposition $\Box p \vee \Box q \to \Box(p \vee q)$ valid in K?

Some of the important K-valid modal propositions are now listed.

Theorem 6.2 (Laws in K) *In the modal logic K, the following laws hold:*

(a) Law of the constants: $\models \Box\top \leftrightarrow \top$, $\models \Diamond\bot \leftrightarrow \bot$

(b) Law of De Morgan: $\models \neg\Box p \leftrightarrow \Diamond\neg p$, $\models \neg\Diamond p \leftrightarrow \Box\neg p$

(c) Law of $\wedge$-distributivity: $\models \Box(p \wedge q) \leftrightarrow (\Box p \wedge \Box q)$

(d) Law of $\vee$-distributivity: $\models \Diamond(p \vee q) \leftrightarrow (\Diamond p \vee \Diamond q)$

(e) Law of $\to$-distributivity: $\Box(p \to q) \to (\Box p \to \Box q)$

***Exercise* 6.6** Prove the laws listed in Theorem 6.2.

Many K-valid modal propositions can be obtained by substituting mps in place of propositional variables in PL-valid propositions. This is so because each world in K is propositional and retains validity of PL. See the following theorem.

Theorem 6.3 (Tautological Substitution) *Let p be a propositional variable and A be a proposition (in PL). Let q be a modal proposition. Let B be a modal proposition obtained from A by substituting each occurrence of p by q. If A is valid in PL, then B is valid in K.*

Proof Let $M = (W, R, \phi)$ be a model and $w \in W$ be a world. Suppose that A is valid in PL. Note that A is also an mp. Since w is propositional, the world w is a PL-interpretation. Since A is valid, each interpretation satisfies it. In particular, $w \Vdash A$. Further, $w \Vdash A$ holds whether $p \in \phi(w)$ or not, i.e., whether $w \Vdash p$ holds or not. (Imagine $w \Vdash p$ as $w(p) = 1$ and $w \nVdash p$ as $w(p) = 0$.) When p is replaced by q in A, the satisfaction of the resulting formula then would not change whether $w \Vdash q$ holds or not. Thus, satisfaction of B would be the same as that of A, i.e., $w \Vdash B$. Since $w \in W$ is any world, $M \models B$. Since M is an arbitrary model, B is K-valid. ∎

***Exercise* 6.7** Use the tautological substitution theorem to show that the mps $\Box p \vee \neg\Box p$, $\Diamond p \vee \neg\Diamond p$, $(\Box p \to \Diamond q) \leftrightarrow (\Box p \leftrightarrow (\Box p \wedge \Diamond q))$ and $\Box p \to (\Diamond q \to \Diamond p) \to ((\Box p \to \Diamond q) \to (\Box p \to \Diamond p))$ are K-valid.

You now have a scheme to generate infinitely many K-valid mps using tautologies (of PL). The following result shows that just like PL, equivalence substitution holds in K. To formulate the result, you must, first of all, say what an equivalence in K means. We say that the modal propositions A and B are **equivalent** (in K), and write it as $A{\equiv_K}B$ iff $\models_K A \leftrightarrow B$. If there is no confusion, we will drop the subscript K and simply write the equivalence as $A \equiv B$. We also write $Z[A := B]$ for the mp obtained from Z by substituting some or all occurrences of A in Z by B.

Theorem 6.4 (Equivalence Substitution) *Let A, B, Z be modal propositions. If $A \equiv B$, then $Z[A := B] \equiv Z$.*

Proof See the corresponding fact and its proof in PL. ∎

Thus, any instance of a valid modal proposition is again a valid modal proposition. Moreover, in the calculational style, you can write $\models Z$ as $Z \equiv \top$, the way you did in PL. There are, of course, many differences between K and PL, the connection being that K is an extension of PL and each world in K is an interpretation in PL.

A model has been taken as a triple $M = (W, R, \phi)$, where ϕ prescribes which atomic mps are true at which worlds. The relation $\Vdash$ of satisfaction between worlds and arbitrary mps is an extension of this ϕ from the atomic mps to all mps. The extension must, of course, satisfy the defining properties of $\Vdash$ as given in Definition 6.3. For economy in notation, we can rewrite the model as $M = (W, R, \Vdash)$, where $\Vdash$ is given as a function from W to the power set of the set of all atomic mps, and then, extended to a relation on the cartesian product of W with the set of all mps, satisfying the conditions of Definition 6.3. Henceforth, we use $\Vdash$ as such a relation and a model will be written as $(W, R, \Vdash)$. However, there is a possible confusion in this notation and you should be aware of it. The relation $\Vdash$ changes from one model to another model. Earlier, the notation ϕ was keeping record of it. Now we must write $\Vdash$ as $\Vdash_M$ to remind ourselves of this dependence. As usual, we omit this subscript unless there is any confusion due to involvement of many models in a given context.

We want to define the notion of consequence in K. Suppose that A and B are two propositions in PL. We have defined the concept of $A \models B$ by relating models of A to those for B, i.e., when each model of A is also a model of B. Later, we have seen that the two metastatements '$A \models B$' and 'if $\models A$, then $\models B$' are equivalent. What has happend for FL? Suppose that A and B are two formulas (in FL). Then $A \models B$ holds whenever every state that satisfies A also satisfies B. In contrast, the metastatement 'if $\models A$, then $\models B$' is different from '$A \models B$'. The former statement means that whenever every state satisfies A, we also see that every state satisfies B. Even if this happens for some A, B, there can exist a state that satisfies A but falsifies B. For example, take the formulas A, B such that A is invalid but $A \not\models B$. In such a case, the metastatement 'if $\models A$, then $\models B$' holds vacuously, though $A \not\models B$. However, one side implication can be established. Whenever $A \models B$ holds, it is clear that 'if $\models A$, then $\models B$' also holds. Thus, in PL, both the notions are equivalent, but in FL, $A \models B$ is a stronger notion of consequence than 'if $\models A$, then $\models B$'.

The situation in K is similar to that in FL since truth in a model (in K) depends on quantification over all possible worlds. We have circumvented the difficulty in FL by sticking to the notion of consequence as meant by $A \models B$. This is so because we had certain application domains in our mind, for example, the arguments in natural languages, or programming. In the verification of programs, we have had to argue with open formulas. We say that the notion of "each state model of A is a state model of B" is more suitable to tackling open formulas than the notion expressed in the metastatement "if $\models A$, then $\models B$". This is so since the latter does not distinguish between open and closed formulas as $\models X$ is equivalent to $\models \forall^* X$ (the universal closure of X). However, in the modal logic K, we cannot discard it as such, as there will be situations which are better represented by the weaker version "if $\models A$, then $\models B$" of the notion of consequence. We define both the notions and modify the phrase 'valid consequence' by putting adjectives such as weak or strong. We also define another related notion of consequence by mixing the ideas together.

Definition 6.6 Let G, L be sets of modal propositions and A be a modal proposition. G **weakly entails** A, written as $G \overset{w}{\models} A$, iff for every model M, if M satisfies all the mps in G, then M satisfies A. L **strongly entails** A, written as $L \overset{s}{\models} A$, iff for every model $M = (W, R, \Vdash)$, for every world $w \in W$, if w satisfies all the mps in L, then w satisfies A. Further, A is a **valid consequence with global assumption** G **and local assumption** L , written as $G \models L \Rightarrow A$, iff for every model $M = (W, R, \Vdash)$ satisfying all the mps in G, and for every world $w \in W$ satisfying all the mps in L, $w \Vdash A$ holds.

Let M be a model and w be a world. If Σ is a set of mps, we write $M \models \Sigma$ whenever $M \models x$ for every $x \in \Sigma$. Similarly, if $w \Vdash x$ holds for every $x \in \Sigma$, we write $w \Vdash \Sigma$. Then, $G \models L \Rightarrow A$ holds iff for every model M satisfying $M \models G$ and for every world w in M satisfying $w \Vdash L$, one has $w \Vdash A$. If $G = \emptyset$, then every model M satisfies every mp in G vacuously; the same also happens when $G = \{\top\}$. In such cases, the validity of consequence with global assumption G and local assumption L reduces to the stronger entailment from L. That is, if $G = \emptyset$, or $G = \{\top\}$, then $G \models L \Rightarrow A$ is equivalent to $L \overset{s}{\models} A$. If $L = \emptyset$, then every world w in M (with $M \models G$) satisfies every mp in L vacuously; the same also happens if $L = \{\top\}$. Now the consequence becomes equivalent to the weak entailment from G. That is, both $G \models \emptyset \Rightarrow A$ and $G \models \{\top\} \Rightarrow A$ are equivalent to $G \overset{w}{\models} A$. It is also clear that strong entailment is stronger than the weak entailment as their names suggest, i.e., if $\Sigma \overset{s}{\models} A$, then $\Sigma \overset{w}{\models} A$.

If any of the sets G or L is a singleton, we will not write the braces around them while writing out the consequence relations. This will save space as $\{B\} \models \{C\} \Rightarrow A$ will be written as $B \models C \Rightarrow A$.

***EXAMPLE* 6.8** $\Box p \to p \overset{w}{\models} \Box\Box p \to \Box p$ but $\Box p \to p \overset{s}{\not\models} \Box\Box p \to \Box p$.
Solution For the first consequence to hold, we consider all the models which satisfy the mp $\Box p \to p$, and then show that any world in such a model also satisfies $\Box\Box p \to \Box p$. So, let $M = (W, R, \Vdash)$ be a model such that $M \models \Box p \to p$. Let $w \in W$ be such that $w \Vdash \Box\Box p$. Let $u \in W$ be a world accessible from w (uRw). Then $u \Vdash \Box p$. As $M \models \Box p \to p$, for every world $v \in W$, $v \Vdash \Box p \to p$. In particular, $u \Vdash \Box p \to p$. Using the fact that $u \Vdash \Box p$, we conclude that $u \Vdash p$. But u is any arbitrary world accessible from w. Hence, $w \Vdash \Box p$. We have shown that if $w \Vdash \Box\Box p$, then $w \Vdash \Box p$. Therefore, $w \Vdash \Box\Box p \to \Box p$.

For showing the non-satisfaction, consider the model: $\boxed{\text{w}} \longrightarrow \boxed{\text{u}} \nVdash p$. The $\nVdash$ sign after the world u says that u satisfies all atomic mps except p. Now, $w \nVdash \Box p$ since the only world accessible from w is u but $u \nVdash p$. Thus, $w \Vdash \Box p \to p$. Again, $w \Vdash \Box\Box p$ iff $u \Vdash \Box p$ iff every world accessible from u satisfies p, which holds vacuously. As $u \nVdash p$, we have $w \nVdash \Box p$ and then $w \Vdash \Box p \to p$. But $w \nVdash \Box\Box p \to \Box p$; therefore, $\Box p \to p \overset{s}{\not\models} \Box\Box p \to \Box p$.

***EXAMPLE* 6.9** Show that $\Box(p \to q) \overset{s}{\models} \Box p \to \Box q$.
Solution Let $M = (W, R, \Vdash)$ be a model and $w \in W$ be a world with $w \Vdash \Box(p \to q)$. If $w \nVdash \Box p$, then $w \Vdash \Box p \to \Box q$, and we are through. Otherwise, assume that $w \Vdash \Box p$. We want to show that $w \Vdash \Box q$. Let u be any world accessible from w. Since $w \Vdash \Box p$, we have $u \Vdash p$. Since $w \Vdash \Box(p \to q)$, we also have $u \Vdash p \to q$. Then, $u \Vdash q$ and, consequently, $w \Vdash \Box q$.

***EXAMPLE* 6.10** Show that $\Box p \to p \models \Box(\neg p \to p) \Rightarrow p \vee \Diamond p$.
Solution Here, $\Box p \to p$ is a global assumption and $\Box(\neg p \to p)$ is a local assumption. So, we consider all the models which satisfy $\Box p \to p$, and any world w in such a model with $w \Vdash \Box(\neg p \to p)$, and then try to show that w also satisfies p.

Let $M = (W, R, \Vdash)$ be a model with $M \models \Box p \to p$. Let $w \in W$ be such that $w \Vdash \Box(\neg p \to p)$. Due to $\to$-distributivity, we have $w \Vdash \Box\neg p \to \Box p$. (You can also see this independently.) Then, either $w \nVdash \Box\neg p$ or $w \Vdash \Box p$. If $w \nVdash \Box\neg p$, then $w \Vdash \neg\Box\neg p$. But by De Morgan's law, $\neg\Box\neg p \equiv \Diamond p$. Hence, $w \Vdash \Diamond p$. On the other hand, if $w \Vdash \Box p$, then with $w \Vdash \Box p \to p$ (since $M \models \Box p \to p$), we have $w \Vdash p$. In either case, $w \Vdash p \vee \Diamond p$.

We start exploring the properties of the consequence relations in K.

Theorem 6.5 (Monotonicity in K) *Let A be an mp and G, G', L, L' be sets of mps with $G \subseteq G'$ and $L \subseteq L'$. If $G \models L \Rightarrow A$, then $G' \models L' \Rightarrow A$.*

Proof Let $G \models L \Rightarrow A$. Let $M = (W, R, \Vdash)$ be a model with $M \models G'$. Let $w \in W$ satisfy L'. Then, $M \models G$ and also $w \Vdash L$. Since $G \models L \Rightarrow A$, we have $w \Vdash A$. Therefore, $G' \models L' \Rightarrow A$. ∎

Though the consequence $G \models L \Rightarrow A$ is a generalization of both strong and weak consequences, it is rarely used except while presenting such general results as monotonicity. We will rather use strong and weak consequences separately. Analogous to PL, the logic K is also compact (Formulate and prove compactness for K.) and it admits of the deduction theorems.

Theorem 6.6 (Strong Deduction) *Let L be a set of mps and A, B be mps. Then $L \cup \{A\} \overset{s}{\models} B$ iff $L \overset{s}{\models} A \to B$.*

Theorem 6.7 (Weak Deduction) *Let G be a set of mps and A, B be mps. Then $G \cup \{A\} \overset{w}{\models} B$ iff $G \models \{A, \Box A, \Box\Box A, \ldots\} \Rightarrow B$.*

***Exercise* 6.8** Prove both the strong and weak deduction theorems.

Due to the strong deduction theorem, $\models A \to B$ iff $A \overset{s}{\models} B$ iff $\overset{s}{\models} A \to B$. To have a better notation, we will write $\overset{s}{\models}$ simply as $\models$ retaining the weak entailment $\overset{w}{\models}$ as it is.

***Exercise* 6.9** What can you say about the following metastatements on the weak entailment:

(a) If $\Sigma \overset{w}{\models} A \to B$ then $\Sigma \cup \{A\} \overset{w}{\models} B$,

(b) If $\Sigma \cup \{A\} \overset{w}{\models} B$, then $\Sigma \overset{w}{\models} A \to B$?

6.3 Axiomatic System KC

Historically, axiomatic systems for modal logics precede the semantic characterization. We have taken a back-door approach. Since more modal logics are to come in succeeding sections, we first see how to characterize valid mps through proofs. As you know, in an axiomatic system, certain types of expressions (mps now) are just declared as theorems; these are the so-called axiom schemes. Axioms are obtained from the axiom schemes by replacing the letters by arbitrary expressions uniformly. Then certain rules of inference are prescribed, which are used to derive other theorems from the existing ones. We will keep the flavour of PC in presenting an axiomatic system, called **KC (K-calculus)** for the logic K. Since K is an extension of PL, the axioms of PL are also the axioms of K. However, we will make a short cut. Suppose that you want to show that $\Box p \to \Box p$ is a theorem in K. Then, you have to essentially repeat the proof of $p \to p$ in PL. We do not want to repeat such proofs, for they can be constructed using PC. Instead, we accept all the theorems of PC as KC-axioms. This will help us to avoid unnecessary aberrations so that we may concentrate on the peculiarity of K. Note that this will not affect the effectiveness of the axiomatic system KC since we can always go back to the three axioms of PC and the inference rule MP for deriving the PC-theorems. As in PC, we do not take all the connectives as basic symbols in KC. The connectives $\wedge, \vee, \leftrightarrow$ and the constants $\top, \bot$ are defined from $\neg, \to$ as in PC. Further, the modal connective $\Diamond$ is defined from $\neg, \Box$ via De Morgan's law, namely, $\Diamond A \equiv \neg\Box\neg A$. So, the basic symbols of KC are the propositional variables, and the connectives $\neg, \to$ and $\Box$. The following are the axiom schemes and the inference rules in KC:

Axiom Schemes of KC

(PC) Each theorem of PC.

(K) $\Box(X \to Y) \to (\Box X \to \Box Y)$

Rules of Inference of KC

(MP) $\dfrac{X \qquad X \to Y}{Y}$

(N) $\dfrac{X}{\Box X}$

The names of the axioms and the inference rules are self-suggestive: PC stands for **axiom-PC**, K stands for **axiom-K**, MP stands for modus ponens, and N for **necessitation**. As in FL, 'from X derive $\forall x X$'; necessitation says that from X, derive $\Box X$.

A **proof** in K is, again, a finite sequence of mps, each of which is either an axiom (an instance of an axiom scheme) or is derived from earlier mps by an application of an inference rule. The last mp in a proof (sequence) is called a **theorem** in K. If A is a theorem (in K), we also say that A is **provable** (in K) and also that A has a proof; the proof is said to **prove** A in K. If no other logic is involved, we will simply omit writing 'in K'.

Note that the rule N can be viewed through a proof in the following manner. If you already have a proof for X, then you have a proof for $\Box X$ as well, i.e., from "X is provable", derive "$\Box X$ is provable". It does not assert that $X \to \Box X$ is provable! We will follow the same three-column way of writing proofs.

Recollect the difficulties you have encountered while constructing proofs in PC in the absence of metatheorems such as the deduction theorem and RAA. So, instead of going for proofs straightaway, we will have some results of this kind. We will not attempt at many such results since our interest is not in generating more and more theorems in KC, but only to show that the logic K can be captured effectively by an axiomatic system.

Theorem 6.8 (Regularity) *The following rule of inference, called the rule of regularity, is a derived rule in KC.*

(R) $\dfrac{X \to Y}{\Box X \to \Box Y}$

Proof We only have to give a proof (a proof scheme) showing the rule R, which can be repeated in any proof so that, instead of repeating the whole proof, we can simply apply the rule. The proof for a derived rule, thus, starts with the premise $X \to Y$, and here it ends with the conclusion $\Box X \to \Box Y$. The proof is easy; from $X \to Y$, derive $\Box(X \to Y)$, then use (K) and (MP). Here is the proof (P for a premise):

1. $X \to Y$ P
2. $\Box(X \to Y)$ N
3. $\Box(X \to Y) \to (\Box X \to \Box Y)$ K
4. $\Box X \to \Box Y$ MP ▮

Before constructing proofs, we will have one more metatheorem. It is the substitution of equals by equals. We call it replacement instead of equivalence substitution.

Theorem 6.9 (Replacement) *Let X, X', Y, Y' be modal propositions such that Y occurs in X as a sub-modal proposition. Let X' be obtained from X by replacing some (or all or no) occurrences of Y in X by Y'. If $Y \leftrightarrow Y'$ is provable, then $X \leftrightarrow X'$ is also provable.*

Proof The proof is by induction on the number of occurrences of connectives in X. See its analogue in PC. ▮

We will use these metatheorems of Regularity (rule R) and Replacement (rule RP, formulate it!) in showing that certain mps are provable in KC. The fact that an mp A is a theorem in KC is written as $\vdash A$, as a shorthand to $\vdash_{KC} A$. We will resort to the full symbolism $\vdash_{KC} A$ whenever we feel that the shorthand is confusing. Note that we have already accepted all the connectives of PC, and all the PC-theorems as our axioms in KC. Since PC is adequate to PL, all the valid propositions can be used as axioms-P in KC. Then, we may use all the connectives of PL instead of $\neg$ and $\to$. We may also use the propositional constants $\top$ and $\bot$ if need arises.

***EXAMPLE* 6.11** Show that $\vdash \Box(X \wedge Y) \to (\Box X \wedge \Box Y)$.

Solution The following is a proof establishing the above:

1. $X \wedge Y \to X$ PC
2. $\Box(X \wedge Y) \to \Box X$ R
3. $X \wedge Y \to Y$ PC
4. $\Box(X \wedge Y) \to \Box Y$ R
5. $(\Box(X \wedge Y) \to \Box X) \to$
   $((\Box(X \wedge Y) \to \Box Y) \to (\Box(X \wedge Y) \to (\Box X \wedge \Box Y)))$ PC
6. $(\Box(X \wedge Y) \to \Box Y) \to (\Box(X \wedge Y) \to (\Box X \wedge \Box Y))$ 2, MP
7. $\Box(X \wedge Y) \to (\Box X \wedge \Box Y)$ 4, MP

We also use all the derived rules of PC as derived rules of KC in proofs. In the following example, we use the rule of hypothetical syllogism (HS).

***EXAMPLE* 6.12** Show that $\vdash (\Box X \wedge \Box Y) \to \Box(X \wedge Y)$.

Solution Here is a proof:

1.	$X \to (Y \to (X \wedge Y))$	PC
2.	$\Box(X \to (Y \to (X \wedge Y))$	N
3.	$\Box X \to \Box(Y \to (X \to Y))$	R
4.	$\Box(Y \to (X \to Y)) \to (\Box Y \to \Box(X \to Y))$	K
5.	$\Box X \to (\Box Y \to \Box(X \wedge Y))$	HS
6.	$(\Box X \wedge \Box Y) \to \Box(X \wedge Y)$ R	

Note that the derivation of line 6 from line 5 has been mentioned as the replacement, since the modal proposition in line 6 is equivalent to that in line 5. You can also use a PC theorem and then MP for deriving the same. Further, since each theorem is a theorem scheme, you can use them in constructing proofs. We use the theorem proved in Example 6.12 in constructing the proof in the following example.

***EXAMPLE* 6.13** Show that $\vdash \Box(p \to q) \wedge \Box(q \to r) \to \Box(p \to r)$.

Solution

1.	$\Box(p \to q) \wedge \Box(q \to r) \to \Box((p \to q) \wedge (q \to r))$	Ex.6.12
2.	$(p \to q) \wedge (q \to r) \to (p \to r)$	PC
3.	$\Box((p \to q) \wedge (q \to r)) \to \Box(p \to r)$	R
4.	$\Box(p \to q) \wedge \Box(q \to r) \to \Box(p \to r)$	$1,\mathrm{HS}$

***EXAMPLE* 6.14** Show that $\vdash \Diamond\Diamond A \leftrightarrow \neg\Box\Box\neg A$.

Solution

1.	$\Diamond\Diamond A \leftrightarrow \neg\Box\neg\Diamond A$	Definition of $\Diamond$
2.	$\Diamond A \leftrightarrow \neg\Box\neg A$	Definition of $\Diamond$
3.	$\Diamond\Diamond A \leftrightarrow \neg\Box\neg\neg\Box\neg A$	$1, 2, \mathrm{R}$
4.	$\neg\neg\Box\neg A \leftrightarrow \Box\neg A$	PC
5.	$\Diamond\Diamond A \leftrightarrow \neg\Box\Box\neg A$	$3, 4, \mathrm{R}$

Is the system KC adequate to the logic K? To show the soundness of KC, we have to verify that each axiom of KC is valid (K-valid). Further, we must ensure that by using the inference rules of KC on valid mps, we get only valid mps.

