Chapter 3

Propositional Logic

3.1 Introduction

Every logic comprises a (formal) language for making statements about objects and reasoning about properties of these objects. This view of logic is very general and actually we will restrict our attention to mathematical objects, programs, and data structures in particular. Statements in a logical language are constructed according to a predefined set of formation rules (depending on the language) called *syntax rules*.

One might ask why a special language is needed at all, and why English (or any other natural language) is not adequate for carrying out logical reasoning. The first reason is that English (and any natural language in general) is such a rich language that it cannot be formally described. The second reason, which is even more serious, is that the meaning of an English sentence can be ambiguous, subject to different interpretations depending on the context and implicit assumptions. If the object of our study is to carry out precise rigorous arguments about assertions and proofs, a precise language whose syntax can be completely described in a few simple rules and whose semantics can be defined unambiguously is required.

Another important factor is conciseness. Natural languages tend to be verbose, and even fairly simple mathematical statements become exceedingly long (and unclear) when expressed in them. The logical languages that we shall define contain special symbols used for abbreviating syntactical constructs.

A logical language can be used in different ways. For instance, a language can be used as a *deduction system* (or proof system); that is, to construct proofs or refutations. This use of a logical language is called *proof theory*. In this case, a set of facts called axioms and a set of deduction rules (inference rules) are given, and the object is to determine which facts follow from the axioms and the rules of inference. When using logic as a proof system, one is not concerned with the meaning of the statements that are manipulated, but with the arrangement of these statements, and specifically, whether proofs or refutations can be constructed. In this sense, statements in the language are viewed as cold facts, and the manipulations involved are purely mechanical, to the point that they could be carried out by a computer. This does not mean that finding a proof for a statement does not require creativity, but that the interpetation of the statements is irrelevant. This use of logic is similar to game playing. Certain facts and rules are given, and it is assumed that the players are perfect, in the sense that they always obey the rules. Occasionally, it may happen that following the rules leads to inconsistencies, in which case it may be necessary to revise the rules.

However, the statements expressed in a logical language often have an intended meaning. The second use of a formal language is for expressing statements that receive a meaning when they are given what is called an interpretation. In this case, the language of logic is used to formalize properties of structures, and determine when a statement is true of a structure. This use of a logical language is called *model theory*.

One of the interesting aspects of model theory is that it forces us to have a precise and rigorous definition of the concept of truth in a structure. Depending on the interpretation that one has in mind, truth may have quite a different meaning. For instance, whether a statement is true or false may depend on parameters. A statement true under all interpretations of the parameters is said to be valid. A useful (and quite reasonable) mathematical assumption is that the truth of a statement can be obtained from the truth (or falsity) of its parts (substatements). From a technical point of view, this means that the truth of a statement is defined by recursion on the syntactical structure of the statement. The notion of truth that we shall describe (due to Tarski) formalizes the above intuition, and is firmly justified in terms of the concept of an algebra presented in Section 2.4 and the unique homomorphic extension theorem (theorem 2.4.1).

The two aspects of logic described above are actually not independent, and it is the interaction between model theory and proof theory that makes logic an interesting and effective tool. One might say that model theory and proof theory form a couple in which the individuals complement each other. To illustrate this point, consider the problem of finding a procedure for listing all statements true in a certain class of stuctures. It may be that checking the truth of a statement requires an infinite computation. Yet, if the class of structures can be axiomatized by a finite set of axioms, we might be able to find a proof procedure that will give us the answer.

Conversely, suppose that we have a set of axioms and we wish to know whether the resulting theory (the set of consequences) is consistent, in the sense that no statement and its negation follow from the axioms. If one discovers a structure in which it can be shown that the axioms and their consequences are true, one will know that the theory is consistent, since otherwise some statement and its negation would be true (in this structure).

To summarize, a logical language has a certain *syntax*, and the meaning, or *semantics*, of statements expressed in this language is given by an interpretation in a structure. Given a logical language and its semantics, one usually has one or more *proof systems* for this logical system.