***Exercise* 6.10** Show that each axiom of KC is K-valid. In fact, we have already done it; find out where.

With Exercise 6.10, let us look at the inference rules. The rule MP is sound; it follows from the soundness of PC. But let us see this in the modal setting of K. Suppose that $M = (W, R, \Vdash)$ is a model and $w \in W$. If $M \models p$ and $M \models p \to q$, then $w \Vdash p$ and $w \Vdash p \to q$. As each world is propositional, we have $w \Vdash q$, by MP in the world w. Since w is any arbitrary world, we conclude that $M \models q$; thus follows the soundness of MP in K.

What about necessitation? Here, we must also argue with accessible worlds. Let $M = (W, R, \Vdash)$ be a model and $M \models p$. We want to show that for every world $w \in W$, $w \Vdash \Box p$. So, let $w \in W$ be an arbitrary world. Let $u \neg W$ be such that uRw, a world accessible from w. Since $M \models p$, for every world $v \in W$, $v \Vdash p$. In particular, $u \Vdash p$. It follows then that $w \Vdash \Box p$; consequently, $M \models \Box p$. This completes the soundness of necessitation. Finally, use induction on the length of proofs to conclude that every KC-provable mp is K-valid as stated in the following metatheorem.

Theorem 6.10 (Soundness of KC) *Let X be any modal proposition. If $\vdash X$ in KC, then $\models X$ in K.*

As it happened for PC, completeness involves more work. We will take the approach of maximally consistent sets of modal propositions and prove the Lindenbaum Lemma. In order to do that we give some definitions.

Definition 6.7 A finite set of modal propositions $\{X_1, X_2, \ldots, X_n\}$ is called **KC-consistent**, if $X_1 \wedge X_2 \wedge \cdots \wedge X_n \rightarrow \bot$ is not provable in KC. An infinite set is KC-consistent if every finite subset of it is KC-consistent. A set Σ of modal propositions is **maximally KC-consistent** iff Σ is consistent and, if any superset $\Sigma' \supseteq \Sigma$ is KC-consistent, then $\Sigma' = \Sigma$.

Lemma 6.11 (Lindenbaum Lemma) *For any KC-consistent set Σ of modal propositions, there exists a maximally consistent set Σ' of modal propositions such that $\Sigma \subseteq \Sigma'$.*

Proof Since the set of all mps is countable, take the enumeration of it as $X_0, X_1, X_2, \ldots$. Let Σ be the given KC-consistent set of mps. Define inductively the sequence of sets by setting $\Sigma_0 = \Sigma$ and

$$\Sigma_{n+1} = \Sigma_n \cup \{X_{n+1}\}, \text{ if } \Sigma_n \cup \{X_{n+1}\} \text{ is KC-consistent; else, } \Sigma_{n+1} = \Sigma_n.$$

As in PC, each Σ_n is KC-consistent (Show it.). Let $\Sigma' = \cup_{n \in \mathbb{N}} \Sigma_n$. Now, $\Sigma \subseteq \Sigma'$. If Σ' is not KC-consistent, then there is a finite subset, say, $\Sigma^* \subseteq \Sigma'$ such that Σ^* is not KC-consistent. Due to the enumeration of all mps, $\Sigma^* \subseteq \{X_0, X_1, \ldots, X_m\}$ for some $m \in \mathbb{N}$. But then, $\Sigma^* \subseteq \Sigma_m$ (Why?). This implies that Σ_m is not KC-consistent, a contradiction. Thus, Σ' is KC-consistent.

If Σ' is not maximally KC-consistent, then there is a proper superset of Σ' which is KC-consistent. Due to the enumeration of all mps, we get one X_k such that $X_k \notin \Sigma'$ and $\Sigma' \cup \{X_k\}$ is consistent. Since $\Sigma_{k-1} \subseteq \Sigma'$, $\Sigma_k \subseteq \Sigma' \cup \{X_k\}$, and $\Sigma_{k-1} \cup \{X_k\}$ is KC-consistent, we have $X_k \in \Sigma_k \subseteq \Sigma'$. This contradicts the fact that $X_k \notin \Sigma'$. Thus, Σ' is the required maximally KC-consistent set (extension of Σ). ▮

Our next job is to see how to use a maximally KC-consistent extension of Σ to construct a model that would satisfy it. Note that there can be many maximally KC-consistent extensions of the same set Σ. However, for now, we consider all possible maximally KC-consistent sets of mps. Each one of them is a maximally KC-consistent extension of some set of mps. We take W, our set of worlds, as the set of all such maximally KC-consistent sets. For worlds $w, u \in W$, define the relation R (a subset of $W \times W$) by

wRu iff for each mp of the form $\Box X \in W$, $X \in u$.

Define the satisfaction ($\Vdash$) of an atomic modal proposition at a world w by

for every atomic mp p, $w \Vdash p$ iff $p \in w$.

With such a **canonical model** obtained from the set of all maximally KC-consistent extensions of Σ, we have the following result.

Lemma 6.12 (Truth Lemma) *Let $M = (W, R, \Vdash)$ be the canonical model as defined above. Let X be any modal proposition and w be any world in W. Then $w \Vdash X$ iff $X \in w$.*

Proof Use induction on the number of occurrences of connectives in X. ▮

We are ready to prove the completeness of KC.

Theorem 6.13 (Completeness of KC) *Let X be any modal proposition. If $\models X$ in K, then $\vdash X$ in KC.*

Proof We show the contrapositive. Suppose that X is not KC-provable. Since $X \leftrightarrow (\neg X \to \bot)$ is provable in PC, $\nvdash \neg X \to \bot$. That is, the set $\{\neg X\}$ is consistent. The Lindenbaum lemma says that there is a maximally KC-consistent extension w of $\{\neg X\}$. That is, w is a world in the canonical model $M = (W, R, \Vdash)$, as in the Truth lemma, such that $\neg X \in w$ and w is maximally KC-consistent. Since w is consistent, $X \notin w$. (Otherwise, $\vdash X \wedge \neg X \to \bot$ can be used to show that w is not KC-consistent.) By the Truth lemma, $w \nVdash X$. Thus, $M \nvDash X$; consequently, X is not KC-valid. ▮

Theorems 6.10 and 6.13 together show that KC is adequate to the logic K. What about the strong adequacy? But we have not yet defined the concept of a consequence in KC! How do we formulate the proof theoretic versions of $\Sigma \overset{s}{\models} X$ and of $\Sigma \overset{w}{\models} X$? We will first extend the idea of a consequence in KC, which seems most natural to us, and then see whether it is adequate to strong or weak entailment. The following is an example of a derivation.

***EXAMPLE* 6.15** Show that $\{\Box p \to p, \Box\Box p\} \vdash \Box p$ in KC.

Solution

1.	$\Box p \to p$	P
2.	$\Box(\Box p \to p)$	N
3.	$\Box(\Box p \to p) \to (\Box\Box p \to \Box p)$	K
4.	$\Box\Box p \to \Box p$	MP
5.	$\Box\Box p$	P
6.	$\Box p$	MP

This example suggests the following definition of a KC-derivation.

Definition 6.8 Let Σ be a set of mps and X be an mp. A **KC-derivation** of X from Σ is a finite sequence of mps such that each mp in the sequence is either an axiom of KC, or a member of Σ (a premise), or is derived from earlier mps by an application of modus ponens (MP) or necessitation (N); further, X must be the last mp of the sequence. If there is a KC-derivation of X from Σ, we write $\Sigma \vdash X$ in KC (for precision, $\Sigma \vdash_{KC} X$).

The above definition, in fact, defines the concept of provability of a **KC-consequence**. If there is a KC-derivation of X from Σ, we say that "the consequence $\Sigma \vdash X$ is KC-provable" or that "the KC-consequence $\Sigma \vdash_{KC} X$ is provable". We often omit writing the outer braces while writing a finite set of premises, and use the same three-column style of writing a derivation. Note that any derived rule of inference, which comes from earlier derivations, can also be used in a derivation.

***EXAMPLE* 6.16** Show that $\{\Box p \to p, p \to q\} \vdash \Box\Box p \to q$ in KC.

Solution

1.	$\Box p \to p$	P
2.	$\Box\Box p \to \Box p$	R
3.	$\Box\Box p \to p$	HS
4.	$p \to q$	P
5.	$\Box\Box p \to q$	HS

Since the rule of necessitation is used in a derivation, Definition 6.8 captures the notion of the weak entailment. Using adequacy of KC to K, and the deduction theorem (which one?), you can prove that $\Sigma \vdash X$ in KC iff $\Sigma \overset{w}{\models} X$. Since $X \vdash \Box X, \nvdash X \to \Box X$, and $\models X \to \Box X$ is equivalent to $X \overset{s}{\models} \Box X$, the notion of KC-consequence of Definition 6.8 does not capture the strong entailment.

***Exercise* 6.11** Let Σ be a set of mps and X be any mp. Then show that $\Sigma \vdash X$ in KC iff $\Sigma \overset{w}{\models} X$.

In general, there is also a notion of axiomatic version of a consequence with global and local assumption; see the Summary at the end of this chapter. In parallel with the strong entailment relation, we do not have 'strong adequacy' in the case of a consequence with global and local assumptions. The situation reminds us the inference rule of universal generalization (UG) in FC. UG states that from X, derive $\forall x X$. In the case of FC and FL, we have seen that $\Sigma \vdash X$ and $\Sigma \models X$ coincide provided that in the proof of $\Sigma \vdash X$, UG has not been used on a free variable of any premise (just as in the deduction theorem). We have not mentioned this condition in the strong adequacy theorem of FC because, our formulation of UG in FC already takes care of this restriction. This restricted version cannot be formulated in KC since in a derivation of $\Box X$, we do not have any trace of a world. However, in a natural deduction proof, we might succeed by using a box as a book-keeping device for worlds. Then, how do we take care of the notion of strong entailment for consequences in an axiomatic system like KC? Can we replace the necessitation rule? You must look at the strong deduction theorem for such a modification of KC.

6.4 Other Proof Systems for K

In this section we present two more proof systems: the natural deduction and the analytic tableau. For a natural deduction system, we keep all the inference rules of PND, remembering that the propositions are now treated as mps. We must have some rules for tackling $\Box$ and $\Diamond$. We plan to have an introduction rule for $\Box$, an elimination rule for $\Box$, and then regard the connective $\Diamond$ as $\neg\Box\neg$ as in KC.

The natural deduction system for K is named as **KND**. The system KND includes all the rules such as $\wedge i, \wedge e, \vee i, \vee e \ldots$ of Section 4.5. The extra rules for handling the connective $\Box$ are:

$$(\Box i)\quad \frac{\boxed{\;\ldots X\;}}{\Box X} \qquad\qquad (\Box e)\quad \frac{\Box X}{\boxed{\;\ldots X \ldots\;}}$$

Before an explanation of the dashed boxes, try to make out why the proof in the following example is, in fact, a proof.

***EXAMPLE* 6.17** Show, in KND, that $\vdash \Box(p \to q) \to (\Box p \to \Box q)$.

Solution Here is a proof:

$$\begin{array}{|ll|}
\hline
1.\ \Box(p \to q) & \text{CP} \\
\multicolumn{2}{|c|}{\begin{array}{|ll|}
\hline
2.\ \Box p & \text{CP} \\
\multicolumn{2}{|c|}{\begin{array}{|ll|}
\hline
3.\ p & \Box e \\
4.\ p \to q & 1, \Box e \\
5.\ q & \text{MP} \\
\hline
\end{array}} \\
6.\ \Box q & \Box i \\
\hline
\end{array}} \\
7.\ \Box p \to \Box q & \to e \\
\hline
\end{array}$$

$$8.\ \Box(p \to q) \to (\Box p \to \Box q) \qquad \to i$$

Could you make out the purpose of the dashed boxes? These proof boxes for the connective $\Box$ are different from the solid boxes used for other rules. The contents inside the dashed boxes are written horizontally, but in a proof, they come vertically. In the rule ($\Box i$), the ... represent many mps preceding the mp X. Similarly, the ... before and after X in the rule ($\Box e$) also represent possibly many mps preceding and following X. These rules are intended to reason in any arbitrary accessible world. Recollect that going into a solid box means that assuming the first formula in it, and after it is closed, the conclusion 'first line $\to$ last line' is written on the line immediately below the box.

The dashed boxes serve a similar but different purpose. For the rule ($\Box i$), it means that a dashed box can be created with any opening line, but that line should have been justified by other rules (unlike an extra assumption in a solid box). The box begins there, meaning that from that line onwards, till the box closes, we are reasoning in a fixed but arbitrary world. Now, when we deduce X in this arbitrary world, we can close the box, showing that in any arbitrary world, X holds. Therefore, $\Box X$ must hold. This is recorded by closing the box and writing $\Box X$ on the following line. Similarly, if $\Box X$ occurs as a line in a proof, then, in any arbitrary world the mp X is true. and this time the scope for going into this arbitrary world can be any fragment of the proof. This is the rule ($\Box e$). The rules for $\Box$-introduction and $\Box$-elimination can be described as

($\Box i$): If X occurs at the end of a dashed box, then $\Box X$ may be introduced after (closing) the dashed box.

($\Box e$): If $\Box X$ occurs somewhere in a proof, then X may be introduced anywhere into a subsequent dashed box.

In general, the dashed boxes are introduced due to an occurrence of $\Box X$. In KND-proofs, both solid and dashed boxes would appear, and they may be nested in any arbitrary manner. As in PND, we will have a justification column documenting whether the mp is a premise or it follows from earlier mps by using some inference rule(s). Again, for the provability of a consequence with premises in Σ and conclusion X, we write $\Sigma \vdash_{KND} X$. If no confusion arises, we write $\Sigma \vdash X$, for ease. Revisit Example 6.17 and understand the proof. Here is another example.

***EXAMPLE* 6.18** Show in KND that $\vdash \Box(p \land q) \leftrightarrow \Box p \land \Box q$.

Solution To show the biconditional, we must show the two implications:

(a) $\vdash \Box(p \land q) \to \Box p \land \Box q$ (b) $\vdash \Box p \land \Box q \to \Box(p \land q)$

The proofs of these implications are given in Figures 6.2–6.3.

Now you can try proving adequacy of KND, i.e., $\Sigma \vdash_{KND} X$ iff $\Sigma \overset{w}{\models} X$. Since KND is an extension of PND, you can abbreviate KND-proofs by assuming all PND consequences as a single rule just as we have taken all PC-theorems as a single axiom in KC. The other two rules, of course, are ($\Box i$) and ($\Box e$). In doing so, you will obtain shorter proofs (quasi-proofs) which can be expanded to proofs in PND. See the following example for such a shortened proof. We will mention ‘PND’ on the third column whenever we use a consequence which has a PND-proof. You are supposed to check whether such consequences have PND-proofs or not.

1.	$\Box(p \wedge q)$	P
2.	$p \wedge q$	$\Box e$
3.	p	$\wedge e$
4.	$\Box p$	$\Box i$
5.	$p \wedge q$	$1, \Box e$
6.	q	$\wedge e$
7.	$\Box q$	$\Box i$
8.	$\Box p \wedge \Box q$	$4, 7, \wedge i$
9.	$\Box(p \wedge q) \to \Box p \wedge \Box q$	$\to i$

Figure 6.2 Proof for Example 6.18(a).

1.	$\Box p \wedge \Box q$	P
2.	$\Box p$	$\wedge e$
3.	$\Box q$	$1, \wedge e$
4.	p	$2, \Box e$
5.	q	$3, \Box e$
6.	$p \wedge q$	$\wedge i$
7.	$\Box(p \wedge q)$	$\Box i$
8.	$\Box p \wedge \Box q \to \Box(p \wedge q)$	$\to i$

Figure 6.3 Proof for Example 6.18(b).

***EXAMPLE* 6.19** Show that $\vdash_{KND} \Diamond(p \to q) \to (\Box p \to \Diamond q)$.

Solution We plan to have a shortened proof using PND-consequences as rules. You must expand the step where ‘PND’ is mentioned by supplying a PND-proof of the quoted or used consequence. The following is a proof of the above mp.

```
┌──────────────────────────────────────────────────┐
│     1. ◇(p → q)                               CP │
│ ┌──────────────────────────────────────────────┐ │
│ │   2. □p                                   CP │ │
│ │ ┌──────────────────────────────────────────┐ │ │
│ │ │ 3. ¬◇q                                CP │ │ │
│ │ │ 4. ¬¬□¬q                        Def of ◇ │ │ │
│ │ │ 5. □¬q                               PND │ │ │
│ │ │ ┌ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ┐        │ │ │
│ │ │ | 6. p                     □e |          │ │ │
│ │ │ | 7. ¬q                    □e |          │ │ │
│ │ │ | 8. ¬(p → q)       6, 7, PND |          │ │ │
│ │ │ └ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ┘        │ │ │
│ │ │ 9. □¬(p → q)                          □i │ │ │
│ │ │ 10. ¬□¬(p → q)               1, Def of ◇ │ │ │
│ │ └──────────────────────────────────────────┘ │ │
│ │   11. ◇q                       3, 9, 10, PND │ │
│ └──────────────────────────────────────────────┘ │
│     12. □p → ◇q                               →i │
└──────────────────────────────────────────────────┘
      13. ◇(p → q) → (□p → ◇q)    →i
```

With this brief exposure to natural deduction for K, we turn to another proof method, the method of analytic tableau. Among many ways of extending tableau method of PL to K, we choose one which is most intuitive. Keep in mind that somehow we must keep track of the worlds.

In KC, we have had difficulty with the rule "from X derive $\Box X$". It has led us to choose between the stronger or weaker entailments ($\overset{s}{\models}$ or $\overset{w}{\models}$) that would be captured by KC. We have seen that this rule does not keep track of the world in which the mp X might have been satisfied so that we had to interprete the rule "from X derive $\Box X$" metalogically. That is, this rule has been interpreted as "for every model M, if M satisfies X, then M satisfies $\Box X$". This problem has been slightly improved in KND by using the dashed boxes. A dashed box (for the rule $\Box e$) says that if an mp X is satisfied in any arbitrary world, then $\Box X$ must also be satisfied in every world. This means that "for every model M, if every world in M satisfies X, then every world in M also satisfies $\Box X$". You see that it is the same as in KC; the dashed boxes *do not* really improve the situation. We need to improve it to the level where we may assert that "for every model M, for every world w in M, if w satisfies X, then w satisfies $\Box X$".

This is similar to the eigenvariable condition in GFC. The idea is that the variable on which universal generalization is applied is somehow tracked in the proof. To achieve the same effect here, we must somehow keep track of the satisfaction of an mp in a world. Suppose that we use the numbers $1, 2, 3, \ldots$ as names of worlds (instead of $w_1, w_2, w_3, \ldots$). But then, we must also have a way to represent the worlds that are accessible from these worlds. Note that in the semantics of K, the valid mps are independent of particular accessibility relations (independent of particular models). Then how are we to represent any accessible relation? That means, we do not know beforehand whether a world named 2 is accessible from the world named 1 or not. In the absence of any, we must be able to invent a naming system, especially when we know that valid mps do not depend on any such particular naming systems; however, we should not also lose any world! A simple way is to name all the worlds accessible from 1 as 1.1, 1.2, 1.3, . . . Similarly, a world accessible from 1.1 would be named as 1.1.1, 1.1.2, 1.1.3, . . . Though these prefixes are countable in number, we do not lose any world just as in FL, where it was enough to have only countable models. A formal proof will involve the completeness of the tableau method, which we are not planning to discuss. Remember that these numbers with dots stand for names of worlds, and they are to be used as prefixes of mps, meaning that the mp is true at the world which is written as a prefix. Here is a formal definition.

Definition 6.9 A **modal prefix** is a finite sequence of positive integers, the integers being separated by a dot in between. If σ is a modal prefix and X is a modal proposition, then $\sigma\ X$ written with a blank in between is called a **prefixed modal proposition**.

For example, 1.2.1 $\Box X \to X$ is a prefixed modal proposition, with the modal prefix 1.2.1. The modal prefix stands for a world accessible from a world named 1.2, which is again accessible from the world named 1. The method of analytic tableau uses the prefixed modal propositions. The rules for expansion of a tableau, called the tableau rules, are the same as those in PT, each coming with a prefix now. In addition, we also have rules for the two extra connectives $\Box$ and $\Diamond$. Just like $\wedge$ and $\vee$ the modal connectives $\Box$ and $\Diamond$ act dually.

Tableau expansion rules for K: The four types of rules – Stacking rules, Branching rules, Necessity rules, and the Possibility rules, are now described.

(i) *Stacking Rules*

$$(\neg\neg)\quad \frac{\sigma\ \neg\neg X}{\sigma\ X} \qquad\qquad (\bot)\quad \frac{\sigma\ \bot}{\sigma\ \neg\top}$$

$$(\wedge)\quad \frac{\sigma\ (X \wedge Y)}{\begin{array}{c}\sigma\ X\\ \sigma\ Y\end{array}} \qquad\qquad (\neg\vee)\quad \frac{\sigma\ \neg(X \vee Y)}{\begin{array}{c}\sigma\ \neg X\\ \sigma\ \neg Y\end{array}}$$

$$(\neg\to)\quad \frac{\sigma\ \neg(X \to Y)}{\begin{array}{c}\sigma\ X\\ \sigma\ \neg Y\end{array}}$$

(ii) *Branching Rules*

$(\vee)\quad \dfrac{\sigma\ (X \vee Y)}{\sigma\ X \qquad \sigma\ Y}$ $\qquad (\neg\wedge)\quad \dfrac{\sigma\ \neg(X \wedge Y)}{\sigma\ \neg X \qquad \sigma\ \neg Y}$

$(\rightarrow)\quad \dfrac{\sigma\ (X \rightarrow Y)}{\sigma\ \neg X \qquad \sigma\ Y}$

$(\leftrightarrow)\quad \dfrac{\sigma\ (X \leftrightarrow Y)}{\begin{array}{cc}\sigma\ X & \sigma\ \neg X\\ \sigma\ Y & \sigma\ \neg Y\end{array}}$ $\qquad (\neg\leftrightarrow)\quad \dfrac{\sigma\ \neg(X \leftrightarrow Y)}{\begin{array}{cc}\sigma\ X & \sigma\ \neg X\\ \sigma\ \neg Y & \sigma\ Y\end{array}}$

(iii) *Necessity Rules*

$(\Box)\quad \dfrac{\sigma\ \Box X}{\sigma.n\ X}$ $\qquad (\neg\Diamond)\quad \dfrac{\sigma\ \neg\Diamond X}{\sigma.n\ \neg X}$

where n is any positive integer.

(iv) *Possibility Rules*

$(\Diamond)\quad \dfrac{\sigma\ \Diamond X}{\sigma.n\ X}$ $\qquad (\neg\Box)\quad \dfrac{\sigma\ \neg\Box X}{\sigma.n\ \neg X}$

where the prefix $\sigma.n$ is new to the branch.