A proof system is acceptable only if every provable formula is indeed valid. In this case, we say that the proof system is *sound*. Then, one tries to prove that the proof system is *complete*. A proof system is complete if every valid formula is provable. Depending on the complexity of the semantics of a given logic, it is not always possible to find a complete proof system for that logic. This is the case, for instance, for second-order logic. However, there are complete proof systems for propositional logic and first-order logic. In the first-order case, this only means that a procedure can be found such that, if the input formula is valid, the procedure will halt and produce a proof. But this does not provide a decision procedure for validity. Indeed, as a consequence of a theorem of Church, there is no procedure that will halt for every input formula and decide whether or not a formula is valid.

There are many ways of proving the completeness of a proof system. Oddly, most proofs establishing completeness only show that if a formula A is valid, then there *exists* a proof of A. However, such arguments do not actually yield a method for *constructing* a proof of A (in the formal system). Only the existence of a proof is shown. This is the case in particular for so-called *Henkin proofs*. To illustrate this point in a more colorful fashion, the above situation is comparable to going to a restaurant where you are told that excellent dinners exist on the menu, but that the inexperienced chef does not know how to prepare these dinners. This may be satisfactory for a philosopher, but not for a hungry computer scientist! However, there is an approach that does yield a procedure for constructing a formal proof of a formula if it is valid. This is the approach using Gentzen systems (or tableaux systems). Furthermore, it turns out that all of the basic theorems of first-order logic can be obtained using this approach. Hence, this author feels that a student (especially a computer scientist) has nothing to lose, and in fact will reap extra benefits by learning Gentzen systems first.

Propositional logic is the system of logic with the simplest semantics. Yet, many of the concepts and techniques used for studying propositional logic generalize to first-order logic. Therefore, it is pedagogically sound to begin by studying propositional logic as a "gentle" introduction to the methods used in first-order logic.

In propositional logic, there are atomic assertions (or atoms, or propositional letters) and compound assertions built up from the atoms and the logical connectives, *and*, *or*, *not*, *implication* and *equivalence*. The atomic facts are interpreted as being either true or false. In propositional logic, once the atoms in a proposition have received an interpretation, the truth value of the proposition can be computed. Technically, this is a consequence of the fact that the set of propositions is a freely generated inductive closure. Certain propositions are true for all possible interpretations. They are called *tautologies*. Intuitively speaking, a tautology is a *universal truth*. Hence, tautologies play an important role.

For example, let "John is a teacher," "John is rich," and "John is a rock singer" be three atomic propositions. Let us abbreviate them as A,B,C. Consider the following statements:

"John is a teacher";

It is false that "John is a teacher" and "John is rich";

If "John is a rock singer" then "John is rich."

We wish to show that the above assumptions imply that

It is false that "John is a rock singer."

This amounts to showing that the (formal) proposition

($*$) (A and not(A and B) and (C implies B)) implies (not C)

is a tautology. Informally, this can be shown by contradiction. The statement ($*$) is false if the premise (A and not(A and B) and (C implies B)) is true and the conclusion (not C) is false. This implies that C is true. Since C is true, then, since (C implies B) is assumed to be true, B is true, and since A is assumed to be true, (A and B) is true, which is a contradiction, since not(A and B) is assumed to be true.

Of course, we have assumed that the reader is familiar with the semantics and the rules of propositional logic, which is probably not the case. In this chapter, such matters will be explained in detail.

3.2 Syntax of Propositional Logic

The syntax of propositional logic is described in this section. This presentation will use the concept of an *inductive closure* explained in Section 2.3, and the reader is encouraged to review it.

3.2.1 The Language of Propositional Logic

Propositional formulae (or propositions) are strings of symbols from a countable alphabet defined below, and formed according to certain rules stated in definition 3.2.2.

Definition 3.2.1 (The alphabet for propositional formulae) This alphabet consists of:

(1) A countable set **PS** of *proposition symbols*: P_0,P_1,P_2...;

(2) The *logical connectives*: $\wedge$ (and), $\vee$ (or), $\supset$ (implication), $\neg$ (not), and sometimes $\equiv$ (equivalence) and the constant $\perp$ (false);

(3) *Auxiliary symbols*: "(" (left parenthesis), ")" (right parenthesis).

The set $PROP$ of propositional formulae (or propositions) is defined as the inductive closure (as in Section 2.3) of a certain subset of the alphabet of definition 3.2.1 under certain operations defined below.

Definition 3.2.2 Propositional formulae. The set $PROP$ of *propositional formulae* (or *propositions*) is the inductive closure of the set $\mathbf{PS} \cup \{\perp\}$ under the functions $C_\neg$, $C_\wedge$, $C_\vee$, $C_\supset$ and $C_\equiv$, defined as follows: For any two strings A, B over the alphabet of definition 3.2.1,

$$\begin{aligned} C_\neg(A) &= \neg A, \\ C_\wedge(A,B) &= (A \wedge B), \\ C_\vee(A,B) &= (A \vee B), \\ C_\supset(A,B) &= (A \supset B) \textit{ and} \\ C_\equiv(A,B) &= (A \equiv B). \end{aligned}$$

The above definition is the official definition of $PROP$ as an inductive closure, but is a bit formal. For that reason, it is often stated less formally as follows:

The set $PROP$ of propositions is the smallest set of strings over the alphabet of definition 3.2.1, such that:

(1) Every proposition symbol P_i is in $PROP$ and $\perp$ is in $PROP$;

(2) Whenever A is in $PROP$, $\neg A$ is also in $PROP$;

(3) Whenever A, B are in $PROP$, $(A \vee B)$, $(A \wedge B)$, $(A \supset B)$ and $(A \equiv B)$ are also in $PROP$.

(4) A string is in $PROP$ only if it is formed by applying the rules (1),(2),(3).

The official inductive definition of $PROP$ will be the one used in proofs.

3.2.2 Free Generation of PROP

The purpose of the parentheses is to ensure unique readability; that is, to ensure that $PROP$ is freely generated on **PS**. This is crucial in order to give a proper definition of the semantics of propositions. Indeed, the meaning of a proposition will be given by a function defined recursively over the set $PROP$, and from theorem 2.4.1 (in the Appendix), we know that such a function exists and is unique when an inductive closure is freely generated.

There are other ways of defining the syntax in which parentheses are unnecessary, for example the prefix (or postfix) notation, which will be discussed later.

It is necessary for clarity and to avoid contradictions to distinguish between the formal language that is the object of our study (the set $PROP$ of propositions), and the (informal) language used to talk about the object of study. The first language is usually called the *object language* and the second, the *meta-language*. It is often tedious to maintain a clear notational distinction between the two languages, since this requires the use of a formidable number of symbols. However, one should always keep in mind this distinction to avoid confusion (and mistakes !).

For example, the symbols P, Q, R, ... will usually range over propositional symbols, and the symbols A, B, C, ... over propositions. Such symbols are called *meta-variables*.

Let us give a few examples of propositions:

EXAMPLE 3.2.1

The following strings are propositions.

$$P_1, \qquad P_2, \qquad (P_1 \lor P_2),$$

$$((P_1 \supset P_2) \equiv (\neg P_1 \lor P_2)), \qquad (\neg P_1 \equiv (P_1 \supset \perp)),$$

$$(((P_1 \supset P_2) \land \neg P_2) \supset \neg P_1), \qquad (P_1 \lor \neg P_1).$$

On the other hand, strings such as

$$(()), \text{ or } \qquad (P_1 \lor P_2)\land$$

are not propositions, because they cannot be constructed from **PS** and $\perp$ and the logical connectives.

Since $PROP$ is inductively defined on **PS**, the induction principle (of Section 2.3) applies. We are now going to use this induction principle to show that $PROP$ is freely generated by the propositional symbols (and $\perp$) and the logical connectives.