We have seen that in the rules for $\forall$ and $\neg\exists$, instead of 'any term', it is enough to take only those terms which already occur in the branch. Analogously, in the necessity rules it is sufficient to use the prefix $\sigma.n$ which have already occurred in the branch. Again, in the same vein, $\sigma.n$ in the possibility rules must be new prefixes. The appearance of $\sigma\ \Box X$ in a branch means that "$\Box X$ is true in a world named σ". Then, X must be true in every world named $\sigma.n$ accessible from σ. Similar explanation is given for the possibility rules.

The tableau proof of a modal proposition X starts with the prefixed modal proposition $1\ \neg X$. Then the rules are applied to extend the tree. A path in such a tree (tableau) is called a **closed path** when it contains two mps of the form $\sigma\ Y$ and $\sigma\ \neg Y$. Note that here the prefix σ can be any prefix (with dots), but then we must have two prefixed mps with the same prefix where the mps are negations of each other. A path which is not closed, is called an **open path**. A tableau is called a **closed tableau** if every path of it is a closed path. A closed tableau for the prefixed modal proposition $1\ \neg X$ means that there cannot be any world named 1 where $\neg X$ may be true (satisfied). However, the name 1 is an arbitrary name; thus it means that the mp $\neg X$ cannot be true in any world. This is again a proof by contradiction showing that the mp X is K-valid. The closed tableau with $1\ \neg X$ as its root is called a **tableau proof** of the mp X.

A **theorem** in the tableau method is then an mp Z for which there is a tableau proof. We will write $\vdash_{KT} X$ (or just $\vdash X$ if no confusion arises) whenever the mp X is a tableau theorem. The tableau system with the above rules is named as **KT**.

***EXAMPLE* 6.20** Construct a tableau proof for $\vdash \Box p \wedge \Box q \to \Box(p \wedge q)$.

Solution As earlier, you go on applying tableau rules and keep an eye on the paths. As and when they close, you must not go for further expansions. You must also try to justify each line in the tableau, which we are not doing here. The justification means writing the previous line number from which the current mp is obtained. But you have the added responsibility now of taking account of the prefix. Here is the tableau:

$$\begin{array}{c}
1\ \neg(\Box p \wedge \Box q \to \Box(p \wedge q) \\
1\ \Box p \wedge \Box q \\
1\ \neg\Box(p \wedge q) \\
1\ \Box p \\
1\ \Box q \\
1.1\ \neg(p \wedge q) \\
\begin{array}{cc}
1.1\ \neg p \qquad\qquad & \qquad\qquad 1.1\ \neg q \\
1.1\ \ p \qquad\qquad & \qquad\qquad 1.1\ \ q \\
\times \qquad\qquad & \qquad\qquad \times
\end{array}
\end{array}$$

Mark the introduction of dots in the prefixes. The prefixed mp $1.1\ \neg(p \wedge q)$ is obtained by applying $(\neg\Box)$ on the second line. So, your justification on the right of line 6 would be a mere mention of '(2)'. The prefixed mps $1.1\ p$ and $1.1\ q$ on the last line are obtained from the lines 4 and 5.

***EXAMPLE* 6.21** Show by tableau method that $\vdash \Box p \wedge \Diamond q \to \Diamond(p \wedge q)$.

Solution

$$\begin{array}{c}
1\ \neg(\Box p \wedge \Diamond q \to \Diamond(p \wedge q)) \\
1\ \Box p \\
1\ \Diamond q \\
1\ \neg\Diamond(p \wedge q) \\
1.1\ q \\
1.1\ \neg(p \wedge q) \\
\begin{array}{cc}
1.1\ \neg p \qquad\qquad & \qquad\qquad 1.1\ \neg q \\
1.1\ \ p \qquad\qquad & \qquad\qquad \times \\
\times \qquad\qquad &
\end{array}
\end{array}$$

Look at the prefixes with dots; 1.1 q comes from $\Diamond q$, where 1.1 is also a new prefix. The next prefixed mp 1.1 $\neg(p \wedge q)$ comes from $\neg\Diamond(p \wedge q)$ by applying the necessity rule $(\neg\Diamond)$; this allows an old prefix. Other prefixed mps are obtained propositionally (by stacking and branching rules).

***EXAMPLE* 6.22** Show that $\vdash_{KT} \Box p \vee \Box q \to \Box(p \vee q)$.

Solution

$$\begin{array}{ccc}
 & 1\ \neg(\Box p \vee \Box q \to \Box(p \vee q)) & \\
 & 1\ \Box p \vee \Box q & \\
 & 1\ \neg\Box(p \vee q) & \\
 & 1.1\ \neg(p \vee q) & \\
 & 1.1\ \neg p & \\
 & 1.1\ \neg q & \\
1\ \Box p & & 1\ \Box q \\
1.1\ p & & 1.1\ q \\
\times & & \times
\end{array}$$

What about consequences? Since we are able to keep track of the worlds where an mp may be true, we expect to be able to capture the general consequence of the type $G \models L \Rightarrow X$. Recollect that in such a consequence, G is a set of global assumptions and L is a set of local assumptions. The consequence $G \models L \Rightarrow X$ holds (or is K-valid) when X is true in a world of a model at which all the members of L are true and if all the members of G are true at each world of that model. That is, if a world is named as 1, then the conclusion X cannot be falsified by 1, whenever the local assumptions are true in the same world named 1. Hence the prefixes of each member of L and X must be the same, which we are taking here as 1. Next, a global assumption is one which is satisfied in every such world. Thus any member of G may be prefixed in any way we like. We will then have two additional tableau rules for handling the local and global assumptions.

Let G and L be sets of modal propositions. Let $X \in L$ and $Y \in G$ be two modal propositions. The rules are given as

$$\textbf{(LA)}\quad \frac{\cdot}{1\ X} \qquad\qquad \textbf{(GA)}\quad \frac{\sigma \quad \cdots}{\sigma\ Y}$$

The rules of **local assumption (LA)** says that

if $X \in L$, then add 1 X to any open path of the tableau.

The rule of **global assumption (GA)** says that

if $Y \in G$ and σ is any prefix occurring in an open path of the tableau, then add $\sigma\ Y$ to that path.

We start the tableau with $1\ \neg X$ on the root and expand it using all the tableau rules including LA and GA. If the tableau closes, then it is called a **derivation** of the consequence $G \models L \Rightarrow X$. Keep it in mind that $\emptyset \models L \Rightarrow X$ is the strong consequence (entailment) $L \overset{s}{\models} X$, and $G \models \emptyset \Rightarrow X$ is the weak consequence $G \overset{w}{\models} X$.

***EXAMPLE* 6.23** Attempt tableau derivations of the following:
(a) $\Box p \to p \overset{w}{\models} \Box\Box p \to \Box p$ (b) $\Box p \to p \overset{s}{\models} \Box\Box p \to \Box p$.

Solution The difference between (a) and (b) is that the mp $\Box p \to p$ is taken as a global assumption in (a), while in (b), it is taken as a local assumption. Check the terminology once again; when something is assumed globally, the assumption is stronger, consequently, the consequence becomes weaker. While, if the same mp is assumed locally, the assumption is a weaker assumption, and if the consequence holds, it is a stronger consequence. Here are the tableaux:

Tableau for (a):

$$
\begin{array}{ccc}
 & 1\ \neg(\Box\Box p \to \Box p) & \\
 & 1\ \Box\Box p & \\
 & 1\ \neg\Box p & \\
 & 1.1\ \neg p & \\
 & 1.1\ \Box p & \\
 & 1.1\ \Box p \to p & \\
1.1\ \neg\Box\Box p & & 1.1\ p \\
\times & & \times
\end{array}
$$

The sixth line is the global assumption and the prefix 1.1 is used with this according to the rule (GA). Give justifications to the other lines. Since the tableau closes, it is a derivation establishing the consequence in (a).

Tableau for (b):

$$
\begin{array}{ccc}
 & 1\ \neg(\Box\Box p \to \Box p) & \\
 & 1\ \Box p \to p & \\
 & 1\ \Box\Box p & \\
 & 1\ \neg\Box p & \\
1\ \neg\Box p & & 1\ p \\
1.1\ \neg p & & 1.1\ \neg p \\
1.1\ \Box p & & \\
1.1.1\ p & &
\end{array}
$$

Neither the left path nor the right path closes since the prefixes of the complementary propositions do not match. Hence this tableau is not a derivation of the consequence in (b). It is no surprise, for we have seen already that $\Box p \to p \not\models^{s} \Box\Box p \to \Box p$.

***EXAMPLE* 6.24** Construct a tableau derivation for the consequence:

$$\Diamond(p \to q) \overset{s}{\models} \Box p \to \Diamond q$$

Solution Below is a tableau derivation. In this tableau, the fourth line is introduced due to the rule (LA). Annotate the tableau with appropriate justifications. Mention when a premise is used by writing the corresponding rule (LA) or (GA).

$$\begin{array}{c}
1\ \neg(\Box p \to \Diamond q)\\
1\ \Box p\\
1\ \neg\Diamond q\\
1\ \Diamond(p \to q)\\
1.1\ p \to q\\
\begin{array}{cc}
1.1\ \neg p \qquad\qquad & \qquad\qquad 1.1\ q\\
1.1\ p \qquad\qquad & \qquad\qquad 1.1\ \neg q\\
\times \qquad\qquad & \qquad\qquad \times
\end{array}
\end{array}$$

***EXAMPLE* 6.25** Construct a tableau derivation for the consequence:

$$\{p \to \Box p, q \to \Box q\} \models \Box p \land \Box q \Rightarrow \Box(\Box p \land \Box q)$$

Solution In the following derivation, LA has been applied for introducing the first line and GA, for the ninth line.

$$\begin{array}{c}
1\ \Box p \land \Box q\\
1\ \neg\Box(\Box p \land \Box q)\\
1.1\ \neg(\Box p \land \Box q)\\
1\ \Box p\\
1\ \Box q\\
1.1\ p\\
1.1\ q\\
\begin{array}{cc}
1.1\ \neg\Box p & 1.1\ \neg\Box q\\
1.1\ p \to \Box p & 1.1\ q \to \Box q\\
\begin{array}{cc} 1.1\ \neg p & 1.1\ \Box p\\ \times & \times \end{array} &
\begin{array}{cc} 1.1\ \neg q & 1.1\ \Box q\\ \times & \times \end{array}
\end{array}
\end{array}$$

***Exercise* 6.12** Attempt tableau derivations for the following consequences and then determine whether each is K-valid or not:

(a) $\Box p \land \Box q \models \emptyset \Rightarrow \Box(\Box p \land \Box q)$ (b) $\emptyset \models \Box p \land \Box q \Rightarrow \Box(\Box p \land \Box q)$

The problem is, if you fail to construct a derivation (see Example 6.23), then can you guarantee that the consequence does not hold? Definitely not; for, there may be another way of constructing a derivation. Let us see one more failed attempt.

***EXAMPLE* 6.26** Construct a tableau for $p \to \Box p \overset{w}{\models} q \to \Box q$.
Solution

$$\begin{array}{c}
1\ \neg(q \to \Box q)\\
1\ q\\
1\ \neg\Box q\\
1\ p \to \Box p\\
\begin{array}{cc}
1\ \neg p & 1\ \Box p\\
1.1\ \neg q & 1.1\ \neg q\\
 & 1.1\ p\\
 & 1.1\ p \to \Box p\\
 & \begin{array}{cc} 1.1\ \neg p & 1.1\ \Box p\\ \vdots & \vdots \end{array}
\end{array}
\end{array}$$

What do you see? The tableau does not close. Whatever way you expand it by reusing the global assumption (one such is done on the fourth line, and again on the eighth line), the tableau does not close. However, this does not guarantee that "the consequence does not hold". Even after the adequacy of tableau method (Prove it.) is proved, this cannot be guaranteed since we do not know how to ensure that no tableau will ever close in such a case. Do you see the difficulty?

Whenever you want to show the validity of a consequence, you can simply give a tableau derivation; it is conclusive. Of course, you must first prove that if there is a derivation for a consequence, then the consequence is K-valid; it is the soundness of the tableau method. On the other hand, if a consequence is K-valid, you can have a tableau derivation for it; this is the completeness of the tableau method. Now, suppose that a consequence is invalid, then how do you show its invalidity? Of course, by constructing a model which satisfies all the premises (global or local) and falsifies the conclusion. But a tableau can never show it. In such a case, whatever way you try to expand the tableau (with premises and the negation of the conclusion at the root), it will never close. At the same time, the tableau method does not show, in any way, that no tableau will close; it is of no use. In such a situation, you can exploit an open path of the tableau to construct a model that would satisfy the premises and falsify the conclusion. See, it is not a waste to attempt a derivation!

It is very much similar to the tableaux in PL. We want to construct a model that would satisfy all the mps in an open path. Take an open path in Example 6.26, say, the leftmost path of the tableau. The path is open and it contains the prefixed mps (from leaf to root):

$$1.1\ \neg q,\ 1\ \neg p,\ 1\ p \to \Box p,\ 1\ \neg\Box q,\ 1\ q,\ 1\ \neg(q \to \Box q)$$

Looking at the prefixes, it is enough to consider only two worlds named 1 and 1.1. As our motivation for prefixes says, the world 1.1 is accessible from the world 1. Since 1 q occurs in this path, we start asserting that $\boxed{1} \Vdash q$. Again, 1 $\neg p$ occurs in the path; so $\boxed{1} \nVdash p$. What about the other world, namely, 1.1? Since 1.1 $\neg q$ occurs in the path, we have $\boxed{1.1} \nVdash q$. These are the only literals occurring in the path. The literal p does not occur with prefix 1.1; though $\neg p$ occurs with prefix 1. Thus, we may fix either $\boxed{1.1} \Vdash p$ or $\boxed{1.1} \nVdash p$. Let us take the first alternative. We have the following model:

$$\begin{tikzcd} M: & \boxed{1}^{\Vdash q} \arrow[r] & \boxed{1.1}^{\Vdash p} \end{tikzcd}$$

In this model, obviously, $p \to \Box p$ is satisfied, as $\boxed{1} \Vdash p \to \Box p$ and $\boxed{1.1} \Vdash p \to \Box p$. Note that the last satisfaction relation holds since there is no world accessible from $\boxed{1.1}$. Now, $\boxed{1} \Vdash q$ but $\boxed{1} \nVdash \Box q$ as the world $\boxed{1.1}$ is accessible from $\boxed{1}$ though it does not satisfy q. Therefore, $\boxed{1} \nVdash q \to \Box q$. Consequently, $p \to \Box p \ \nvDash^{w}\ q \to \Box q$.

So, a model can be constructed from a failed attempt at proving a consequence, by taking hint from an open path. But any open path would not serve this purpose. This is so because, before sufficient expansion of the tableau (say, in the beginning), a path is always open. Recollect what you have done in the case of PT. You may need the notion of a finished tableau. Try it!

***Exercise* 6.13** Construct a model from an open path in the tableau for Example 6.23(b).

6.5 Other Modal Logics

Recall that in a scenario of a generalized modal consequence, the global assumptions restrict the view of models. The models to be considered must satisfy all the global assumptions. Then, we think about the worlds in each of these models and look for satisfying the local assumptions, restricting the worlds further. For the consequence to hold, these worlds must satisfy the conclusion. It does not really matter in whatever way you read the connectives. What happens if you read the connectives in a certain way? Say, you read $\Box$ as the modality, 'it is known that. . .is true'. Then certainly, we would admit that in such a world, where 'X is known to be true' the statement that 'X is true' holds. Essentially, we are admitting the truth of $\Box X \to X$ in all such worlds. In other words, if we read the connective $\Box$ as 'it is known that', then the mp $\Box X \to X$ becomes a global assumption.

It is rather an assumption scheme for, in such a case, $\Box Y \rightarrow Y$ is also one of the global assumptions. What does it mean semantically? If in a world w, the mp $\Box X$ is true, then in the same world w, the mp X is also true. Thus, the world w must be accessible from itself. So, what do you see? Such **assumption schemes** impose conditions on the accessibility relation. The class of frames, in turn, becomes restricted. We will have to consider various types of frames then. But remember that various types of frames correspond to various ways of reading the modal connectives of necessity ($\Box$) and possibility ($\Diamond$).

Since properties of frames come from the properties of the binary relation on the worlds, you must recollect the usual types of binary relations. If you have forgotten them (you are supposed not to), here is a small list. Let R be a binary relation on a set W, the set of worlds, for us. The relation R is called

reflexive :	if for every $w \in W$, wRw
symmetric :	if for every $u, v \in W$, uRv implies vru
transitive :	if for every $u, v, w \in W$, uRv and vRw implies uRw
an equivalence relation :	if R is reflexive, symmetric, and transitive
functional :	if for every $u \in W$, there is a unique $v \in W$ such that uRv
linear :	if for every $u, v, w \in W$, uRv and uRw implies either vRw or $v = w$ or wRv
serial :	if for every $u \in W$, there is $v \in W$ such that uRv
euclidian :	if for every $u, v, w \in W$, uRv and uRw implies vRw
total :	if for every $u, v \in W$, either uRv or vRu or both

Let W be any set (of worlds), and R be a binary relation on W having one or more of the above properties. Then, we say that the frame (W, R) has that property. Thus, a **reflexive frame** is a frame (W, R), where R is a reflexive relation. We want to know what kind of frames give rise to which assumption schemes. You have had an example already; reflexive frames give rise to the assumption scheme $\Box X \rightarrow X$.

Our plan is to restrict our frames to satisfy certain properties so that we will have different assumption schemes which can be used as axiom schemes (along with KC), thereby giving rise to different modal logics. Let us start with reflexivity. To be able to study this so-called **correspondence theory** of modal logics with ease, we will have a definition. Note that if you restrict all your frames to satisfy certain property, you may think of a sub-collection of frames (satisfying that property). That is, properties and collections of frames can be taken as one and the same.

Definition 6.10 Let $\mathcal{L}$ be a collection of frames and X be a modal proposition. If (W, R) is a frame in $\mathcal{L}$ and $\Vdash$ is a satisfaction relation specifying whether a modal proposition is satisfied (true) at a world, then the model $(W, R, \Vdash)$ is called a **model based on the frame** (W, R). The modal proposition X is called **valid in the frame** (W, R) if each model based on the frame (W, R) satisfies X. The modal proposition X is called **$\mathcal{L}$-valid** and written as $\models_{\mathcal{L}} X$ iff X is valid in each frame in $\mathcal{L}$.

When $\mathcal{L}$ is a collection of all frames (without any particular restriction or property), $\models_{\mathcal{L}} X$ coincides with $\models_K X$. Due to this coincidence, we will use the symbol K for the logic K and also for the collection of all frames.

Now, what about the reflexive frames? Denote by T, the set of all reflexive frames. As you have seen, we must add another axiom scheme, namely, $\Box X \to X$ to KC for capturing $\models_T$. Conversely, suppose that the mp $\Box X \to X$ is C-valid for a collection C of frames $(W, R, \Vdash)$. Then, in each world $w \in W$, we have $w \Vdash \Box X \to X$. This means that if $w \Vdash \Box X$, then we must also have $w \Vdash X$. But $w \Vdash \Box X$ means that in every world u with wRu, $u \Vdash X$. Thus, if in every world u with wRu, $u \Vdash X$, then $w \Vdash X$. This holds for every world $w \in W$. Hence R must be reflexive, i.e., $C = T$.

The following theorem (Prove it.) summarizes some important properties of frames in terms of additional axiom schemes to KC. Our naming system keeps the collection of frames and the name of the axiom as the same. For example, the modal logic with reflexive frames is denoted as T, the collection of all reflexive frames is also denoted as T. This is the way you must read the table as given in the following theorem.

Theorem 6.14 (Correspondence Theorem) *The following table summarizes the correspondence between a frame property and the axiom scheme to be added to KC.*

Name	*Axiom scheme*	*Frame property*
K	*none*	*none*
T	$\Box X \to X$	*reflexivity*
B	$X \to \Box\Diamond X$	*symmetry*
D	$\Box X \to \Diamond X$	*serial*
4	$\Box X \to \Box\Box X$	*transitivity*
5	$\Diamond X \to \Box\Diamond X$	*euclidian*
6	$\Box X \leftrightarrow \Diamond X$	*functional*
7	$\Box(X \wedge \Box X \to Y)$	
	$\vee \Box(Y \wedge \Box Y \to X)$	*linearity*

You may read the axioms T, B, D, L as 1, 2, 3, 8, respectively to go with other axioms. But these names are historical and we refer to them this way. Further, we will name other the logics by combining the axioms. For example, the logic KT4 is one with all the axioms and inference rules of KC along with the extra axiom schemes as Axiom T and Axiom 4. Then semantically it is the modal logic of all reflexive and transitive frames. To make a shorthand, you may also omit the K from KT4; that applies to all modal logics considered here. This is so because all the modal logics here are extensions of K. However, we will continue writing this K as a prefix with the names of other logics.

The logic KT is a sublogic of KT4 as each reflexive and transitive frame is vacuously reflexive. Notice the use of the word 'sublogic'. When we say KT is a sublogic of KT4 what we mean is, if a modal proposition X is KT-valid, then it is also KT4-valid, and every proof in KT is also a proof in KT4.

The logic KT4 is also written as **S4**, and the logic KT45 is written as **S5**. There is one more important modal logic which has not been considered in this naming scheme; it is the **Gödel logic G**. The logic G has the extra axiom (with KC):

(L) $\quad \Box(\Box A \to A) \to \Box A$

In our notation the logic G is simply the logic KL. Figure 6.4 shows which modal logic is a sublogic of another; only some important logics are covered here.

***Exercise* 6.14** Show that K4 is a sublogic of G.
[*Hint*: Decide whether you require the axiom 4 to have a proof in G or the axiom L to have a proof in K4.]

***Exercise* 6.15** What is the frame property corresponding to the axiom L of the Gödel logic G?

***Exercise* 6.16** Can you find how the logics K6 and K7 are related to other logics by the relation of 'sublogic'?

Various modal logics constructed in this section play, in an abstract way, with the axioms and the frame properties. Theorem 6.14 is the soundness and completeness of the various axiomatic systems obtained as extensions of KC. Do you see this? You can then have other proof procedures such as the natural deductions, Gentzen systems, and analytic tableau for these logics as well. The natural deduction system can be extended in a very natural way by adding a corresponding rule to the extra axioms to the system KND. Table 6.1 shows additional inference rules for various logics. Alternatively, you can get an inference rule corresponding to an axiom by keeping the axiom in the denominator with an empty numerator.

The notions of proof and theorem are kept as they were except that we admit the new inference rules in proofs. Similarly, theorems are not only

KND-theorems but to be prefixed with the respective logics. The same applies to the axiomatic systems also. See the following examples.