Lemma 3.2.1 (i) Every proposition in $PROP$ has the same number of left and right parentheses.

(ii) Any proper prefix of a proposition is either the empty string, a (nonempty) string of negation symbols, or it contains an excess of left parentheses.

(iii) No proper prefix of a proposition can be a proposition.

Proof: (i) Let S be the set of propositions in $PROP$ having an equal number of left and right parentheses. We show that S is inductive on the set of propositional symbols and $\perp$. By the induction principle, this will show that $S = PROP$, as desired. It is obvious that S contains the propositional symbols (no parentheses) and $\perp$. It is also obvious that the rules in definition 3.2.2 introduce matching parentheses and so, preserve the above property. This concludes the first part of the proof.

(ii) Let S be the set of propositions in $PROP$ such that any proper prefix is either the empty string, a string of negations, or contains an excess of left parentheses. We also prove that S is inductive on the set of propositional symbols and $\perp$. First, it is obvious that every propositional symbol is in S, as well as $\perp$. Let us verify that S is closed under $C_\wedge$, leaving the other cases as an exercise. Let A and B be in S. The nonempty proper prefixes of $C_\wedge(A, B) = (A \wedge B)$ are:

$($
$(C$ where C is a proper prefix of A
$(A$
$(A\wedge$
$(A \wedge C$ where C is a proper prefix of B
$(A \wedge B$

Applying the induction hypothesis that A and B are in S, we obtain the desired conclusion.

Clause (iii) of the lemma follows from the two previous properties. If a proper prefix of a proposition is a proposition, then by (i), it has the same number of left and right parentheses. If a proper prefix has no parentheses, it is either the empty string or a string of negations, but neither is a proposition. If it has parentheses, by property (ii), it has an excess of left parentheses, a contradiction. $\square$

The above lemma allows us to show the theorem:

Theorem 3.2.1 The set $PROP$ of propositions is freely generated by the propositional symbols in **PS**, $\perp$, and the logical connectives.

Proof: First, we show that the restrictions of the functions $C_\neg$, $C_\wedge$, $C_\vee$, $C_\supset$ and $C_\equiv$ to $PROP$ are injective. This is obvious for $C_\neg$ and we only check this for $C_\wedge$, leaving the other cases as an exercise. If $(A \wedge B) = (C \wedge D)$, then $A \wedge B) = C \wedge D)$. Either $A = C$, or A is a proper prefix of C, or C is a proper prefix of A. But the last two cases are impossible by lemma 3.2.1. Then $\wedge B) = \wedge D)$, which implies $B = D$.

Next, we have to show that the ranges of the restrictions of the above functions to $PROP$ are disjoint. We only discuss one case, leaving the others as an exercise. For example, if $(A \wedge B) = (C \supset D)$, then $A \wedge B) = C \supset D)$. By the same reasoning as above, we must have $A = C$. But then, we must have $\wedge = \supset$, which is impossible. Finally, since all the functions yield a string of length greater than that of its arguments, all the conditions for being freely generated are met. $\square$

The above result allows us to define functions over $PROP$ recursively. Every function with domain $PROP$ is uniquely determined by its restriction to the set **PS** of propositional symbols and to $\perp$. We are going to use this fact in defining the semantics of propositional logic. As an illustration of theorem 3.2.1, we give a recursive definition of the set of propositional letters occurring in a proposition.

EXAMPLE 3.2.2

The function $symbols : PROP \to 2^{\mathbf{PS}}$ is defined recursively as follows:

$$\begin{aligned} symbols(\perp) &= \emptyset, \\ symbols(P_i) &= \{P_i\}, \\ symbols((B * C)) &= symbols(B) \cup symbols(C), \ \text{for } * \in \{\wedge, \vee, \supset, \equiv\}, \\ symbols(\neg A) &= symbols(A). \end{aligned}$$

For example,

$$symbols(((P_1 \supset P_2) \vee \neg P_3) \wedge P_1)) = \{P_1, P_2, P_3\}.$$

In order to minimize the number of parentheses, a *precedence* is assigned to the logical connectives and it is assumed that they are left associative. Starting from highest to lowest precedence we have:

$\neg$

$\wedge$

$\vee$

$\supset, \equiv$.

EXAMPLE 3.2.3

$A \wedge B \supset C$ is an abbreviation for $((A \wedge B) \supset C)$,

$A \vee B \wedge C$ an abbreviation for $(A \vee (B \wedge C))$, and

$A \vee B \vee C$ is an abbreviation for $((A \vee B) \vee C)$.

Parentheses can be used to disambiguate expressions. These conventions are consistent with the semantics of the propositional calculus, as we shall see in Section 3.3.

Another way of avoiding parentheses is to use the *prefix notation*. In prefix notation, $(A \lor B)$ becomes $\lor AB$, $(A \land B)$ becomes $\land AB$, $(A \supset B)$ becomes $\supset AB$ and $(A \equiv B)$ becomes $\equiv AB$.

In order to justify the legitimacy of the prefix notation, that is, to show that the set of propositions in prefix notation is freely generated, we have to show that every proposition can be written in a unique way. We shall come back to this when we consider terms in first-order logic.

PROBLEMS

3.2.1. Let $PROP$ be the set of all propositions over the set $\mathbf{PS}$ of propositional symbols. The *depth* $d(A)$ of a proposition A is defined recursively as follows:

$$\begin{aligned}
d(\perp) &= 0, \\
d(P) &= 0, \text{ for each symbol } P \in \mathbf{PS}, \\
d(\neg A) &= 1 + d(A), \\
d(A \lor B) &= 1 + max(d(A), d(B)), \\
d(A \land B) &= 1 + max(d(A), d(B)), \\
d(A \supset B) &= 1 + max(d(A), d(B)), \\
d(A \equiv B) &= 1 + max(d(A), d(B)).
\end{aligned}$$

If PS_i is the i-th stage of the inductive definition of $PROP = (\mathbf{PS} \cup \{\perp\})_+$ (as in Section 2.3), show that PS_i consists exactly of all propositions of depth less than or equal to i.

3.2.2. Which of the following are propositions? Justify your answer.

$\neg\neg\neg P_1$

$\neg P_1 \lor \neg P_2$

$\neg(P_1 \lor P_2)$

$(\neg P_1 \supset \neg P_2)$

$\neg(P_1 \lor (P_2 \land (P_3 \lor P_4) \land (P_1 \land (P_3 \land \neg P_1) \lor (P_4 \lor P_1)))$

(*Hint*: Use problem 3.2.1, lemma 3.2.1.)

3.2.3. Finish the proof of the cases in lemma 3.2.1.

3.2.4. The function $sub : PROP \to 2^{PROP}$ which assigns to any proposition A the set $sub(A)$ of all its subpropositions is defined recursively as follows:

$$\begin{aligned}
sub(\perp) &= \{\perp\},\\
sub(P_i) &= \{P_i\}, \text{ for a propositional symbol } P_i,\\
sub(\neg A) &= sub(A) \cup \{\neg A\},\\
sub((A \lor B)) &= sub(A) \cup sub(B) \cup \{(A \lor B)\},\\
sub((A \land B)) &= sub(A) \cup sub(B) \cup \{(A \land B)\},\\
sub((A \supset B)) &= sub(A) \cup sub(B) \cup \{(A \supset B)\},\\
sub((A \equiv B)) &= sub(A) \cup sub(B) \cup \{(A \equiv B)\}.
\end{aligned}$$

Prove that if a proposition A has n connectives, then $sub(A)$ contains at most $2n + 1$ propositions.

3.2.5. Give an example of propositions A and B and of strings u and v such that $(A \lor B) = (u \lor v)$, but $u \neq A$ and $v \neq B$. Similarly give an example such that $(A \lor B) = (u \land v)$, but $u \neq A$ and $v \neq B$.