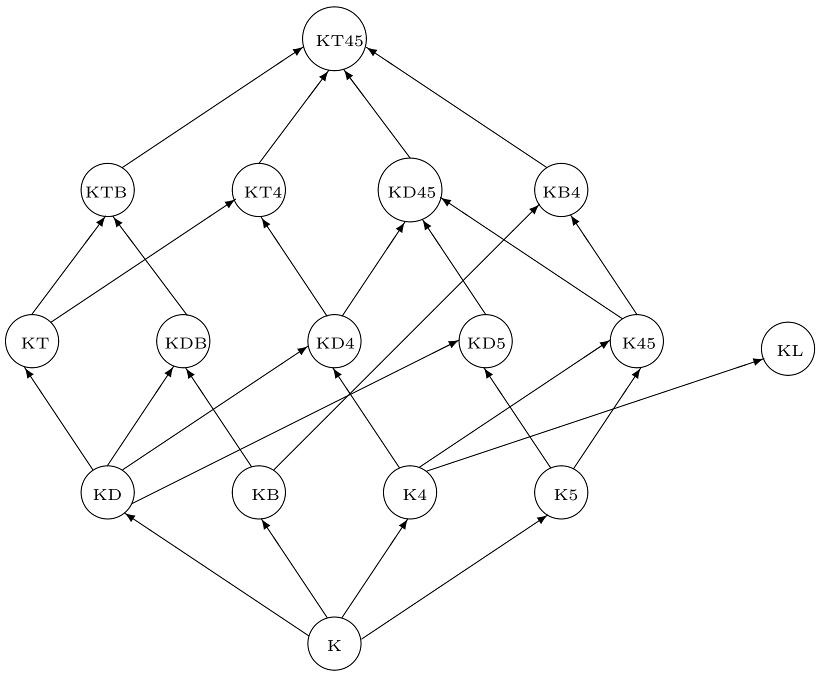


Figure 6.4 Modal logics.

Table 6.1 Natural Deduction: Additional Rules

System	Rule of Inference	System	Rule of Inference
KND	No extra rule	TND	$\dfrac{\Box X}{X}$
BND	$\dfrac{X}{\Box\Diamond X}$	DND	$\dfrac{\Box X}{\Diamond X}$
4ND	$\dfrac{\Box X}{\Box\Box X}$	5ND	$\dfrac{\Diamond X}{\Box\Diamond X}$
6ND	$\dfrac{\Box X}{\Diamond X}, \quad \dfrac{\Diamond X}{\Box X}$	7ND	$\dfrac{\neg\Box(X \wedge \Box X \to Y)}{\Box(Y \wedge \Box Y \to X)}$

***EXAMPLE* 6.27** Construct (a) K4ND and (b) K4C proofs to show that

$$\vdash \Box p \wedge \Box q \to \Box\Box p \wedge \Box\Box q$$

Solution (a) Here is a proof of $\vdash_{K4ND} \Box p \wedge \Box q \to \Box\Box p \wedge \Box\Box q$:

1. $\Box p \wedge \Box q$	CP
2. $\Box(\Box p \wedge \Box q)$	4ND
3. $\Box p \wedge \Box q$	$\Box e$
4. $\Box p$	$\wedge\ e$
5. $\Box\Box p$	$\Box i$
6. $\Box p \wedge \Box q$	1, $\Box e$
7. $\Box q$	$\wedge\ e$
8. $\Box\Box q$	$\Box i$
9. $\Box\Box p \wedge \Box\Box q$	5, 8, $\wedge i$
10. $(\Box p \wedge \Box q) \rightarrow \Box\Box p \wedge \Box\Box q$	$\rightarrow i$

(b) The following is a proof of $\vdash_{K4C} \Box p \wedge \Box q \rightarrow \Box\Box p \wedge \Box\Box q$:

1.	$\Box p \wedge \Box q \rightarrow \Box(\Box p \rightarrow \Box q)$	Axiom-4
2.	$\Box p \wedge \Box q \rightarrow \Box p$	PC
3.	$\Box(\Box p \wedge \Box q) \rightarrow \Box\Box p$	R
4.	$\Box p \wedge \Box q \rightarrow \Box q$	PC
5.	$\Box(\Box p \wedge \Box q) \rightarrow \Box\Box q$	R
6.	$\Box(\Box p \wedge \Box q) \rightarrow \Box\Box p \wedge \Box\Box q$	3, 5, PC
7.	$\Box p \wedge \Box q \rightarrow \Box\Box p \wedge \Box\Box q$	1, 6, HS

***EXAMPLE* 6.28** Show that in K45ND, $\vdash p \rightarrow \Box\Diamond p$.

Solution

1. p	CP
2. $\Box\neg p$	CP
3. $\neg p$	TND
4. $\bot$	1, 3, $\neg e$
5. $\neg\Box\neg p$	$\neg i$
6. $\Box\neg\Box\neg p$	5ND
7. $p \rightarrow \Box\neg\Box\neg p$	$\rightarrow i$
8. $p \rightarrow \Box\Diamond p$	Def. of $\Diamond$

***EXAMPLE* 6.29** Construct a KT5ND proof for $\vdash \Diamond p \rightarrow \Diamond\Box\Diamond p$.

Solution

1.	$\Diamond p$	CP
2.	$\neg\Diamond\Box\Diamond p$	CP
3.	$\neg\neg\Box\neg\,\Box\Diamond p$	Def. of $\Diamond$
4.	$\Box\neg\Box\Diamond p$	$\neg\neg e$
5.	$\neg\Box\Diamond p$	TND
6.	$\Box\Diamond p$	$1, 5\mathrm{ND}$
7.	$\bot$	$\neg e$
8.	$\Diamond\Box\Diamond p$	$\neg e$
9.	$\Diamond p \to \Diamond\Box\Diamond p$	Def. of $\Diamond$

Similarly, analytic tableau for the logic K is extended for other logics. The additional rules for expanding any path of a tableau are given below:

K : No Additional Rules

$$\mathrm{T}: \quad \frac{\sigma\ \Box X}{\sigma\ X} \qquad \frac{\sigma\ \neg\Diamond X}{\sigma\ \neg X}$$

$$\mathrm{B}: \quad \frac{\sigma.n\ \Box X}{\sigma\ X} \qquad \frac{\sigma.n\ \neg\Diamond X}{\sigma\ \neg X}$$

$$\mathrm{D}: \quad \frac{\sigma\ \Box X}{\sigma\ \Diamond X} \qquad \frac{\sigma\ \neg\Diamond X}{\sigma\ \neg\,\Box X}$$

$$4: \quad \frac{\sigma\ \Box X}{\sigma.n\ \Box X} \qquad \frac{\sigma\ \neg\Diamond X}{\sigma.n\ \neg\Diamond X}$$

$$5: \quad \frac{\sigma\ \Box X}{\sigma.n\ \Box X} \qquad \frac{\sigma\ \neg\Diamond X}{\sigma.n\ \neg\Diamond X} \qquad \frac{\sigma.n\ \Box X}{\sigma\ \Box X} \qquad \frac{\sigma.n\ \neg\Diamond X}{\sigma\ \neg\Diamond X}$$

$$6: \quad \frac{\sigma\ \Box X}{\sigma\ \Diamond X} \qquad \frac{\sigma\ \neg\Diamond X}{\sigma\ \neg\,\Box X} \qquad \frac{\sigma\ \Diamond X}{\sigma\ \Box X} \qquad \frac{\sigma\ \neg\,\Box X}{\sigma\ \neg\Diamond X}$$

$$7: \quad \frac{\sigma\ \neg\,\Box(X \wedge \Box X \to Y)}{\sigma\ \Box(Y \wedge \Box Y \to X)} \qquad \frac{\sigma\ \Diamond(X \wedge \Box X \wedge \neg Y)}{\sigma\ \neg\Diamond(Y \wedge \Box Y \wedge \neg X)}$$

Try to solve the following examples yourself before reading the solutions.

***EXAMPLE* 6.30** Show by analytic tableau that $\vdash_B \Diamond\Box X \to X$.

Solution In the following tableau, the rule B is applied on the last line.

$$
\begin{array}{c}
1\ \neg(\Diamond\Box X \to X)\\
1\ \Diamond\Box X\\
1\ \neg X\\
1.1\ \Box X\\
1\ X\\
\times
\end{array}
$$

***EXAMPLE* 6.31** Show that $\vdash_T (\Box(X \vee Y) \wedge \neg X) \to Y$.

Solution In the proof below, where is the rule T applied?

$$
\begin{array}{c}
1\ \neg(\Box(X \vee Y) \wedge \neg X) \to Y\\
1\ \Box(X \vee Y) \wedge \neg X\\
1\ \neg Y\\
1\ \Box(X \vee Y)\\
1\ \neg X\\
\begin{array}{cc}
1\ X \qquad & \qquad 1\ Y\\
\times \qquad & \qquad \times
\end{array}
\end{array}
$$

***EXAMPLE* 6.32** Show $\Box\Diamond(\Box X \to \Box\Diamond Y) \overset{w}{\models} (\Box X \to \Box\Diamond Y)$, using the analytic tableau in KT4.

Solution In the proof below, where is the rule KT4 used and why does the proof show the weak entailment?

$$
\begin{array}{c}
1\ \Box\Diamond(\Box X \to \Box\Diamond Y)\\
1\ \neg(\Box X \to \Box\Diamond Y)\\
1\ \Box X\\
1\ \neg\Box\Diamond Y\\
1.1\ \neg\Diamond Y\\
1.1\ \Box X\\
1.1\ \Diamond(\Box X \to \Box\Diamond Y)\\
1.1.1\ \Box X \to \Box\Diamond Y\\
\begin{array}{cc}
1.1.1\ \neg\Box X \qquad & \qquad 1.1.1\ \Box\Diamond Y\\
1.1.1\ \Box X \qquad & \qquad 1.1.1\ \Diamond Y\\
\times \qquad & \qquad 1.1.1\ \neg\Diamond Y\\
 & \times
\end{array}
\end{array}
$$

***EXAMPLE* 6.33** In KT45, show by analytic tableau that $\Diamond\Box X \overset{w}{\models} \Box X$.

Solution Find out where the rule KT45 is used. See why the prefix '1.2' is introduced. Can you close the tableau somewhere before the last line?

$$\begin{array}{c} 1\ \Diamond\Box X \\ 1\ \neg\Box X \\ 1.1\ \Box X \\ 1.2\ \neg X \\ 1\ \Box X \\ 1.2\ X \\ \times \end{array}$$

We have not proved the adequacy of analytic tableau or the natural deduction method. In the above examples, the adequacy results have been implicitly used. Try to prove them. Moreover, the strong adequacy of the tableau method follows when you have a generalized consequence with global and local assumptions; prove this result.

6.6 Various Modalities

The modal logics, as constructed in Section 6.5, are an interplay of axioms and frame properties. Do they actually play a role in representing various modalities? In this section, we will discuss some of the practical issues in representing modalities such as knowledge and belief. You have already seen some of these in Section 6.1. Modalities may express the necessity or the possibility of truth, convenience, lawfulness, certainty, agreement, acceptance, quotations, temporality, belief, contingency, knowledge, execution of programs, etc. Look at the emphasized phrases that express modalities in the following sentences:

It is *necessarily true that* moon is made of cheese.

It is *possible that* the morning star is the evening star.

It is *convenient to* have your residence near your workplace.

It is *unlawful for* Indians to smoke in a public place.

It is *certain that* Plato was a student of Socrates.

It is *doubtful* whether Descartes had doubt over everything.

It is *allowed by* the authorities to park your vehicles here.

It is *said by the ancients that* if you cannot decide now, then you can never decide.

Yesterday he was in a jolly mood.

Today morning he is depressed.

Sun will rise in the east *for all time to come.*

I *believe that* you will certainly like this book.

It is *a belief that* each algorithm is a Turing machine.

It is *a fact* that Turing was happier towards the end of his life.

I know it very well that Hausdorff committed suicide.

It is *common knowledge that* Bertrand Russell married thrice.

After the execution of the program P, the hard disk will burn.

All the emphasized phrases in the above sentences are modalities of some sort. Some of them are modalities of truth (**alethic**), some are **temporal** modalities, some are obligatory (**deontic**) modalities, some are modalities of knowledge and belief (**doxastic**), etc. Our plan is to symbolize such modalities by the two symbols $\Box$ and $\Diamond$. For example, we may translate the phrase 'it is necessarily true that p' as $\Box p$. Then, 'it is possibly true that' will be translated to $\Diamond p$. Similarly, we may translate 'I believe that p' to $\Box p$. But then we assume implicitly that both the modalities 'it is necessarily true that' and 'I believe that' do not occur in the same context. If they do occur in some context, then we may have to invent new symbols for representing them.

Note that different modalities may follow different logics; and on deciding which logic would be more suitable for arguing about a particular modality is always debatable. However, certain modalities are well understood and their governing principles are more or less clear.

In case we represent the modality 'it is necessarily true that p' as $\Box p$, we accept the mp $\Box p \to p$ to hold since it represents the acceptable assertion: "If p is necessarily true, then p is true". Thus, in the problem domain of necessarily true and possibly true, we will be translating our informal sentences to the modal propositions in the formal language of the logic K. But there can be various views. Suppose that the necessity here is interpreted as logical necessity. Then, if p is necessarily true, so is $\Box p$. This would force us to assume that in our problem domain, the mp $\Box p \to \Box\Box p$ must hold. But if we read $\Box p$ as the physical necessity, then $\Box p \to \Box\Box p$ would no more be valid since physical laws of the universe need not be physically necessary.

Further, if we read $\Box p$ as 'I believe that' then, $\Box p \to p$ need not be valid. This happens because, for example, even if I believe that $\mathcal{P} \neq \mathcal{NP}$, it need not be so. I may believe that ghosts exist, but they may not. Thus the logic of beliefs cannot have $\Box p \to p$ as a valid mp. Depending upon the problem domain, the meaning of the modal connectives $\Box$ and $\Diamond$ would change, and then you have to choose the appropriate logic. If none of the standard logics seem to be appropriate for some reason or the other, you may have to create new logics.

***Exercise* 6.17** For this exercise, use the following abbreviations for reading the table given below. If $\Box$ is interpreted as given in the leftmost column of the table, see that the corresponding modal propositions in the corresponding columns having entry 1 hold. Give reasons why the mps having corresponding entry 0 do not hold.

$A : \Box p \to p$	$B : \Box p \to \Box\Box p$
$C : \Diamond p \to \Box\Diamond p$	$D : \Diamond\top$
$E : \Box p \to \Diamond p$	$F : \Box p \vee \Box\neg p$
$G : \Box(p \to q) \wedge \Box p \to \Box q$	$H : \Diamond p \wedge \Diamond q \to \Diamond(p \wedge q)$

$\Box X$	A	B	C	D	E	F	G	H
It is necessarily true that X	1	1	1	1	1	0	1	0
It shall be the case that X	0	1	0	0	0	0	1	0
It should be the case that X	0	0	0	1	1	0	1	0
You believe that X	0	1	1	1	1	0	1	0
You know that X	1	1	1	1	1	0	1	0
After execution of the program P, X holds	0	0	0	0	0	0	1	0

In the following, you will see some of the modalities of Exercise 6.17 and their formalizations. Note that we may end up at a multimodal logic if various modalities do occur in the same context. But the new modal connectives will necessarily be similar to our basic ones, may be, with different names.

A Logic of Knowledge

Suppose that we want to build up a logic for handling knowledge of an agent A. We would translate the phrase 'A knows that p' to $\Box p$. Then, the dual modal operator $\Diamond$ is read as "it is possible for all that A knows that ..." If there is more than one agent, then we have to invent more symbols. The multimodal logic would then have at least two modal operators corresponding to each agent. In such cases, we use the small letter corresponding to an agent's name as the subscript of the modal operator. The scheme of symbolization is

$\Box_x p$: 'X knows that p', and

$\Diamond_x p$: 'it is possible for all that X knows that p'.

Then, $\Box_a$ and $\Box_b$ will be treated as two different symbols, one for the knowledge of the agent A, and the other for the knowledge of the agent B. Note that you can interpret $\Box_a$ also as "it follows from whatever A knows that" or as "A is entitled to know that", etc. What about the axioms of such a logic? The axioms of this logic will include all the tautologies of PC and three more:

(LK1) $\Box_x(p \to q) \to (\Box_x p \to \Box_x q)$

(LK2) $\Box_x p \to p$

(LK3) $\Box_x p \to \Box_x \Box_x p$

The rules of inference of the logic of knowledge are the modus pones and the rule of necessitation as in K.

If you read $\Box_x$ as $\Box$, then LK1 is simply the axiom K, LK2 is the axiom T, and LK3 is the axiom 4. Hence the logic you get is simply KT4 for each agent. In this multimodal logic of knowledge, the knower (the agent) is assumed to be logically omniscient, for, once he knows p, he also knows all the consequences of p. The axiom LK3 says that one cannot know p and yet fail to know that he knows it. This is why LK3 is called the axiom or the *principle of positive introspection.*

We may take another stand by replacing LK3 by LK4, as:

(LK4) $\neg \Box_x p \to \Box_x \neg \Box_x p$

This says that if one (the agent X) does not know p, then he knows that he does not know p, the *principle of negative introspection.* So you see, nothing is conclusive here; it all depends upon what you want to symbolize, and which properties you want to keep and which to discard. Suppose that you want to interpret the modal operators as in the following:

$\Box_x$: 'the agent X believes that p', and

$\Diamond_x$: 'it is compatible with whatever X believes that'.

Then, the same logic KT4 (for each agent) might work. However, beliefs are more problematic than knowledge. An agent is not required to be entirely consistent on the set of his beliefs. Try making a better logic than KT4 for understanding the beliefs!

A Temporal Logic

When there is a reference to time, we think of it as a linear extent, just as the real line, but with a varying reference point unlike a fixed origin. For example, when you tell me, "Yesterday, Sam was in a jolly mood", your reference is a time interval called 'today'. The sentence 'Sam is in a jolly mood' may be devoid of any temporal concern. To make temporality more effective you might fill the blank-modality with a time index. Thus you may consider the sentence:

Sam is in a jolly mood at time t.

In so doing, the sentence "Sam is in a jolly mood" is viewed no longer as a whole sentence, but just as a property of some time point, say, of t. You may universally or existentially quantify over this variable t to get a (whole) sentence:

For all time t, Sam is in a jolly mood.

For some time t, Sam is in a jolly mood.

Interpreting the temporality as a modal operator, the sentences above can be rewritten, respectively, as

$\Box$ (Sam is in a jolly mood)

$\Diamond$ (Sam is in a jolly mood)

This reading of $\Box$ and $\Diamond$ would naturally give way to the logic K, as you can see that the formulas $p \to \Diamond p$, $\Box p \to p$ are true in any situation, whereas $\Diamond p \to p$, $p \to \Box p$ are not.

If you want to operate with past and present as two connectives, instead of 'for all time' or 'for some time', then you would have to introduce two modal operators for past and future separately. Here is one such scheme:

Fp : It will sometime be the case that p.

Pp : It was sometimes the case that p.

Gp : It will always be the case that p.

Hp : It was always the case that p.

Here, Gp corresponds to the $\Box$-future, Hp to $\Box$-past, Fp to $\Diamond$-future, and Pp corresponds to $\Diamond$-past. This symbolization brings in a bimodal logic with two types of $\Box$'s and two types of $\Diamond$'s. However, there is a marked difference between this logic and the logic K. For example,

G(Sam is in a jolly mood)

is read as

Sam will always be in a jolly mood.

whereas the sentence "Sam is in a jolly mood" is interpreted as "Sam is in a jolly mood now". So that the mp

G(Sam is in a jolly mood) $\to$ (Sam is in a jolly mood)

is no longer valid. That is, in this logic, we cannot assert that $\Box p \to p$ holds. On the other hand, $Gp \to Fp$ (i.e., $\Box p \to \Diamond p$) holds. It says that if p will always be true then at some time p will be true. Similarly, since temporal order is transitive, $Gp \to GGp$ is also valid.

As another variant, if you take $\Box p \equiv p \wedge Gp$ and $\Diamond p \equiv p \vee Fp$, then $\Box p \to p$ holds. However, $\Diamond p \to \Box\Diamond p$ does not hold. For example, take p as the sentence "Scientists are able to discover a cure for cancer". You can imagine a world where, in some future time, p will hold, i.e., $\Diamond p$ holds (now). But, in no future time to that time the cure can again be discovered since it would have been discovered by then. That is, $\Box\Diamond p$ does not hold.

In the next section, we will describe another temporal logic which helps in verifying the properties of a real system.

6.7 Computation Tree Logic

Consider an elevator operating in a multistorey building. We want to develop a formal language to express various states and functionality of the elevator. For example, consider representing, in our language:

> An upward going elevator at the third floor keeps on going upward when it has some passengers who wish to go to the fourth floor.

Of course, you can simply take the whole sentence as an atomic proposition in PL. Or, you may symbolize in FL by identifying the predicates. But that would not help much if you want to verify this property of the elevator after designing its switching circuit.

As a preliminary to symbolization, let us have the integer variables $floor$ and $direction$. Now, $floor = 2$ means that the elevator is on the second floor, and $direction = 1$ means that it is going upward. Similarly, $direction = -1$ tells that its movement is downward, and $direction = 0$ shows that it is idle. The phrase 'it has a passenger wishing to go to fifth floor' would mean that someone has pressed the No. 5 button. That is, we introduce another integer variable, *buttonpressed* so that this state is represented as $buttonpressed = 5$. Then, our sentence (quoted above) will be translated as

> for all possible states of the elevator starting from any state,
> $((floor = 2 \wedge direction = 1 \wedge buttonpressed = 5) \rightarrow$
> (for all possible start states, $(direction = 1$ until $floor = 4)))$

The temporality in the above 'until' cannot be omitted here. In FL, you would have translated 'until' to the connective $\vee$; in this case, it is insufficient. So, we must have a way in working with time. Here, it is enough to think of time as a discrete linearly ordered set just as the set of integers or natural numbers. (Will a finite segment of natural numbers suffice?)

Sometimes, we need to consider branching time. For example, consider a computing system where many processors are run in parallel and they might request to use certain other processes time to time. Here, once a job is submitted, it is distributed by a scheduler; the fragments, so to say, are to be worked out by many processes. Process 1 is doing its part and for it, time goes linearly in a discrete way. For Process 2, again, time goes linearly in a discrete way. For the scheduler, time goes in a branching way, looking at the different processes and their requests. In such a situation, suppose that a process has started but it is not yet ready. Then, we have a property to be verified, namely,

> It is possible to get a state where *started* holds but *ready* does not.

Assume that *started* and *ready* are the propositions with their obvious meanings. Then the above sentence will be formalized as

> It is possible for a state that $(started \wedge \neg ready)$

The sentence "for each state, if a request occurs, then it will eventually be acknowledged" will similarly be translated to "for all possible states starting with any state, ($requested \rightarrow$ for any starting state there will be a future state where ($acknowledged$))". To have a symbolic way of writing such sentences, we devise a scheme. We will use the mnemonics:

A :	for all possible states	E :	for some possible states
F :	for some future state	G :	for all future states
X :	for the next state	U :	for expressing 'until'

Then our connectives are:

$$AX,\ EX,\ AF,\ EF,\ AG,\ EG,\ A[\cdot\, U\, \cdot],\ E[\cdot\, U\, \cdot]$$

***Exercise* 6.18** Plugging in the meanings of the individual letters, write in words what the above connectives mean.

For propositions p, q, these connectives are used as follows:

$$AXp,\ EXp,\ AFp,\ EFp,\ AGp,\ EGp,\ A[p\,U\,q],\ E[p\,U\,q]$$

We use the square brackets in the last two connectives, only as a convention. It has been followed in a tool called SMV model checker, which uses this logic for verifying the real systems.

Formally, the language of **computation tree logic**, or **CTL**, for short, has the (well formed) formulas defined by the following grammar:

$$w ::= \top \mid \bot \mid p \mid \neg w \mid (w \wedge w) \mid (w \vee w) \mid (w \rightarrow w) \mid AXw \mid EXw \mid AGw \mid$$
$$EGw \mid AFw \mid EFw \mid E[w\,U\,w] \mid A[w\,U\,w]$$

where p is any atomic formula (atomic proposition in PL). As you have sufficient experience in tackling syntax, we will not go to parse trees and their uniqueness; you can prove it as well. Here are some examples of CTL-formulas:

$$\neg EFA[p\,U\,q],\ AF(p \rightarrow EGq),\ E[p\,U\,A[q\,U\,r]],$$
$$(EF(E[p\,U\,((p \wedge q) \wedge A[q\,U\,\neg p])]) \rightarrow AG(p \vee q))$$

whereas the following are not CTL-formulas (Why?):

$$AF(p\,U\,q),\ A\neg G(p \wedge q),\ EG[(p\,U\,q) \rightarrow (p\,U\,r)]$$

So, be careful with brackets larking around U. You can define subformulas etc., as usual. Now, what is the semantics of CTL? As is done for any modal logic, CTL, a temporal logic will also have a possible world semantics.

A model for CTL is a triple $M = (W, R, \Vdash)$, where W is a set of worlds, a set of possible states of computation, R is the accessible relation over W, and $\Vdash$ is the satisfaction relation giving details of which world satisfies which atomic formulas. A model is represented as a directed graph with nodes as the states annotated with the propositions (atomic formulas) which are satisfied there, and the accessibility relation is represented as the set of edges between the states. Whenever uRv holds, the graph has an edge $\boxed{\text{u}}\longrightarrow\boxed{\text{v}}$, as earlier. This edge may be interpreted as "during computation, there might be a transition from the state u to the state v". That is, v is the next immediate state to u. Given a CTL formula p, we write $M \Vdash p$ whenever $s \Vdash p$ holds for any state $s \in W$. The relation $s \Vdash p$ (shorthand for $M, s \Vdash p$) extends the definition of the given relation $\Vdash$ from atomic formulas or propositions to all formulas by the following rules:

1. $s \Vdash \top$, $s \nVdash \bot$, for all $s \in W$.
2. for any $s \in W$, $s \Vdash p$, for atomic propositions p as it has been mentioned in the model.
3. $s \Vdash \neg q$ iff $s \nVdash q$, for any $s \in W$.
4. $s \Vdash q \wedge r$ iff $s \Vdash q$ and $s \Vdash r$, for any $s \in W$.
5. $s \Vdash q \vee r$ iff $s \Vdash q$ or $s \Vdash r$, for any $s \in W$.
6. $s \Vdash q \to r$ iff $s \nVdash q$ or $s \Vdash r$, for any $s \in W$.
7. $s \Vdash AXq$ iff for all states s' with $s \longrightarrow s'$, $s' \Vdash q$.
   (AX stands for 'in every next state')
8. $s \Vdash EXq$ iff for some state s' with $s \longrightarrow s'$, $s' \Vdash q$.
   (EX stands for 'in some next state')
9. $s \Vdash AFq$ iff for all paths of the form $s_1 \longrightarrow s_2 \longrightarrow s_3 \longrightarrow \ldots$, with $s_1 = s$, there is an s_i such that $s_i \Vdash q$.
   (AF stands for "for all computation paths, beginning with s, there will be some future state such that")
10. $s \Vdash EFq$ iff for some path of the form $s_1 \longrightarrow s_2 \longrightarrow s_3 \longrightarrow \ldots$, with $s_1 = s$, there is an s_i such that $s_i \Vdash q$.
    (EF stands for "for some computation path, beginning with s, there will be some future state such that")
11. $s \Vdash AGq$ iff for all paths of the form $s_1 \longrightarrow s_2 \longrightarrow s_3 \longrightarrow \ldots$ with $s_1 = s$, and all s_i along the path, $s_i \Vdash q$.
    (AG stands for "for all computation paths, beginning with s, and for all future states")
12. $s \Vdash EGq$ iff for some path of the form $s_1 \longrightarrow s_2 \longrightarrow s_3 \longrightarrow \ldots$, with $s_1 = s$, and all s_i along such a path, $s_i \Vdash q$.
    (EG stands for "for all computation paths, beginning with s, there will be some future state along such a path such that")
13. $s \Vdash A[q\,U\,r]$ iff for all paths of the form $s_1 \longrightarrow s_2 \longrightarrow s_3 \longrightarrow \ldots$, with $s_1 = s$, there is an s_i along the path such that $s_i \Vdash r$, and for each $j < i$, $s_j \Vdash q$. ($A[\cdot\,U\,\cdot]$ stands for "all computation paths, beginning with s, satisfy q until r")
14. $s \Vdash E[q\,U\,r]$ iff for some path of the form $s_1 \longrightarrow s_2 \longrightarrow s_3 \longrightarrow \ldots$, with $s_1 = s$, there is an s_i along the path such that $s_i \Vdash r$, and for each $j < i$, $s_j \Vdash q$. ($E[\cdot\,U\,\cdot]$ stands for "some computation path, beginning with s, satisfies q until r")

Note that in this semantics, the future of a state includes the state itself. The computation paths referred to above is obtained from the (transition) graph of a model by unwinding the graph into infinite trees. For example, if a model has the graph (omitting $\Vdash$ relation):

$$\begin{tikzcd}
& \boxed{s_2} \arrow[dr] & \\
\boxed{s_1} \arrow[ur] \arrow[rr] & & \boxed{s_3} \arrow[loop below]
\end{tikzcd}$$

Here, a computation path beginning with s_1 is

$$s_1 \longrightarrow s_2 \longrightarrow s_3 \longrightarrow s_3 \longrightarrow s_3 \longrightarrow \ldots$$

Another computation path beginning with s_1 is

$$s_1 \longrightarrow s_3 \longrightarrow s_3 \longrightarrow s_3 \longrightarrow s_3 \longrightarrow \ldots$$

To check whether in a model, a state s satisfies a CTL formula, the computation tree is constructed. Since this tree contains all computation paths beginning with s, one has to check whether $\Vdash$ for the formula holds on these paths. The tree of all computation paths beginning with s_1 is given in Figure 6.5.

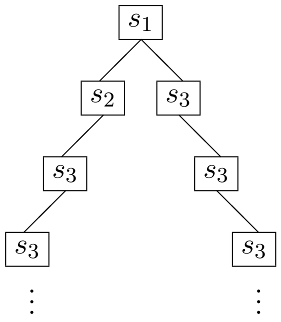


Figure 6.5 Computation tree.

***EXAMPLE* 6.34** Let M be the model as given by the following graph. Let Y be the CTL formula $[EG(r \vee q)\, U\, AFt] \rightarrow AXr$. Determine whether $s_1 \Vdash Y$ and/or $s_3 \Vdash Y$.

$$\begin{tikzcd}
& \Vdash p, q & & \Vdash q, r \\
& \boxed{s_1} \arrow[rr, shift left] \arrow[d] & & \boxed{s_2} \arrow[ll, shift left] \\
M : & & & \\
& \boxed{s_3} \arrow[rr] & & \boxed{s_4} \arrow[uu] \\
& \Vdash r & & \Vdash p, t
\end{tikzcd}$$

Solution The computation tree beginning with s_1 is given in Figure 6.6.

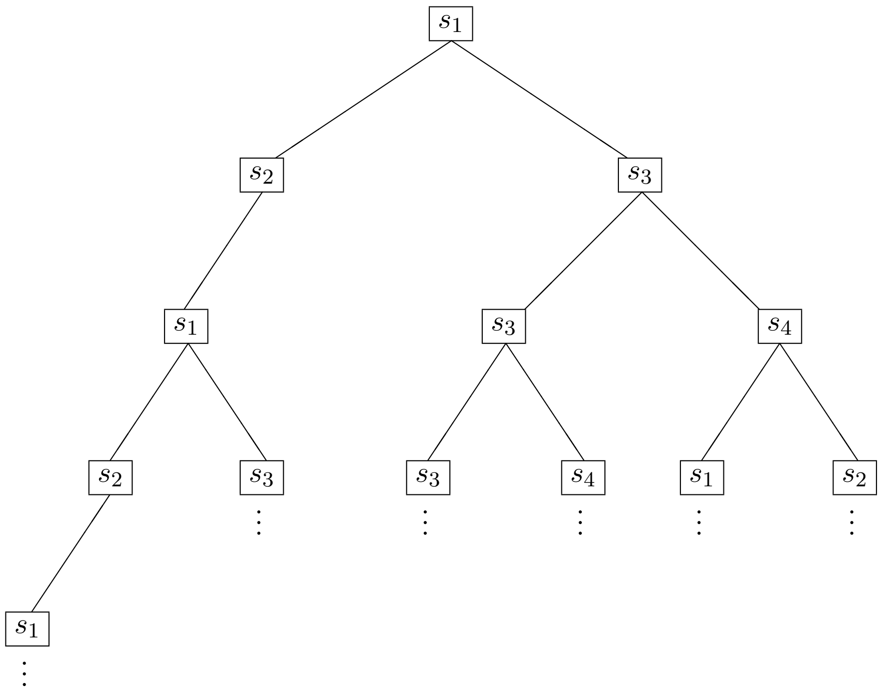


Figure 6.6 Computation tree for s_1.

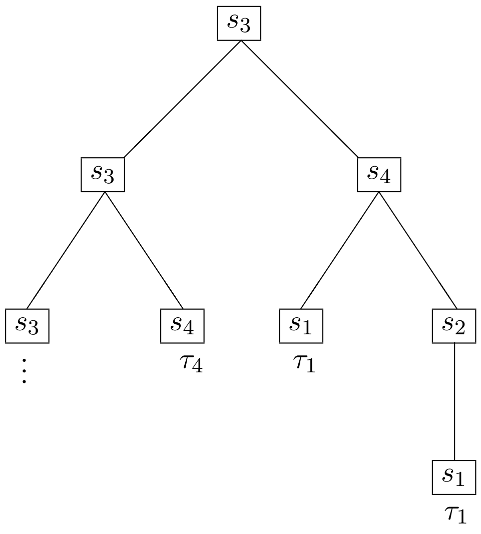


Figure 6.7 Computation tree for s_3.

[In Figure 6.7, τ_4 means the tree of s_4 repeated thereafter. Similarly, τ_1 is the tree for s_1 to be appended there.]

For Y to be true at the state s_1, we see that

$$s_1 \Vdash [EG(r \vee q)\, U\, AFt] \to AXr \text{ iff } s_1 \nVdash [EG(r \vee q)\, U\, AFt] \text{ or } s_1 \Vdash AXr$$

We take the simpler case first. $s_1 \Vdash AXr$ iff for every next state s to s_1, $s \Vdash r$. The 'next states' of s_1 are s_2 and s_3. As $s_2 \Vdash r$ and $s_3 \Vdash r$, we conclude that $s_1 \Vdash AXr$. Therefore, $s_1 \Vdash [EG(r \vee q)\, U\, AFt] \to AXr$.

For $s_3 \Vdash Y$, we consider the next states of s_3; they are s_4 and s_3. The computation tree with all paths beginning with s_3 is given in Figure 6.7. We see that $s_3 \Vdash r$ but $s_4 \nVdash r$. Hence $s_3 \nVdash AXr$. We must check the other alternative, i.e., whether $s_3 \nVdash [EG(r \vee q)\, U\, AFt]$. This happens if there is a computation path beginning with s_3 which falsifies "$EG(r \vee q)$ until AFt".

Let us take the leftmost path in the computation tree of Figure 6.7; which is $s_3 \longrightarrow s_3 \longrightarrow \cdots$. We want to check whether this path falsifies the sentence "$EG(r \vee q)$ until AFt". This means that either $s_3 \nVdash EG(r \vee q)$ or $s_3 \nVdash AFt$. (Why?) Check the simpler one first. $s_3 \Vdash AFt$ iff for all computation paths beginning with s_3, there is a future state satisfying t. From the computation tree of s_3, we see that s_1 is a future state and $s_1 \nVdash t$. Therefore, $s_3 \nVdash AFt$. Consequently, $s_3 \nVdash A[EG(r \vee q)\, U\, AFt]$. Thus, $s_3 \Vdash Y$.

***Exercise* 6.19** Check the following with the model M as given below.

(a) $s_1 \Vdash EX(q \wedge r)$ (b) $s_1 \Vdash \neg AX(q \wedge r)$ (c) $s_1 \Vdash EF(p \wedge r)$

(d) $s_2 \Vdash EGr \wedge AGr$ (e) $s_1 \Vdash AFr \vee A[p\, U\, r]$ (f) $s_1 \Vdash E[(p \wedge q)\, U\, r]$

$$M: \quad \begin{tikzcd} & \boxed{s_2}\ \Vdash q, r \arrow[dl, shift right] \arrow[dr] & \\ \boxed{s_1} \arrow[ur, shift right] \arrow[rr] & & \boxed{s_3}\ \Vdash r \\ \Vdash p, q & & \end{tikzcd}$$

***Exercise* 6.20** Define equivalence of two CTL formulas by $q \equiv r$ iff any state in any model which satisfies q also satisfies r and vice versa. Then show that

(a) $AF\neg t \equiv \neg EGt$ (b) $EF\neg t \equiv \neg AGt$ (c) $AX\neg t \equiv \neg EXt$

(d) $AFt \equiv A[\top U\, t]$ (e) $EFt \equiv E[\top U\, t]$ (f) Equivalences of PL.

In fact, any model of CTL is an abstraction of any transition system such as concurrent and reactive systems, networks, etc. We assume that you know how to represent such a system with a graph. Your aim is to verify whether such a system satisfies certain properties. These properties are now represented as CTL formulas, by looking at the satisfaction (truth) at each state (world) of the model. You have seen in Example 6.19 how cumbersome and lengthy it is to check this satisfaction relation. Once we have a finite model, programs can be written to do this repetitive job.

These programs are called **model checkers**. SMV is one such model checker. There are also other model checkers using specification language as CTL. Note that it all depends upon how you visualize time and the reactions of the system components. In such a scenario of model checking, you have a real system for which some properties are to be verified. Then, with the model checker, you will be doing the following:

- Model the real system using the description language (such as SMV) of the model checker arriving at an abstract model M.
- Translate the properties of the real system which are to be verified, to the specification language (such as CTL) of the model checker and arrive at a formula p.
- Run the model checker with inputs M and p to determine whether $M \models p$.

It is of much practical importance to develop model checkers which use very general specification languages so that interesting properties of real systems can be checked efficiently.

SUMMARY

In this chapter, you have learnt how to handle various modalities. The modalities might be pertaining to truth, knowledge, belief, or even behaviour of real systems. We have taken a simplistic approach of introducing new modal connectives (of necessity, $\Box$, and possibility, $\Diamond$) and arguing about the minimum requirements they should satisfy. This approach has led us to the logic K, the basic modal logic.

The semantics of K has taken us to the relational structures called *frames*. A frame consists of a set, called the set of *worlds*, and a relation on this set, called the *accessibility relation*. Each world is assumed to be propositional in the sense that it is an interpretation of PL. A world thus comes with a satisfaction relation; it is loaded with the information as to which propositions are true in it and which are false. For a unified notation, we have denoted the satisfaction relations of all the worlds with a single symbol, $\Vdash$. The frame with this satisfaction relation (or truth) prescribed for each world (possibly differently) is called a model. The models serve the same job as interpretations in PL. The truth in each world is taken into consideration in defining satisfaction of modal propositions in models, and consequently, the notion of validity is introduced.

Trying for an axiomatization of the basic logic K has led us to considering various properties and other logical extensions. We have proved that the axiomatic system KC is adequate to the logic K. The extensions of KC obtained by addition of different axiom schemes have resulted in restricting frames to satisfy certain properties. Studying the interplay of addition of axioms and frame properties, you have discovered the correspondence theorem and thereby many different modal logics. The modal logics so obtained as modifications of the basic logic K have been found to be interesting on their own right, addressing different problem domains. You have also seen how natural deduction and the method of analytic tableau could be extended to these logics.

As an application to real systems, you have learnt the computation tree logic which uses the idea of branching time. To understand various phenomena in computer science, you will naturally use logical models which would differ quite a bit from the ones discussed here, though this will form a basis for your future work. The following remarks about resources will be of some help.

Modal logic is bread and butter for a computer scientist. Once it mixes into the blood stream, it is impossible to find the source. However, one must start somewhere, and this text is meant for that. This is a mere introduction; so pointers must be given for pursuing the topics further. The first text to discuss modal logic was by C. I. Lewis [45, 46]. You may start with the texts [18, 55, 74]. In [74], you will also get another proof procedure, called cancellation technique, a shortened form of Gentzen systems. For a comprehensive introduction to modal logics, you may like the incomplete *Lemmon notes* [44]. The semantics, called the **possible world** semantics was invented by S. Kripke; thus the name K for the basic modal logic. Some good texts exclusively meant for modal logics are [7, 32, 40]. These books discuss adequacy of calculi and decidability of the modal logics via *finite models property* in great detail. You will also find the references there to be very resourceful. The natural deduction system and the tableau, as presented here, have their origin in [21, 23]. Whatever discussed here is only an introduction to the propositional modal logic. For a modal extension of the first order logic, see [23]. For applications of modal logics for reasoning about knowledge (originated by J. Hintikka) two good sources are [20, 53]. The computation tree logic as presented here is based on the chapter on Verification by Model Checking in [41]. You will also find discussions of two nice puzzles, the wise men puzzle and the muddy children puzzle in [41]. This text has more information on the SMV (symbolic model verifier) than given here. You may go on a search for materials after trying the following problems.

PROBLEMS

1. Determine whether the following are valid in K. If one is not valid, then in which modal logic is it valid?

(a) $A \to \Box A$ (b) $\Box A \to A$ (c) $A \to \Diamond A$ (d) $\Diamond A \to A$

(e) $\Box A \to \Diamond A$ (f) $\Diamond A \to \Box A$ (g) $\Box\Diamond A \to \Diamond A$ (h) $\Diamond A \to \Box\Diamond A$

(i) $\Box\Box A \to \Box A$ (j) $\Diamond\Diamond A \to \Diamond A$ (k) $\Diamond(A \wedge B) \to \Diamond A \wedge \Diamond B$

2. Suppose that in a model, no world is accessible from the world u and the world v is not accessible from any world. Let A be any proposition (not an mp) and B be any mp. Use the symbol $\Vdash$ for the satisfaction at a world. What about the truth of the following?

(a) $u \Vdash A$	(b) $u \Vdash \Box A$	(c) $u \Vdash \Diamond A$	(d) $u \Vdash B$
(e) $u \Vdash \Box B$	(f) $u \Vdash \Diamond B$	(g) $v \Vdash A$	(h) $v \Vdash \Box A$
(i) $v \Vdash \Diamond A$	(j) $v \Vdash B$	(k) $v \Vdash \Box B$	(l) $v \Vdash \Diamond B$

3. Consider the following models $\mathcal{M}_i = (W_i, R_i, \Vdash)$ given in (a)−(e) and the mps in (f)−(k). Determine which models satisfy which mps.

(a) $W_1 = \{u_1, u_2, u_3\}$, $R_1 = \{(u_1, u_2), (u_1, u_3)\}$, $u_2 \Vdash p, u_3 \Vdash q$

(b) $W_2 = \{v_1, v_2, v_3\}$, $R_2 = \{(v_1, v_2), (v_2, v_3)\}$, $v_2 \Vdash p$

(c) $W_3 = \{w_1, w_2\}$, $R_3 = \{(w_1, w_2)\}$, $w_2 \Vdash p$

(d) $W_4 = \{x_1, x_2\}$, $R_4 = \{(x_1, x_2), (x_2, x_2)\}$, $x_2 \Vdash p$

(e) $W_5 = \{y_1, y_2, y_3, y_4, y_5\}$, $R_5 = \{(y_1, y_2), (y_1, y_3), (y_2, y_4), (y_3, y_5)\}$, $y_4 \Vdash p$, $y_5 \Vdash q$.

(f) $\Box(p \vee q) \to (\Box p \vee \Box q)$

(g) $\Box p \to \Box\Box p$

(h) $\Diamond p \to \Box\Diamond p$

(i) $(\Diamond\Box p \wedge \Diamond\Box q) \to \Diamond\Box(p \wedge q)$

(j) $\Box\Box \cdots \Box p \to p$; consider $1, 2, 3, \ldots$ times $\Box$ in succession.

(k) $(\Diamond p \wedge \Diamond q) \to \Diamond(p \wedge q)$

4. Determine whether the following metastatements about K hold:

(a) $\models \Box(A \to B) \to (\Box A \to \Box B)$	(b) $\models \Box(A \wedge B) \leftrightarrow \Box A \wedge \Box B$
(c) $\models (\Box A \to \Box B) \to \Box(A \to B)$	(d) $\Box(A \to B) \equiv (\Diamond A \to \Diamond B)$
(e) $\Box(A \wedge B) \equiv (\Box A \wedge \Box B)$	(f) $\Box(A \vee B) \equiv (\Box A \vee \Box B)$
(g) $\Diamond(A \wedge B) \equiv (\Diamond A \wedge \Diamond B)$	(h) $\Diamond(A \vee B) \equiv (\Diamond A \vee \Diamond B)$
(i) $\Diamond(A \to B) \to (\Diamond A \to \Diamond B)$	(j) $(\Diamond A \to \Diamond B) \to \Diamond(A \to B)$

(k) If $\models A \leftrightarrow B$, then $\models \Box A \leftrightarrow \Box B$.

5. Let $\mathcal{M} = (W, R, \Vdash)$ be a model and $w \in W$ be a world. Define the connective $\twoheadrightarrow$ by "$w \Vdash (A \twoheadrightarrow B)$ iff for all $z \in W$, with wRz, $z \Vdash A \to B$." Is it true that $(A \twoheadrightarrow B) \equiv \Box(A \to B)$? Determine whether the following hold in K:

(a) $\Box A \equiv \neg A \twoheadrightarrow A$	(b) $\Box A \equiv (A \twoheadrightarrow A) \twoheadrightarrow A$
(c) $\Diamond A \equiv \neg(A \to \neg A)$	(d) $\models B \twoheadrightarrow (A \twoheadrightarrow B)$
(e) $\models A \twoheadrightarrow (B \twoheadrightarrow B)$	(f) $\models (A \wedge \neg A) \twoheadrightarrow B$
(g) $\models \neg\neg A \twoheadrightarrow A$	(h) $\models ((A \twoheadrightarrow B) \twoheadrightarrow A) \twoheadrightarrow A$

(i) $\models (\neg A \twoheadrightarrow A) \rightarrow A$ (j) $\models (A \twoheadrightarrow B) \rightarrow (\neg B \twoheadrightarrow \neg A)$

(k) $((A \twoheadrightarrow A) \twoheadrightarrow A) \equiv (\neg A \twoheadrightarrow A)$

6. Prove Theorem 6.14.

7. Prove that the modal proposition $\Box A \rightarrow \Diamond A$ is valid in all serial models (frames). What about the converse?

8. Prove that a frame is transitive iff every mp of the form $\Box p \rightarrow \Box\Box p$ is valid in it.

9. The modal logic **S4.3** is characterized by the reflexive transitive and linear frames. Show the following in the logic S4.3:

(a) $\Box(\Box A \rightarrow \Box B) \vee \Box(\Box B \rightarrow \Box A)$ is valid.

(b) $A \rightarrow \Box\Diamond A$ is invalid. [Construct such a frame to falsify it.]

(c) $\Diamond\Box(A \rightarrow B) \rightarrow (\Diamond\Box A \rightarrow \Diamond\Box B)$ is valid.

10. Prove the deduction theorems in the logics KT4 and KT45: $\Sigma \cup \{A\} \overset{w}{\models} B$ iff $\Sigma \overset{w}{\models} \Box A \rightarrow \Box B$. What about other modal logics?

11. How do you formulate the a syntactic counterpart of the the metastatement "If $\mathcal{M} \models \Sigma$, then $\mathcal{M} \models A$" in KC?

12. Consider the modal logics K, D, T, B, K4, KT4, and KT5. Denote the phrase "L_1 is a sublogic of L_2" by $L_1 \subset_l L_2$. You know from Figure 6.4 that $\text{K} \subset_l \text{D} \subset_l \text{T} \subset_l \text{B} \subset_l \text{KT45}$, $\text{K} \subset_l \text{K4} \subset_l \text{KT4} \subset_l \text{KT45}$, and $\text{T} \subset_l \text{KT4}$. Assuming transitivity of the relation $\subset_l$ implicitly, show that these are the only way $\subset_l$ relation works on these logics.

13. Show in K that the consequence $\Box A \rightarrow A \models \emptyset \Rightarrow \Box\Box A \rightarrow A$ holds but $\emptyset \models \Box A \rightarrow A \Rightarrow \Box\Box A \rightarrow A$ does not hold.

14. In the temporal logic of Section 6.6, consider the readings 1, 2, 3 of the necessity ($\Box$) and possibility ($\Diamond$) and then determine the validity status of the mps (a)$-$(l) in the different readings.

Reading 1: $\Box A$ as GA and $\Diamond A$ as FA

Reading 2: $\Box A$ as HA and $\Diamond A$ as PA

Reading 3: $\Box A$ as $A \wedge FA$ and $\Diamond A$ as $A \vee GA$

(a) $A \rightarrow \Diamond A$	(b) $\Diamond A \rightarrow A$	(c) $\Box A \rightarrow A$	(d) $A \rightarrow \Box A$
(e) $\Box A \rightarrow \Diamond A$	(f) $\Diamond A \rightarrow \Box A$	(g) $A \rightarrow \Box\Box A$	(h) $A \rightarrow \Box\Diamond A$
(i) $A \rightarrow \Diamond\Box A$	(j) $A \rightarrow \Diamond\Diamond A$	(k) $\Box A \rightarrow \Box\Box A$	(l) $\Box A \rightarrow \Box\Diamond A$

15. Show that in the temporal logic of Section 6.6, with the Reading 3 (Problem 13 above) the assumptions $PA \rightarrow \Box PA$, $\neg A \wedge \neg FA$ with a new rule of inference: "From $A \rightarrow B$ derive $\Diamond A \rightarrow \Diamond B$", it follows that $\neg\Diamond A$. Further, with discrete time, see that this is simply the *Diodorian argument*:

"Everything that is past and true is necessary. The impossible does not follow from the possible. Therefore, nothing is possible which neither is nor will be true."

16. In the logic D, show that $\Box A \to A$ is not valid. The reading of $\Box$ here is of *obligation.* It says that even if an action is obligatory, it does not mean that it is performed. Thus, D stands for deontic logic, the logic of action. Show that in D, $\models \Box A \to \Diamond A$ but $\not\models \Box A \to A$.

17. Show in T that $\models \Box(A \vee \neg A$. However, if $\Box A \vee \neg\Box A$ is assumed for every A, then $\Box A \equiv A$. What if the logic K is used instead of T?

18. Show in K that an mp A is valid iff $\Box A$ is valid.
[*Hint*: Use the tableau method.]

19. Discuss the differences (in any modal logic) between the following:

(a) $A \to \Box A$ (b) $A \equiv \Box A$ (c) $\models A \leftrightarrow \Box A$
(d) $\models A$ iff $\models \Box A$ (e) If $\models A$, then $\models \Box A$

20. Give KND and tableau proofs (in K) of the following mps:

(a) $(\Box A \wedge \Box B) \to \Box(A \wedge B)$ (b) $\Box(A \to B) \to \Box A \wedge \Box B$
(c) $(\Box A \vee \Box B) \to \Box(A \wedge B)$ (d) $(\Box A \wedge \Diamond B) \to \Diamond(A \wedge B)$
(e) $(\Box\Box A \wedge \Box\Diamond B) \to \Box\Diamond(A \wedge B)$

21. Attempt a tableau proof of $\Box(A \vee B) \to \Box A \vee \Box B$, in K. Explain why it fails. Why do you think this mp should not be valid?

22. Show by using analytic tableaux in K that "if $\vdash \Box A$, then $\vdash A$".

23. Attempt analytic tableau proofs and natural deduction proofs of the following mps in the logics K, T, B, D, K4, KT4, KT45 and determine where you succeed.

(a) $\Diamond\Box A \to A$ (b) $\Box(A \vee B) \wedge \neg X \to B$
(c) $\Diamond\Box A \to \Box A$ (d) $\Box\Diamond(\Box A \to \Box\Diamond B) \to (\Box A \to \Box\Diamond B)$
(e) $\Box A \vee \Box\neg\Box A$ (g) $(\Box A \wedge \Box B) \to \Box(\Box A \wedge \Box B)$
(f) $\Diamond(A \to \Box A)$ (h) $(\Box\Diamond A \wedge \Box\Diamond B) \to \Box\Diamond(\Box\Diamond A \wedge \Box\Diamond B)$
(i) $\Box A \vee \Box(\Box A \to B)$ (j) $\Box(\Diamond A \to A) \to \Box(\Diamond\neg A \to \neg A)$

24. Construct appropriate models from the failed attempts of tableau proofs of mps in Problem 22 which falsifies the corresponding mp.

25. Prove soundness of analytic tableau for the logics K, T, B, D, K4, KT4, and KT45. What about the completeness?

7

Some Other Logics

7.1 Introduction

Towards the end of Chapter 6, you exposed to the basics of logic engineering. The issue was: with a problem domain in view, what would be the most appropriate logic to use? There are still some situations where an earlier logic would naturally fit in; some other logic would be more appropriate. In such a case, you may attempt at modifying an existing logic. This activity of modification of an existing logic can broadly be of two kinds. One approach is to extend the vocabulary and then give an appropriate semantics. These new logics talk about things that the earlier logics could not. An example was FL as a modification of PL. The same way, modal logics have also been obtained as extensions of PL. The other direction in creating new logics is not by extension, but by restriction on validity. Logics obtained this way have almost the same vocabulary as those of the earlier ones. They differ from the earlier ones by way of a different or restricted semantics, and having a different set of valid formulas. These logics are also obtained by asserting different axioms rather than by prescribing a semantics. If you follow this direction, you have then two options: the syntactic way of proof theory where a different axiomatic system would result in a different logic or the semantic way of giving different meanings to the symbols. The logics so obtained are called **deviant logics**.

The new logics obtained either way may or may not keep the metalogical properties of the earlier ones. By picking out different properties and then looking at various logics through the properties is another way of classification. One of the many such properties, namely, monotonicity, has attracted considerable attention. Recall the definition of $\Sigma \models w$ in PL or $\Sigma \vdash w$ in PC, GSC, PND, PT, or in resolution. All these definitions attempt at defining the consequence relation which we visualize to hold between a set of propositions Σ and a proposition w. This notion can be thought of as a relation and, then the semantic (model theoretic) considerations or the syntactic derivations (proof theoretic) are simply mechanisms to determine whether the pair (Σ, w) is an element of this relation or not. Here, a logic is viewed as a mechanism to define the consequence relation.

Thus, defining a consequence relation (any relation between a set of formulas and a formula) arbitrarily leads to different logics. If the relations coincide with the ones corresponding to the consequence relations of existing logics, then nothing has been achieved. However, the relation might be different from the existing ones, and in such cases, you would get logics very different from the existing ones. For example, suppose that you have a mechanism to convert every proposition to an integer. (You can have many such mechanisms since the set of propositions, in PL, is countable.) Define the consequence relation $\mathcal{C}$ by $(\Sigma, w) \in \mathcal{C}$ iff the integer corresponding to w divides the product of the integer representations of some of the elements of Σ. Taking this as the consequence relation, you can get a deviant logic of propositions.

But, do you really want any arbitrary relation between a set of propositions and a proposition to serve as a consequence relation? For ease in reading, let us write $(\Sigma, w) \in \mathcal{C}$ as $\Sigma \Vdash w$. A suggestion was put forth by A. Tarski and D. Scott to regard any such relation as a consequence relation provided it possesses the following properties:

Reflexivity : If $w \in \Sigma$, then $\Sigma \Vdash w$.

Transitivity : If $\Sigma \Vdash X$ and $\Sigma \cup \{X\} \Vdash w$, then $\Sigma \Vdash w$.

Monotonicity : If $\Sigma \subseteq \Gamma$ and $\Sigma \Vdash w$, then $\Gamma \Vdash w$.

Such are the **monotonic logics**. Both PL and FL are monotonic logics, in the sense that their consequence relations satisfy all the above properties. There are other logics which do not satisfy the monotonicity condition. These logics are known as nonmonotonic logics. A nonmonotonic logic can be an extended logic, or a deviant logic, depending upon whether it involves a nontrivial extension of vocabulary or a reinterpretation of the same vocabulary.

In this chapter, we plan to discuss logics keeping in mind both the classes. The treatment is very brief, and it aims to show you how so many varieties of logics have been invented out of necessity. You will, most probably, create your own logic which may fit a new situation more tightly than any of the logics known as of today.

7.2 Intuitionistic Logic

You have seen a proof (in FL) of existence of two algebraic irrational numbers a and b such that a^b is rational. Specifically, the proof uses the idea that either $\sqrt{2}^{\sqrt{2}}$ is rational, or else, $(\sqrt{2}^{\sqrt{2}})^{\sqrt{2}}$ does the job. However, it does not exhibit, in particular, which of the two pairs, $(\sqrt{2}, \sqrt{2})$ or $(\sqrt{2}^{\sqrt{2}}, \sqrt{2})$, serves the purpose of (a, b).

Some mathematicians, now called intuitionists, object to accepting such a proof. The objection is that we do not know whether $\sqrt{2}^{\sqrt{2}}$ is rational or irrational. Therefore, we do not yet have, at our disposal, a pair of irrationals (a, b) with the required property.

According to this school of thought, a proposition $p \vee q$ can be true only when at least one of them has been demonstrated to be true. This implies that the law of excluded middle cannot be regarded as a law at all. We know of many twin primes (a pair of positive integers $(m, m+2)$ is a twin prime if both $m, m+2$ are primes), but we do not know whether there is a twin prime bigger than (both $m, m+2$ bigger than) $10^{10^{10}}$. We also do not know whether there is no twin prime bigger than $10^{10^{10}}$. Hence, we cannot, at this stage, accept the truth of the sentence: "Either there is twin prime bigger than $10^{10^{10}}$ or there is no twin prime bigger than $10^{10^{10}}$." This is the basic insight of L. E. J. Brower who says that if the law of excluded middle is accepted, then we are assuming that every mathematical problem is solvable, and this is certainly objectionable.

Look at the consequences of rejecting the law of excluded middle. In its absence, we cannot assert the law of double negation in PL. Though $A \rightarrow \neg\neg A$ would still remain valid, its converse $\neg\neg A \rightarrow A$ will no more hold. So, in the intuitionistic logic, INT, negation must have a different meaning.

***Exercise* 7.1** Why is it that rejection of the law of excluded middle leads to the rejection of the validity of $\neg\neg A \rightarrow A$? Why does the conditional $A \rightarrow \neg\neg A$ still hold? [*Hint*: Use PC instead of PL.]

***Exercise* 7.2** Show that the rejection of the law of excluded middle leads to the invalidity of $\neg(A \wedge B) \rightarrow (\neg A \vee \neg B)$.

Similarly, a contradiction in INT cannot be understood as any statement of the form $A \wedge \neg A$, rather an instance of it such as $0 = 1$ or $3 \times 5 = 16$. Note that INT is the **intuitionistic propositional logic**; so it has the same vocabulary as that of PL. We do not take $\leftrightarrow$ into its vocabulary right now since this can be expressed in terms of $\rightarrow$ and $\wedge$ as in PC. INT assigns different meanings to the connectives; a variant logic, indeed. We follow the reverse process here to the one taken historically. An earlier axiomatization of INT was by A. Heyting, in which axioms and the law of uniform substitution had been used. From among many axiomatizations, we choose one by M. Dummett, which uses axiom schemes. It is as follows:

Axiom Schemes of INT:

(I1) $A \rightarrow (B \rightarrow A)$

(I2) $A \rightarrow (B \rightarrow (A \wedge B))$

(I3) $(A \wedge B) \rightarrow A$

(I4) $(A \wedge B) \rightarrow B$

(I5) $A \rightarrow (A \vee B)$

(I6) $B \rightarrow (A \vee B)$

(I7) $(A \vee B) \rightarrow ((A \rightarrow C) \rightarrow ((B \rightarrow C) \rightarrow C))$

(I8) $(A \to B) \to ((A \to (B \to C)) \to (A \to C))$

(I9) $(A \to B) \to ((A \to \neg B) \to \neg A)$

(I10) $A \to (\neg A \to B)$

Rules of Inference of Int:

(MP) $\dfrac{A \qquad A \to B}{B}$

A possible world semantics of Int uses the same idea of a frame and a model as in the modal logics, except that all our frames are now reflexive and transitive. That is, a model is a triple $M = (W, R, \Vdash)$, where W is a nonempty set of worlds, R is a reflexive and transitive relation on W, and $\Vdash$ is a relation from W to the power set of all atomic propositions. The relation $\Vdash$ is extended to include all propositions in its domain, as

$w \Vdash \neg p$ iff for all $z \in W$ with wRz, $z \nVdash p$

$w \Vdash p \wedge q$ iff $w \Vdash p$ and $w \Vdash q$.

$w \Vdash p \vee q$ iff $w \Vdash p$ or $w \Vdash q$.

$w \Vdash p \to q$ iff for all $z \in W$ with wRz, $z \nVdash p$ or $z \Vdash q$.

Finally,

$M \models p$ iff for all $w \in W$, $w \Vdash p$.

As assumed earlier, the completeness and the soundness of the axiomatic system Int can be proved with respect to the above semantics. However, since we use a possible world semantics for Int, it must have some connection with modal logics. It can be verified that the following translation (due to K. Gödel) of the logic S4 (KT4) holds. Writing the translation by the map $*$, we have

$p^* = p$

$(\neg p)^* = \neg \Box p^*$

$(p \vee q)^* = \Box p^* \vee \Box q^*$

$(p \wedge q)^* = \Box p^* \wedge \Box q^*$

$(p \to q)^* = \Box p^* \to \Box q^*$

You can interpret $\Box$ as ‘it is demonstrated that’. Now, rethink along the lines of the two introducing illustrations above (that of a^b and of the twin primes).

What is the relation between PL-validity and Int-validity? Note that all axioms of Int are PL-valid, and the inference rule is MP, which is also a valid consequence of PL. It follows (By induction? Induction on what?) that every Int-valid proposition is PL-valid. The converse must not hold;

otherwise, INT will be the same as PL! The relation is best understood by noting that whenever p is PL-valid, its double negation $\neg\neg p$ is also PL-valid. But this does not happen in INT. The following result (We will not prove it, see [31]) explains the connection between PL and INT. We write $\vdash_{PL}$ for PL-validity (or PC-provability, consequences) and $\vdash_{\text{INT}}$ for INT-validity or INT-provability or INT-consequences.

Theorem 7.1 *For a set of propositions Σ, let $\neg\neg\Sigma = \{\neg\neg p : p \in \Sigma\}$. Let A be any proposition. Then,*

(a) $\Sigma \vdash_{PL} A$ iff $\neg\neg\Sigma \vdash_{\text{INT}} \neg\neg A$ (b) $\vdash_{PL} \neg A$ iff $\vdash_{\text{INT}} \neg A$

Note that in part (b) of Theorem 7.1, $\neg A$ cannot be replaced by A. What is the difference between the statement in (b) and '$\vdash_{PL} A$ iff $\vdash_{\text{INT}} A$'? If A is a proposition that uses only $\neg$ and $\wedge$, then $\vdash_{PL}$ iff $\vdash_{\text{INT}}$. Prove it. Again, the consequence relation does not have this property even in the restricted language, with the only connective as $\neg$ and $\wedge$. For example, $\neg\neg p \vdash_{PL} p$ but $\neg\neg p \nvdash_{\text{INT}} p$.

Theorem 7.1 suggests that all PL-theorems can have a translation into INT since each theorem of PL is a theorem of INT, and some of INT-theorems might be missing in PL. The following is such a translation of PL-theorems to INT-theorems via the map † :

$$p^\dagger = \neg\neg p$$
$$(p \wedge q)^\dagger = p^\dagger \wedge q^\dagger$$
$$(p \vee q)^\dagger = \neg(\neg p^\dagger \wedge \neg q^\dagger)$$
$$(p \to q)^\dagger = \neg(p^\dagger \wedge \neg q^\dagger)$$

This translation preserves consequences. However, if $p^\dagger = \neg\neg p$ is replaced by $p^\dagger = p$, then only theoremhood will be preserved, (see [30, 49]). There is also one more semantics, called the set assignment semantics for INT due to [16]. Other proof procedures for INT have also been developed. Instead of PL, if you consider FL, you can get the intuitionistic first order logic. See the summary to this chapter for suggested references.

7.3 Łukasiewicz Logics

Consider the sentence: "There will be an Indo-Pak war tomorrow". Is it true? Is it false? Since nothing about its truth or falsity is known today, it is neither true nor false. But it is quite possible that tomorrow such a war might break out. Its truth is beyond the classical bivalence. It is something like an amoral action, which is neither moral nor immoral; it is beyond morality. To take into account such propositions, which do come up in abundance in day-to-day life, the bivalent logics like PL or FL would not suffice. What about assigning a new truth value, with the understanding that this new truth value does not say that 'it is this much true'; but that, it is beyond the truth values of 0 and 1, representing 'false' and 'true'. It admits of another choice to the seemingly exhaustive dichotomy: "I can accept it" or "I cannot accept it".

Suppose that we agree to have one more truth value, say, $\frac{1}{2}$, in addition to 0 and 1. As in PL, we start with $\top, \bot$, and the propositional variables as atoms and use the connectives $\neg, \wedge, r, \rightarrow$ (we suppress $\leftrightarrow$ for the time being) to build up a formal language of propositions. The semantics of the logic is, of course, different; we have to take three truth values into consideration. Table 7.1 fixes the meanings of the connectives.

Table 7.1 Truth Table with Three Truth Values

p	q	$\neg p$	$p \wedge q$	$p \vee q$	$p \rightarrow q$
0	0	1	0	0	1
$\frac{1}{2}$	0	$\frac{1}{2}$	0	$\frac{1}{2}$	$\frac{1}{2}$
1	0	0	0	1	0
0	$\frac{1}{2}$		0	$\frac{1}{2}$	1
$\frac{1}{2}$	$\frac{1}{2}$		$\frac{1}{2}$	$\frac{1}{2}$	1
1	$\frac{1}{2}$		$\frac{1}{2}$	1	$\frac{1}{2}$
0	1		0	1	1
$\frac{1}{2}$	1		$\frac{1}{2}$	1	1
1	1		1	1	1

***Exercise* 7.3** Construct the truth table for $\leftrightarrow$, where $p \leftrightarrow q$ is defined as $(p \rightarrow q) \wedge (q \rightarrow p)$.

Writing the truth value of a proposition p as $t(p)$, a mapping, the table above can be abbreviated to

$$t(\top) = 1,\ t(\bot) = 0$$

$$t(\neg p) = 1 - t(p)$$

$$t(p \wedge q) = \min\{t(p), t(q)\}$$

$$t(p \vee q) = \max\{t(p), t(q)\}$$

$$t(p \rightarrow q) = 1, \text{ if } t(p) = t(q) = 1, \text{ else, } t(p \rightarrow q) = \max\{1 - t(p), t(q)\}$$

***Exercise* 7.4** Verify the above properties of $t(p)$.

Thus, the PC-tautology that $p \rightarrow q$ is equivalent to $\neg p \vee q$ holds no more in this three-valued logic. But now, what does it mean by equivalence? Well, $p \equiv q$ iff they have the same truth table. Formally, you can define an interpretation as a map from all propositions to the set of truth values $\{0, \frac{1}{2}, 1\}$, obeying the conditions given in the truth table. Alternatively, you can show that the map $t(\cdot)$ is well defined on the set of all propositions (due to unique parsing), starting from a preliminary definition on the set of atomic propositions. That defines the concept of an interpretation; each such map is an interpretation. Then you can define equivalence by

$p \equiv q$ iff $t(p) = t(q)$ for any interpretation t.

We call this *three-valued logic*, $\mathbf{L_3}$ after Łukasiewicz.

***Exercise* 7.5** Show that in L_3, $p \vee \neg p \not\equiv \top$.

The law of trivalence (like excluded middle), however, holds in L_3. To formulate this law, we introduce the indeterminate proposition in L_3. Let p be any propositional variable. Define a proposition ιp by

$$\iota p \overset{\circ}{=} p \leftrightarrow \neg p$$

The proposition ιp is called an **indeterminate proposition**. The truth table of ιp can be given as

p	0	$\frac{1}{2}$	1
ιp	0	1	0

Any proposition A equivalent to $\top$ is called an **L_3-tautology** or an **L_3-valid** proposition, and is written as $\models_{\mathrm{L}_3} A$.

***Exercise* 7.6** Show that $\models_{\mathrm{L}_3} p \vee \neg p \vee \iota p$. What are $\iota\top$ and $\iota\bot$?

The **law of trivalence** is the statement in Exercise 7.6:

$p \vee \neg p \vee \iota p \equiv \top$ in L_3 for any proposition p.

You can also check the following for any interpretation t in L_3:

$t(p \to q) = 1$ iff $t(p) \leq t(q)$

$t(p \leftrightarrow q) = 1$ iff $t(p) = t(q)$

Let Σ be a set of propositions (in L_3 now) and w be any proposition. Define the validity of an **L_3-consequence** by

$\Sigma \models_{\mathrm{L}_3} w$ iff for every interpretation t, if $t(A) = 1$ for every $A \in \Sigma$, then $t(w) = 1$.

You can check that the following consequences hold:

Modus Ponens :	$\{p, p \to q\} \models_{\mathrm{L}_3} q$
Equivalence Substitution :	$p \leftrightarrow q \models_{\mathrm{L}_3} A \leftrightarrow A[p := q]$
Uniform Substitution :	If $\models_{\mathrm{L}_3} B(p)$, then $\models_{\mathrm{L}_3} B(q)$

where $A[p := q]$ is obtained from A by substituting some or all or no occurrence of p in A by q, and $B(q)$ is obtained from $B(p)$ by substituting every occurrence of p in B by q.

Moreover, J. Łukasiewicz constructed L_3 for arguing about 'possibility' and 'necessity' by using the following truth table:

p	$\Box p$	$\Diamond p$
0	0	0
$\frac{1}{2}$	0	1
1	1	1

From the above data, it is clear that $\Diamond p \equiv \neg p \to p$ and $\Box p \equiv \neg\Diamond\neg p$. (Verify.) Though $A \to \Box A$ is not L$_3$-valid, we have $\models_{\text{L}_3} A \to (A \to \Box A)$. This suggests a deduction theorem.

Theorem 7.2 (Deduction in L$_3$) *For any set* Σ *of propositions and any propositions* A, B, $\Sigma \cup \{A\} \models_{L_3} B$ *iff* $\Sigma \models_{L_3} A \to (A \to B)$.

***Exercise* 7.7** Construct a truth table for $p \to (p \to q)$ and show that $p \to (p \to q) \equiv \Diamond\neg p \vee q$. Then, prove Theorem 7.2.

Further, we can have a translation of PL (or PC) into L$_3$, which preserves consequences. This is achieved via the translation map $*$ from PL to L$_3$ by

$$p^* = p, \quad \text{for atomic propositions } p$$
$$(\neg p)^* = p^* \to (p^* \to \neg(p^* \to p^*))$$
$$\Sigma^* = \{p^* : p \in \Sigma\}, \quad \text{for any set } \Sigma \text{ of propositions.}$$

***Exercise* 7.8** With $*$ as above, show that $\Sigma \models_{PL} A$ iff $\Sigma^* \models_{\text{L}_3} A^*$. Also, show in L$_3$ that $p \vee q \equiv (p \to q) \to q$ and $p \wedge q \equiv \neg(\neg p \vee \neg q)$.

Note that a translation such as the above identifies PL in L$_3$. A simplistic way of identifying both 1 and $\frac{1}{2}$ with 'true', i.e., 1 in Pl would not do. Similarly, identifying both 0 and $\frac{1}{2}$ with 0 will also fail. (Why?) However, any L$_3$-tautology is vacuously a PL-tautology since the truth tables of connectives in L$_3$ restricted to the values of 0 and 1 are simply the PL-truth tables.

The truth function σp defined by σp is $\frac{1}{2}$, for every value of p is not definable from the connectives $\neg, \wedge, \vee$, and $\to$. This is so because, if it were, then $\neg\sigma p$ and σp would be equivalent, forcing $\models_{\text{L}_3} \neg\sigma p \to \sigma p$ to hold. Since each L$_3$-tautology is a PL-tautology, $\neg\sigma p \to \sigma p$ would also be PL-valid. But this is obviously wrong as $\neg A \to A$ is not PL-valid. This truth function σ is called the *Słupecki operator*. Consideration of the Słupecki operator shows that for the truth functional completeness, the connectives $\neg, \wedge, \vee$ are not enough.

An adequate axiomatic system (see [82]) for L$_3$ which uses $\neg$ and $\to$ as in PC, has the following axiom schemes and the rule of inference:

Axiom Schemes for L$_3$

(L1) $A \to (B \to A)$

(L2) $(A \to B) \to ((B \to C) \to (A \to C))$

(L3) $(\neg A \to \neg B) \to (B \to A)$

(L4) $((A \to \neg A) \to A) \to A$

Rule of Inference

(MP) $\dfrac{A \quad A \to B}{B}$

A generalization of Łukasiewicz three-valued logic L_3 has been obtained by him and A. Tarski. We briefly mention its peculiarity here. The idea is to view the truth values as any real number between 0 and 1, instead of the three values 0, $\frac{1}{2}$, 1. An interpretation is taken as a map from the set of atoms to the interval $[0,1]$. This map is extended to all propositions by

$$t(\neg p) = 1 - t(p)$$
$$t(p \to q) = 1, \text{ if } t(p) \le t(q), \text{ else, } t(p \to q) = 1 - t(p) + t(q)$$
$$t(p \vee q) = \max\{t(p), t(q)\}$$
$$t(p \wedge q) = \min\{t(p), t(q)\}$$

***Exercise* 7.9** Look at $t(p \to q)$ above. Show that it is the same as the one mentioned in L_3 if the range of t is restricted to $\{0, \frac{1}{2}, 1\}$. Also, find how $t(p \leftrightarrow q)$ is to be defined in the above.

Many logics can be defined using this scheme t of truth values, depending upon what subset of the interval $[0,1]$ is fixed as the co-domain of the interpretation t. The logics and their corresponding co-domain of the interpretations are given as follows:

Logic	Co-domain of interpretations
L_n	$\{\frac{m}{n-1} : 0 \le m \le n-1\}$
$\mathrm{L}_{\aleph_0}$	$\{\frac{m}{n} : 0 < m \le n,\ m, n \in \mathbb{N}\}$
$\mathrm{L}_{\aleph}$	$[0,1]$

In all the cases, a valid proposition is defined as the one in which $t(p) = 1$ for all interpretations. For example, $L_{\aleph}$ has the valid propositions as all p for which $t(p) = 1$ for every interpretation t with the co-domain as the interval $[0,1]$. In this scheme, we get many logics such as

$$\mathrm{L}_2,\ \mathrm{L}_3,\ \mathrm{L}_4,\ \mathrm{L}_5,\ \ldots,\ \mathrm{L}_{\aleph_0},\ \mathrm{L}_{\aleph}$$

Note that L_2 is simply the propositional logic PL. Further, $\mathrm{L}_n \ne \mathrm{L}_{n+1}$, and each valid proposition of L_m is also L_n-valid provided that n divides m. Again, $\mathrm{L}_{\aleph_0}$ is the same as $\mathrm{L}_{\aleph}$ in the sense that their valid propositions coincide. This is a nontrivial result (see [18]).

***Exercise* 7.10** Show that $\mathrm{L}_{\aleph}$ can be axiomatized with the axiom schemes L1, L2, L3, $(A \to B) \vee (B \to A)$, and the inference rule MP.

Now, in view of these many-valued logics, what do you think of INT? For a proposition p, you have $\neg p$, $\neg\neg p$, and $\neg\neg\neg p$ is equivalent to $\neg p$ in INT.

So, is Int a three-valued logic? Is it L_3, say, in the sense that Int-valid propositions coincide with L_3-valid propositions? Or, is the converse true? It is again not so trivial a result. Gödel proved in 1932 that Int cannot be characterized as any finite valued logic L_n (for $n = 2, 3, 4, \ldots$).

Still there are other varieties of many-valued logics. An example is Kleene's three-valued logic which has also the same three truth values $0, \frac{1}{2}, 1$. The difference between L_3 and this logic is that here, when both A, B are $\frac{1}{2}$, the formula $A \to B$ is evaluated to 1 (Compare with L_3, where $t(\frac{1}{2} \to \frac{1}{2}) = \frac{1}{2}$). Similarly, you can define any other many-valued logic by playing with the truth values and the connectives. However, Gödel's result that Int cannot be characterized by any such finite-valued logics still holds. See the summary at the end of the chapter for sources if you are interested in this (non)characterization result.

7.4 Probabilistic Logic

A doctor wants to decide whether a patient has the disease D after listening to and examining the patient for symptoms. He only knows that certain symptoms and certain diseases occur with so and so probabilities. (In this section, we write $p(\cdot)$ for probabilities and not for denoting propositions.) Let us write $S \to D$ to mean that the symptom S implies the infliction of the disease D. To be specific, suppose that the doctor knows the following:

A1: $A \to D$ holds with probability p_1

A2: $\neg A \to D$ holds with probability p_2

A3: $B \to D$ holds with probability p_3

When the patient exhibits the symptoms $\neg A$ and B, the doctor wants to determine whether the patient has the disease D. Note that the probabilities assigned to the above statements are the doctor's knowledge, i.e., they are subjective. So, the diagnosis will also be subjective; it is his rational opinion. It is rational in the sense that the final decision uses an objective procedure even on the subjective probabilities. He uses his subjective probabilities to compute the probability of the patient suffering from the disease D given that the patient exhibits the symptoms $\neg A$ and B. The doctor's interest is in determining the conditional probability

$$p(D|\neg A \wedge B) = \frac{p(D \wedge \neg A \wedge B)}{p(\neg A \wedge B)}$$

However, such computations become all right when we have a probability space of propositions and then a predefined probability measure (may be subjective) on such a space, i.e., a probability distribution of propositions. A probabilistic logic assumes all these. In the diagnosis problem, we may consider the set of relevant propositions to be in the set

$$U = \{A, B, A \wedge D, \neg A, \neg A \wedge D, B \wedge D, D, A \wedge B, \neg A \wedge B, D \wedge \neg A \wedge B\}$$

Then, we go for constructing the set of elementary propositions (not necessarily atomic) so that a probability measure can be defined. In general, the set $E = \{E_1, E_2, \ldots, E_n\}$ of elementary propositions will satisfy the following properties:

1. Each of the relevant propositions is a disjunction of some elementary propositions.
2. The elementary propositions are exhaustive: $E_1 \vee \ldots \vee E_n \equiv \top$.
3. The elementary propositions are mutually exclusive, i.e., $E_i \wedge E_j \equiv \bot$ for $i \neq j$.

Thus, in our diagnostic problem, we take the set $E = \{A, B, D, \neg A, \neg B, \neg D\}$ assuming that A, B, D are atomic. Next, a probability measure is defined on the set

$$\mathcal{E} = E \cup \{\top, \bot\}$$

A probability measure is a function $p : \mathcal{E} \to [0, 1]$ satisfying the following properties:

1. For any $X, Y \in \mathcal{E}$, $p(X) \geq 0$.
2. $p(\top) = 1$, $p(\bot) = 0$.
3. If $X \wedge Y \equiv \bot$, then $p(X \vee Y) = p(X) + p(Y)$.

The probabilities are used to compute the required conditional probability. Note that the restrictions of a probability measure are imposed on the subjective probabilities of a decision-maker here, the doctor. This is the reason that a decision-maker who uses probabilistic logic is assumed to be a rational agent. As you have guessed, probabilistic logic is rather a logical methodology for decision making than a logic. One can go further in deciding upon a consensus by taking opinions from experts for fixing the subjective probabilities of elementary propositions. One may also resort to statistical methods while arriving at a consensus. Look at the summary at the end of this chapter for materials on this topic.

7.5 Possibilistic and Fuzzy Logic

In a probabilistic framework, it is not possible to assign a probability to ignorance; we only assign a probability to what is known. On the contrary, possibilistic logic is a logic of partial ignorance. And fuzzy logic is a logic of vagueness. In this section, we will have a rough outline of the two types of logics that deal with uncertainty and vagueness.

Consider the sentence s: Sam is tall. By introducing a variable h for the height of Sam, you can translate the sentence to $h = \mathit{tall}$. If h takes values from the set { tall, medium, short }, then the truth predicate P_h can be evaluated to one value, depending upon whether h is tall, medium or short. Say, P_h is either 1, $\frac{1}{2}$ or 0 according as h is tall, medium, or short.

Then the predicate P_h is called a *crisp predicate*, as it has definite values. If height is allocated a value, say, between 100 cm and 250 cm, then P_h will not have one of the many finite values; it is now referred to as a *vague predicate.* The range of values, the closed interval $U = [100\,\text{cm}, 250\,\text{cm}]$, is our universe for height h, and P_h is a vague predicate defined on this universe U. Since here the predicate P_h is unary, when it is crisp, it is just a subset of U. What would happen if P_h is a vague predicate? It is clearly not a subset, for we do not know or we cannot know for certainty which elements of U are in P_h and which are not. In such a case, we say that P_h is a *fuzzy subset* of the universe U.

A subset of a set U is characterized by its *characteristic* or *indical function* $\chi_A : U \to \{0, 1\}$, by the rule that for any $u \in U$, $u \in A$ iff $\chi_A(u) = 1$. In case A is a fuzzy subset of U, we identify this fuzzy subset with the *membership function* $\mu_A : U \to [0, 1]$. Thus, any element $u \in U$ is a fuzzy member of the fuzzy subset A of U determined by its degree of membership $\mu_A(u)$. (Imagine membership to admit of degrees rather than being either 0 or 1.) If $\mu_A(u) = 1$, then $u \in A$ as in crisp subset case. If $\mu_A(u) = 0$, then $u \notin A$ again as in the crisp subset case. If $\mu_A(u) = \alpha$ for $0 < \alpha < 1$, then the degree of membership of u in A is α. When P_h is taken as fuzzy subset of U (as a vague predicate), we have a membership function μ_{P_h}. For curtailing subscripts, write P_h as B. Then μ_B gives the idea as to how much tall Sam is, or rather, "what is the degree of tallness of Sam when his height is so and so". Obviously, the following cases occur.

1. *Crisp Sentences and Precise Information*

Suppose that $B = \{150\,\text{cm}\}$, i.e., we now know that Sam is 150 cm tall. Here, $B \subseteq U$ is a crisp subset of U. Depending upon whether we consider 150 cm as a height to be termed as tall or not, $\mu_B(s)$ will take a value in $\{0, 1\}$. That is, the truth of the sentence s is either 0 (s is false) or 1 (s is true): $t(s) = \mu_B(s) \in \{0, 1\}$. Here, we know for precision, that Sam has so and so height. We also have a crisp sentence since we have the information whether this height is termed as tall or short. Thus the case is compared to PL, where the truth of every sentence is either 0 or 1.

2. *Crisp Sentences and Imprecise Information*

Suppose that we know that Sam's height is within 150 cm to 200 cm, i.e., we do not have precise information here. Suppose also that the meaning of tall is crisp, i.e., if the height of a person is within a range, say, a to b in centimetres, then he is called *tall.* Symbolically, the predicate 'tallness' is identified with the crisp subset $[a, b] \subseteq U$. Then, the following cases are to be considered:

(a) If $B = [150, 200] \subseteq A = [a, b]$, then clearly, $t(B) = 1$; Sam is tall.

(b) If $A \cap [a, b] = \emptyset$, then $t(B) = 0$; Sam is not tall.

(c) If $A \cap [a, b] \neq \emptyset$ but $A \not\subseteq [a, b]$, then $t(B)$ cannot be fixed to either 0 or 1. The sentence "Sam is tall" is possibly true or possibly false.

These situations are tackled by introducing a possibility measure π as

for any subset $E \subseteq U$ $\pi(E) = 1$, if $E \cap A \neq \emptyset$, else, $\pi(E) = 0$.

Then the above cases correspond to:

(a) $\pi(s) = 1, \pi(\neg s) = 0$.

(b) $\pi(s) = 0, \pi(\neg s) = 1$.

(c) $\pi(s) = 1, \pi(\neg s) = 1$.

The case $\pi(s) = 0, \pi(\neg s) = 0$ leads to inconsistency; hence, it cannot occur.

3. ***Crisp Sentences and Fuzzy Information***

Suppose that the meaning of tallness is crisp, i.e., we have a range of values for tallness, say, if the height of anyone is in $[a, b] \subseteq U$, then he is called tall. Assume also that our information on Sam's height is fuzzy. That is, $B = P_h$ is now a fuzzy subset of U which, of course, is given by a membership function μ_B. Then, our answer to the query 'whether Sam is tall' will have a fuzzy answer, i.e., tallness of Sam will have the degree of truth as $\mu_B(s)$.

4. ***Vague Sentences and Fuzzy Information***

Here, both the sets representing tallness and Sam's height are fuzzy subsets of U. Suppose that S represents Sam's height, i.e., S is a fuzzy subset of U given by its membership function μ_S, and tallness is also a fuzzy subset B of U given by its membership function μ_B. Then the answer to the query 'whether Sam is tall' will also be fuzzy. That is, the truth of the sentence "Sam is tall" will take a fuzzy truth value in the interval $[0, 1]$, depending upon the values of μ_B and μ_S. For example, "Sam's height is about 170 cm" is a fuzzy sentence. Such a sentence is translated to a membership function $\mu_S : [0, 1] \to [0, 1]$ defined as

$$\mu_S(v) = \sup \{\pi(u) : \mu_B(u) = v, u \in U\}$$

Note that whenever $\mu_B^{-1}(v) = \emptyset$, $\mu_S(v) = 0$ since the supremum over an empty set is taken as 0. The values $\mu_S(v)$ is the grade of possibility that the degree of truth of S is v. Then the truth value $t(s)$ of the sentence "Sam is tall" is approximated by the numbers $N(s)$ and $\Pi(s)$ given by

$$N(s) \leq t(s) \leq \Pi(s)$$

$$\Pi(s) = \sup \min \{\mu_B(u), \pi(u) : u \in U\}$$

$$N(s) = 1 - \Pi(\neg s) = \inf \max \{\mu_B(u), 1 - \pi(u) : u \in U\}$$

This is the most general case, as all the earlier cases will fall into place by regarding the fuzzy subsets as crisp. The function Π is called the *possibility* *measure* and N is called the *necessity measure* so that the uncertainty or fuzzy measure $t(s)$ lies between the possibility and necessity.

The high rate at which work in fuzzy logic and fuzzy mathematics is growing at present prevents a nonspecialist to catch up. However, it is wise to brush up some basic literature so that it will be easy for you later when some application domain demands fuzzy logic ideas. The summary at the end of this chapter will help you make a good start.

7.6 Default Logic

In this section, we will briefly look at another way of dealing with vagueness. The vagueness here is associated to limitations of knowledge or in its representation. For example, when we say that 'birds fly', we do not mean that all birds fly without exception, nor do we assert that only some birds fly. It is almost a general rule that birds fly; however, there might be exceptions, as we know that penguins do not fly and that a wounded bird may not fly. It is this kind of vagueness in knowledge or in the representation of knowledge that we want to look at. Such facts can be represented by first order consequences such as

$$P(x) \land \neg\, exception_1(x) \land \cdots \land \neg\, exception_m(x) \models Q(x)$$

In so doing, we are treating exceptions as general facts defeating the intention of a 'general rule'. We are also assuming that these are all possible exceptions. Tomorrow, we may discover another species of birds which might not fly! To tackle such cases, we introduce a type of rule, called *default.*

If we know that tweety is a penguin, all penguins are birds, birds fly, and penguins do not fly, then we should be able to conclude that tweety does not fly. We would not just dispense with the facts by telling that the facts are inconsistent. (Where is the inconsistency?) Here, we may take the following as facts:

Each penguin is a bird.

Tweety is a bird.

Tweety is a penguin.

The sentence 'birds fly' will be taken not as a fact, but as a default rule since this is the one that may admit of exceptions. The default rule is:

If x is a bird and it cannot be proved that x is a penguin, then deduce that x flies.

Or, as a fraction:

$$\frac{x \text{ is a bird} : x \text{ is not a penguin}}{x \text{ flies}}.$$

Thus ,“ : x is not a penguin” would now mean "if it cannot be proved that x is not a penguin"

This can also be written as

"if it is consistent to assume that x is not a penguin"

Formally, a default (a default rule) looks like

$$\frac{u(x)\ :\ v_1(x),\ldots,v_m(x)}{w(x)}$$

where $u(x), v_1(x), \ldots, v_m(x)$ and $w(x)$ are well-formed formulas (of FL, in general) whose free variables are among $x_1, \ldots, x_n$, written here as a single symbol x. The formula $u(x)$ is called the *prerequisite* of the default. The formulas $v_1(x)\ldots, v_m(x)$ are the *justifications* and $w(x)$ is called the *consequent* of the default. The meaning of the default is

If $v_1(x), \ldots, v_m(x)$ are consistent with what is already known, then $w(x)$ is inferred.

A *default theory* is a pair of sets $\Delta = (D, W)$, where D is a set of default rules and W is a set of FL-sentences.

Default logic is a nonmonotonic logic in the sense that addition of new facts may invalidate earlier established consequences. For example, with the default theory $\Delta = (D, W)$, where $W = \{bird(tweety)\}$ and D having the single rule:

$$\frac{bird(x)\ :\ \neg\, penguin(x)}{flies(x)}$$

we have a consequent $flies(tweety)$. Now, with $W' = W \cup \{penguin(tweety)\}$, we cannot deduce the consequent $flies(tweety)$.

***EXAMPLE* 7.1** Suppose that we have a universe (constants) having two members, say a and b. Assume that an object in this universe is not an elf unless it is required to be. Moreover, at least one of a or b is an elf. Then its default theory is

$$\Delta = (D, W) \text{ with } W = \{elf(a) \vee elf(b)\},\ D = \left\{\frac{:\ \neg\, elf(x)}{\neg\, elf(x)}\right\}.$$

The default rule in D says that "if it is consistent to assume that x is not an elf, then deduce that x is not an elf". Then, you see that neither $elf(a)$ nor $elf(b)$ can be inferred from the single fact $elf(a) \vee elf(b)$. It suggests that (as in PL or in FL) $\neg\, elf(a) \wedge \neg\, elf(b)$ should be provable.However, this sentence is inconsistent with the fact $elf(a) \vee elf(b)$.

In order to avoid this inconsistency, default reasoning admits of many *extensions*. In the above example, we have an extension (an extended theory of Δ), where we can infer $\neg\, elf(a) \wedge elf(b)$, and in another extension of Δ, we can infer $elf(a) \wedge \neg\, elf(b)$.

Formally, let $\Delta = (D, W)$ be a given default theory. Let S be a set of FL-sentences. For any set A of FL-sentences, denote by $Th(A)$, the theory of A, i.e., the set of all FL-sentences which can be inferred from the set of premises as A. Define $\Gamma(S)$ as the smallest set satisfying the properties:

1. $W \subseteq \Gamma(S)$
2. $\Gamma(S) = Th(\Gamma(S))$
3. For $u \in \Gamma(S), \neg v_1, \ldots, \neg v_m \notin S$ if $\dfrac{u\ :\ v_1, \ldots, v_m}{w} \in D$, then $w \in \Gamma(S)$.

A set E is an extension for Δ iff $\Gamma(E) = E$.

You can interpret $\Gamma(S)$ as the minimal set of beliefs that one can have in view of S, where S indicates which justification for beliefs is to be admitted. E is a fixed point of the operator Γ, just as $Th(A)$ is a fixed point of the operator Th in FL $(Th(Th(A)) = Th(A))$.

Alternatively, extensions of default theories can be defined starting from within. Let $\Delta = (D, W)$ be a default theory. An extension for Δ is any set

$$E = \cup_{i \in \mathbb{N}} E_i$$

where the sets E_i are defined recursively by

$$E_0 = W$$

$$E_{i+1} = Th(E_i) \cup \{w : \frac{u\ :\ v_1, \ldots, v_m}{w} \in D, u \in E_i, \neg v_1 \ldots, \neg v_m \notin E_i\}$$

Note that this is not a constructive definition of an extension E. It can be proved that whenever W is consistent, every extension of the corresponding theory is consistent. The following examples will show you that a proper translation of facts and rules to a default theory leads to a desired inference possible.

EXAMPLE 7.2 (From [15]) Consider the following facts:

(a) Generally, if Mary attends a meeting, Peter does not.

(b) Generally, if Peter attends a meeting, Mary does not.

(c) At least, one of Peter or Mary attends the meeting.

To have a default theory, let us use the following symbolization:

M : Mary attends the meeting. P : Peter attends the meeting.

Here,

$$\Delta_1 = (D_1, W_1),\ W_1 = \{M \vee P\},\ D_1 = \left\{\frac{M\ :\ \neg P}{\neg P}, \frac{P\ :\ \neg M}{\neg M}\right\}$$

This default theory has the unique extension $E_1 = Th(\{M \vee P\})$. However, it is awkward since we cannot infer from this extension that "both do not attend the same meeting". However, this sentence ought to be worth concluding. We will have another translation below, where this is possible.

This time, we take $\Delta_2 = (D_2, W_2)$, where $W_2 = W_1$, as earlier, but

$$D_2 = \left\{\frac{:\ M}{\neg P}, \frac{:\ P}{\neg M}\right\}$$

Here, the default translation is done by following the scheme:

‘Generally, if x attends a meeting, then y does not’

as

“if it is consistent to assume that x attends a meeting, then infer that y does not attend it”

Then, Δ_2 has two extensions (Verify!):

$$E_2 = Th(\{M, \neg P\}) \text{ and } E_2' = Th(\{P, \neg M\})$$

You see that $\neg(M \wedge P) \in E_2$ and $\neg(M \wedge P) \in E_2'$.

A still better translation would be:

$$\Delta_3 = (D_3, W_3),\ W_3 = W_1,\ D_3 = \left\{ \frac{:\ M \wedge \neg P}{\neg P}, \frac{:\ P \wedge \neg M}{\neg M} \right\}$$

This theory has two extensions E_2, E_2', as earlier so that $\neg(M \wedge P)$ holds in both of them. Further, suppose that we have additional facts:

(d) If Bill attends the meeting, then Peter attends.

(e) Bill attends the meeting.

Then, we can infer that Peter attends the meeting.

***Exercise* 7.11** In the first translation above, add (d) and (e), and see that ‘Peter attends’ is not provable in that default theory.
[Adding means having two more elements as B, $B \to P$ in W_1.]

7.7 Autoepistemic Logic

Can there be a knowledge base which has information about the scope and limits of its own knowledge? There can be. For example, in a database system, we often use the so-called *closed world assumption*, where ‘not’ is interpreted as “not in the database”. It is a built-in metaknowledge about the database. More flexible knowledge bases should have the ability to determine explicitly whether the knowledge of the base about a particular object in the base is in some sense complete or not. In such a scenario, we do not have to think about many agents and their knowledge or belief about facts stored in the base. We would rather have a formalization of the phrases such as ‘it is known that’, or ‘it is believed that’ as operators.

Writing $\Box p$ for ‘it is believed (known) that p’, the logic K45 would be an appropriate logic. This is so because the K45 axioms

$$\Box(p \to q) \to (\Box p \to \Box q), \quad \Box p \to \Box\Box p, \quad \neg\Box p \to \Box(\neg\Box p)$$

capture the functionality of the operator $\Box$ here. However, we require a nonmonotonic logic which is capable of dealing with this operator since new beliefs do change the earlier concluded propositions. In this section, we briefly discuss such a nonmonotonic logic of knowledge and belief. The requirement of nonmonotonicity is made clearer in the following example.

***EXAMPLE* 7.3** Let p be the proposition: "Sam has died in the ongoing war". First, if p holds, then I also believe it, i.e., $p \to \Box p$. Next, suppose that I do not believe in a statement if I do not have any basis for such a belief. Hence I have the sole premise $p \to \Box p$. And, due to the absence of any information regarding Sam's death, I do not believe that Sam has died in the ongoing war, i.e., I have $\neg \Box p$,. Since $p \to \Box p$ is equivalent to $\neg \Box p \to \neg p$, I conclude that $\neg p$. That is, Sam has not died in the ongoing war. However, if I have the information that Sam has, indeed, died, then I have two premises, p and $p \to \Box p$, from which I conclude $\Box p$. Note that I have p instead of $\neg p$, which was inferred earlier in the absence of any information in the knowledge base.

This is the kind of nonmonotonicity involved in a logic of belief or knowledge. Such a logic is called an **autoepistemic logic**. Syntactically, the **formulas of an autoepistemic logic** are the modal propositions of K; the semantics differs. Let us denote by L, the mps of K, which are now our building blocks for an autoepistemic logic. For simplicity, omit the symbols $\leftrightarrow$ and $\Diamond$; they can be introduced with the help of definitions such as $p \leftrightarrow q$ as $(p \to q) \wedge (q \to p)$ and $\Diamond p$ as $\neg \Box \neg p$. A subset $T \subseteq L$ is called an **autoepistemic theory**. An **autoepistemic interpretation** of an autoepistemic theory T is a function $I : L \to \{0, 1\}$ satisfying

(a) I conforms with a PL-interpretation,

(b) $I(\Box p) = 1$ iff $p \in T$.

An autoepistemic interpretation I is an **autoepistemic model** of T if $I(p) = 1$ for every $p \in T$. In such a case, we also say that p is true in I. As usual, an autoepistemic theory T is called **semantically complete** if T contains every formula that is true in every model of T. Similarly, T is **sound** with respect to a set of premises $A \subseteq L$ if every autoepistemic interpretation of T, which is a model of A is also a model of T.

Due to the absence of monotonicity, it is not easy to define the notion of a consequence. However, as in default logic, extensions would give rise to some *closure conditions* which can be used to define the set of all consequences of a theory. Instead of defining a 'closure of a theory', we will give a name to a theory which is equal to its closure. A **stable theory** is an autoepistemic theory satisfying the following conditions:

(a) If $q_1, \ldots, q_n \in T$ and $\{q_1, \ldots, q_n \models_{PL} q$, then $q \in T$.

(b) If $q \in T$, then $\Box q \in T$.

(c) If $q \notin T$, then $\neg \Box q \in T$.

It can be proved that an autoepistemic theory is semantically complete iff it is stable. However, stability is silent about what is not believed; hence, soundness with respect to a set of premises cannot be achieved under mere stability. The initial premises must be included somewhere. Suppose that A is a set of premises (formulas). A theory T is **grounded** in A if

$$T \subseteq \{q : A \cup \{\Box p : p \in T\} \cup \{\neg \Box p : p \notin T\} \models_{PL} q\}$$

That is, T is grounded in A if T is stable and it does not contain any formula which is not a PL-consequence of a *stable extension* of A.

It can be shown that a theory T is sound with respect to A iff T is grounded in A. Now, using both soundness and completeness, we arrive at the notion of a stable expansion of a theory. An autoepistemic theory T is a **stable expansion** of a set of premises A iff T is a superset of A that is stable and grounded in A, i.e.,

$$T = \{q : A \cup \{\Box p : p \in T\} \cup \{\neg \Box p : p \notin T\} \models_{PL} q\}$$

The notion of a stable expansion is, in fact, the set of all autoepistemic consequents of the premises in A. That is, whatever that can be believed when all the formulas in A are believed are in the stable expansion. Does there exist always a stable expansion? Is it unique?

***EXAMPLE* 7.4** Let $A = \{\neg \Box p \rightarrow p : p \text{ is any formula}\}$ and T be a stable autoepistemic theory that contains the formula $\neg \Box p \rightarrow p$. If $p \notin T$, then due to stability, $\neg \Box p \in T$. Since $\neg \Box p \rightarrow p \in T$, we have $p \in T$. Hence, $p \in T$. If T is grounded in A, then T contains no more formulas than the PL-consequences of

$$\{\neg \Box p \rightarrow p : p \text{ is any formula}\} \cup \{\Box p : p \in T\} \cup \{\neg \Box p : p \notin T\}$$

Since $p \in T$, $\Box p \in T$, we have $\neg \Box p \notin T$; otherwise, we would have $p \notin T$, resulting in a contradiction. However, $\neg \Box p \notin T$ implies that $\neg \Box \neg \Box p \in T$. Again, $\neg \Box \neg \Box p \rightarrow \neg \Box p \in A$ implies $\neg \Box p \in T$ as a PL-consequence. Thus, we arrive at a contradiction that $\neg \Box p \in T$ and $\neg \Box p \notin T$. Hence T cannot be grounded in A. That is, there is no stable expansion of T in A.

***EXAMPLE* 7.5** Let $A = \{\neg \Box p \rightarrow q, \neg \Box q \rightarrow p\}$. Let T be any stable expansion in A. The first element of A says that if p is not believed then q holds, and the second one asserts that if q is not believed then p holds. Now, if $p \notin T$, then $q \in T$ and if $q \notin T$, then $p \in T$. Thus, T contains either p or q. Then there can be two stable expansions of T in A, one containing p and not q, and the other containing q and not p.

These examples show that stable expansions may not exist, and even if one exists, it need not be the only stable expansion. Note that a stable expansion in A formalizes the notion of all consequences of the premises in A. Thus, even if a theory is consistent, there may not be any consequence, and even if there are consequences, the set of all consequences need not be unique. The status of theoremhood with axioms in A is doubtful. However, this is pragmatic since belief in some statements need not conclusively guarantee other beliefs. An alternative is to regard the intersection of all stable expansions as the set of all beliefs one may have. That is, any statement which is in every stable expansion of a theory in a set of premises A can be regarded as theorems of A. This will be the view of an external observer about the agent. Also, in such a case, an empty intersection will lead to a definition of an inconsistent autoepistemic theory grounded in a set of premises.

SUMMARY

In this chapter you have been exposed to some logics which have not yet come into the mainstream, hence the name, Nonstandard Logics. There are monographs and also textbooks, though very few in number, which deal with one or more of these logics. However, each of them has received considerable attention by AI researchers. You will find more logics of this sort in AI applications. You can start from the edited text [69] to get an overall picture of nonstandard logics having some general frameworks and then go through the topics that interest you in [27]. Many more logics have been constructed for understanding specific problem domains. To get a comprehensive source of materials, the best way is to search the web. This chapter gives you a summary of some of those logics; so it does not really require another summary. However, this is also the concluding chapter of this book; so we would rather say, your study of logic begins now. To get the summary of this book, solve the following problems.

PROBLEMS

1. Read Sections 1.7, 2.9, 3.7, 4.8, 5.8, 6.8, and 7.8 and write a summary.

2. How do you answer a layman's question: "What is logic all about?"

3. How to make teaching and learning of logic more interesting?

4. What is your plan for deeper study of logic and logic engineering?

References

[1] P.B. Andrews, Resolution in Type Theory, *J. Symbolic Logic*, Vol.36, pp.414−432, 1970.

[2] K.R. Apt and E.-R. Olderog, *Verification of Sequential and Concurrent Programs*, Springer-Verlag, New York, 1991.

[3] R.C. Backhouse, *Program Construction and Verification*, Prentice Hall Series in Computer Science, Prentice Hall, Englewood Cliffs, N.J., 1986.

[4] J. Barwise and J. Etchemendy, *Language, Proof and Logic*, CSLI Publications, Stanford, 1999. (Distributor: Cambridge University Press), See also http://www.csli.stanford.edu/hp/Logic-software.html for accompanying beautiful softwares.

[5] G. Boole, *An Investigation of the Laws of Thought*, Walton & Maberley, London, 1854, Reprinted by Dover, New York, 1951.

[6] C. Chang and R.C. Lee, *Symbolic Logic and Mechanical Theorem Proving*, Academic Press, New York, 1973.

[7] B.F. Chellas, *Modal Logic − An introduction*, Cambridge University Press, Cambridge, 1980.

[8] I.M. Copi, Symbolic Logic, Prentice Hall of India, New Delhi, 2001.

[9] M.D. Davis and H. Putnam, A Computing Procedure for Quantification Theory, *J. ACM*, Vol.7, pp.210−215, 1960.

[10] E.W. Dijkstra, *A Discipline of Programming*, Prentice Hall series in Automatic Computation, Prentice Hall, Englewood Cliffs, N.J., 1976.

[11] E.W. Dijkstra, *Selected Writings on Computing: A personal perspective*, Springer-Verlag Texts and Monographs in Computer Science, Springer-Verlag, New York, 1982.

[12] J. Doyle, E. Sandewall and P. Torassi, (Ed), KR'94, *Proc. Fourth International Conference on Principles of Knowledge Representation and Reasoning*, Morgan Kaufmann, San Fransisco, 1994.

[13] W.F. Dowling and J.H. Gallier, Linear Time Algorithms for Testing the Satisfiability of Propositional Horn Formulas, *Journal of Logic Programming*, Vol.3, pp.267−284, 1984.

[14] D. Du, J. Gu and P.M. Pardalos, (Ed.), *Satisfiability Problem : Theory and applications*, AMS DIMACS Series : Vol.35, American Mathematical Society, Providence, 1997.

[15] D. Dubois, H. Farreny and H. Prade, Sur divers problèms inhérents á l'automatisation des raisonnements de sens commun, *Congrs AFCET-RFIA, Grenoble*, Vol.1, pp.321−328, Brigitte Biébow, Paris, 1985.

[16] M. Dummett, *Elements of Intuitionism*, Clarendon Press, Oxford, 1977.

[17] H.B. Enderton, *A Mathematical Introduction to Logic*, Academic Press, New York, 1972.

[18] R.L. Epstein, *Propositional Logics: The semantic foundations of logic*, Wadsworth, Belmont, USA, 2001.

[19] Yu.L. Ershov and E.A. Palyutin, *Mathematical Logic*, Mir Publishers, Moscow, 1984.

[20] R. Fagin, J.Y. Halpern, Y. Moses and M.Y. Vardi, *Reasoning about Knowledge*, MIT Press, Cambridge (Mass.), 1995.

[21] M. Fitting, Basic Modal Logic, In: *Handbook of Logic in Artificial Intelligence and Logic Programming*, Vol.I, D. Gabbay, C. Hogger and J. Robinson (Eds.), Oxford University Press, Oxford, 1993.

[22] M. Fitting, *First Order Logic and Automated Theorem Proving*, Springer-Verlag, New York, 1996.

[23] M. Fitting and R.L. Mendelsohn, *First Order Modal Logic*, Kluwer Academic Publishers, Dordrecht, The Netherlands, 1998.

[24] N. Francez, *Program Verification*, Addison-Wesley, Reading (Mass.), 1992.

[25] G. Frege, *Die Grundlagen der Arithmetik, Eine logischmathematische Untersuchung ber der Begriff der Zahl.*, Breslau, Reprinted Marcus, Breslau,1934. (See [80] for an English translation.)

[26] G. Frege, *Collected Papers on Mathematics, Logic, and Philosophy*, edited by B.Mc. Guninness, Basil Blackwell, Oxford, 1984.

[27] D.M. Gabbay and F. Guenthner (Ed.), *Handbook of Philosophical Logic*, 2nd ed., Vols.1−9, Kluwer Academic Publishers, Dordrecht, The Netherlands, 2002.

[28] J.H. Gallier, *Logic for Computer Science: Foundations of automatic theorem proving*, John Wiley & Sons, New York, 1987.

[29] M.R. Garey and D.S. Johnson, *Computers and Intractability: A guide to the theory of* $\mathcal{NP}-$*completeness*, Freeman, New York, 1979.

[30] G. Gentzen, Die Widerspruchsfreiheit der reinen Zahlentheorie, *Mathemaische Annalen*, Vol.12, pp.493−565, 1936.

[31] V. Glivenko, Sur quelques points de la logique de M. Brouwer, In: *Académie Royale de Belgique, Bulletins de la classe des sciences, Ser. 5*, Vol.15, pp.183−188, 1929.

[32] R. Goldblatt, *Logics of Time and Computation*, CSLI Lecture Notes, CSLI Publications, Stanford, 1987.

[33] D. Gries, *The Science of Programming*, Springer-Verlag, New York, 1981.

[34] D. Gries, A Note on a Standard Strategy for Developing Loop Invariants and Loops, *Science of Computer Programming*, Vol.2, pp.207−214, 1982.

[35] D. Gries and F.B. Schneider, *A Logical Approach to Discrete Math*, Springer-Verlag, New York, 1993.

[36] A.G. Hamilton, *Logic for Mathematicians*, Cambridge University Press, Cambridge, 1978.

[37] E.C.R. Hehner, *The Logic of Programming*, Prentice Hall Series in Computer Science, Prentice Hall, Englewood Cliffs, N.J., 1984.

[38] C.A.R. Hoare, An Axiomatic Basis for Computer Programming, *Comm. ACM*, Vol.12, pp.576−580,583, 1969.

[39] W. Hodges, *Elementary Predicate Logic*, In: Vol.1 of [27], 2002.

[40] G.E. Hughes and M.J. Cresswell, *An Introduction to Modal Logic*, Methuen, New York, 1971.

[41] M. Huth and M. Ryan, *Logic in Computer Science: Modelling and reasoning about systems*, Cambridge University Press, Cambridge, 2000.

[42] C.B. Jones, *Software Development: A rigorous approach*, Prentice Hall series in Computer Science, Prentice Hall, Englewood Cliffs, N.J., 1980.

[43] R. Kowalski, *Logic for Problem Solving*, Artificial Intelligence Series, Computer Science Library, Elsevier, Amsterdam, 1979.

[44] E.J. Lemmon, *An Introduction to Modal Logic*, edited by N. Rescher, Monograph No.11, American Philosophical Quaterly Monograph series, Basil Blackwell, Oxford, 1977.

[45] C.I. Lewis, *A Survey of Symbolic Logic*, Dover, New York, 1918.

[46] C.I. Lewis and C.H. Langford, *Symbolic Logic*, Century, New York, 1938.

[47] H.R. Lewis and C.H. Papadimitriou, *Elements of the Theory of Computation*, 2nd ed., Prentice Hall of India, New Delhi, 2003.

[48] D.W. Loveland, *Automated Theorem Proving: A logical basis*, Elsevier, Amsterdam, 1979.

[49] J. Łukasiewicz, *Selected Works*, edited by L. Borkowski, North Holland, Amsterdam, 1970.

[50] A.B. Manaster, *Completeness,Compactness and Undecidability: An introduction to mathematical logic*, Prentice Hall of India, New Delhi, 1978.

[51] B. Mates, *Elementary Logic*, Oxford University Press, New York, 1972.

[52] E. Mendelson, *Introduction to Mathematical Logic*, D. Van Nostrand, New York, 1979.

[53] J.-J.Ch. Meyer and W. van der Hoek, *Epistemic Logic for AI and Computer Science*, Vol.41 of Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, Cambridge, 1993.

[54] L.C. Paulson, *ML for the Working Programmer*, Cambridge University Press, Cambridge, 1991.

[55] S. Popkorn, *First Steps in Modal Logic*, Cambridge University Press, 1994.

[56] E.L. Post, Introduction to General Theory of Elementary Propositions, *American Journal of Mathematics*, Vol.43, pp.163−185, Reprinted in [80], pp.264−283.

[57] D. Pretolani, *Satisfiability and Hypergraphs*, Ph.D. Thesis, TD-12/93, Universit di Pisa, Pisa (Italy), 1993.

[58] W.V. Quine, On Cores and Prime Implicates of Truth Functions, *American Math. Monthly*, Vol.66, pp.755−760, 1959.

[59] R. Reiter, J. de Kleer, Foundations of Assumption Based Truth Maintenance Systems : Preliminary Report, *Proc. AAAI−87*, pp.183−188, 1987.

[60] J.C. Reynolds, *The Craft of Programming*, Prentice Hall Series in Computer Science, Prentice Hall, Englewood Cliffs, N.J., 1981.

[61] J.A. Robinson, A Machine Oriented Logic Based on the Resolution Principle, *J. ACM*, Vol.12, pp.23−41, 1965.

[62] J.A. Robinson, *Logic: Form and function*, Elsevier, New York, 1979.

[63] J.A. Robinson and L. Wos, Paramodulation and Theorem Proving in First Order Logic with Equality, *Machine Intelligence*, Vol.4, pp.135-150, 1969.

[64] D.A. Schmidt, *The Structure of Typed Programming Languages*, Foundations of Computing Series, MIT Press, Cambridge (Mass.), 1994.

[65] B. Selman and H. Kautz, Knowledge Compilation using Horn Approximation, *Proc. Ninth National Conference on Artificial Intelligence*, Anaheim, CA, July 1991.

[66] J.R. Shoenfield, *Mathematical Logic*, Addison Wesley, Reading (Mass.), 1967.

[67] A. Singh and C. Goswami, *Fundamentals of Logic*, Indian Council of Philosophical Research, New Delhi, 1998.

[68] A. Singh, Computing Prime Implicants via Transversal Clauses, *Int. J. Computer Math.*, Vol.70, pp.417−427, 1999.

[69] P. Smets, A. Mamdani, D. Dubois and H. Prade, *Non-standard Logics for Automated Reasoning*, Academic Press, New York, 1988.

[70] R.R. Stoll, Set Theory and Logic, W.H. Freeman, New York, 1963.

[71] R. Smullyan, *First Order Logic*, Springer-Verlag, New York, 1968.

[72] R. Smullyan, *What is The Name of This Book?*, Prentice Hall, Englewood Cliffs, N.J., 1978.

[73] R. Smullyan, *The Lady or the Tiger and the Other Logical Puzzles*, Penguin Books, Harmondsworth (UK), 1983.

[74] D.P. Snyder, *Modal Logic and its Applications*, Van Nostrand Reinhold Company, New York, 1971.

[75] V. Sperchneider and G. Antoniou, *Logic, a Foundation for Computer Science*, Addison Wesley, Reading (Mass.), 1991.

[76] R.D. Tennent, *Semantics of Programming Languages*, Prentice Hall, Englewood Cliffs, N.J., 1991.

[77] P. Tison, Generalization of Consensus Theory and Application to the Minimization of Boolean Functions, *IEEE Trans. on Elec. Comp.*, Vol.EC-16, No.4, pp.446−456, 1967.

[78] R. Turner, *Constructive Foundations for Functional Languages*, McGraw-Hill, New York, 1991.

[79] D. van Dalen, *Logic and Structure*, Universitext, Springer-Verlag, 1989.

[80] J. van Heijenoort, (Ed.), *From Frege to Gödel: A source book in mathematical logic 1879−1931*, Harvard University Press, Cambridge (Mass.), 1967.

[81] L. Wittgenstein, *Tractatus Logico Philosophicus*, Translated by C.K. Ogden, Routledge & Kegan Paul, London, 1922 (see also http://www.kfs.org/ jonathan/witt/tlph.html).

[82] R. Wójcicki, *Theory of Logical Calculi*, D. Reidel, Dordrecht, The Netherlands, 1988.

Index