$*$ **3.2.6.** The set of propositions can be defined by a context-free grammar, provided that the propositional symbols are encoded as strings over a finite alphabet. Following Lewis and Papadimitriou, 1981, the symbol P_i, $(i \geq 0)$ will be encoded as $PI...I\$$, with a number of I's equal to i. Then, $PROP$ is the language $L(G)$ defined by the following context-free grammar $G = (V, \Sigma, R, S)$:

$\Sigma = \{P, I, \$, \land, \lor, \supset, \equiv, \neg, \perp\}$, $V = \Sigma \cup \{S, N\}$,

$$\begin{aligned}
R = \{&N \to e,\\
&N \to NI,\\
&S \to PN\$,\\
&S \to \perp,\\
&S \to (S \lor S),\\
&S \to (S \land S),\\
&S \to (S \supset S),\\
&S \to (S \equiv S),\\
&S \to \neg S\}.
\end{aligned}$$

Prove that the grammar G is unambiguous.

Note: The above language is actually SLR(1). For details on parsing techniques, consult Aho and Ullman, 1977.

3.2.7. The set of propositions in prefix notation is the inductive closure of $\mathbf{PS} \cup \{\perp\}$ under the following functions:

For all strings A, B over the alphabet of definition 3.2.1, excluding parentheses,

$$C_\wedge(A,B) = \wedge AB,$$
$$C_\vee(A,B) = \vee AB,$$
$$C_\supset(A,B) = \supset AB,$$
$$C_\equiv(A,B) = \equiv AB,$$
$$C_\neg(A) = \neg A.$$

In order to prove that the set of propositions in prefix notation is freely generated, we define the function K as follows:

$K(\wedge) = -1$; $K(\vee) = -1$; $K(\supset) = -1$; $K(\equiv) = -1$; $K(\neg) = 0$; $K(\perp) = 1$; $K(P_i) = 1$, for every propositional symbol P_i.

The function K is extended to strings as follows: For every string $w_1 ... w_k$ (over the alphabet of definition 3.2.1, excluding parentheses), $K(w_1 ... w_k) = K(w_1) + ... + K(w_k)$.

(i) Prove that for any proposition A, $K(A) = 1$.

(ii) Prove that for any proper prefix w of a proposition, $K(w) \leq 0$.

(iii) Prove that no proper prefix of a proposition is a proposition.

(iv) Prove that the set of propositions in prefix notation is freely generated.

3.2.8. Suppose that we modify definition 3.2.2 by omitting all right parentheses. Thus, instead of

$$((P \wedge \neg Q) \supset (R \vee S)),$$

we have

$$((P \wedge \neg Q \supset (R \vee S.$$

Formally, we define the functions:

$$C_\wedge(A,B) = (A \wedge B,$$
$$C_\vee(A,B) = (A \vee B,$$
$$C_\supset(A,B) = (A \supset A,$$
$$C_\equiv(A,B) = (A \equiv A,$$
$$C_\neg(A) = \neg A.$$

Prove that the set of propositions defined in this fashion is still freely generated.

3.3 Semantics of Propositional Logic

In this section, we present the semantics of propositional logic and define the concepts of satisfiability and tautology.

3.3.1 The Semantics of Propositions

The semantics of the propositional calculus assigns a truth function to each proposition in $PROP$. First, it is necessary to define the meaning of the logical connectives. We first define the domain $BOOL$ of truth values.

Definition 3.3.1 The set of *truth values* is the set $BOOL = \{\mathbf{T}, \mathbf{F}\}$. It is assumed that $BOOL$ is (totally) ordered with $\mathbf{F} < \mathbf{T}$.

Each logical connective X is interpreted as a function H_X with range $BOOL$. The logical connectives are interpreted as follows.

Definition 3.3.2 The graphs of the logical connectives are represented by the following table:

P	Q	$H_\neg(P)$	$H_\wedge(P,Q)$	$H_\vee(P,Q)$	$H_\supset(P,Q)$	$H_\equiv(P,Q)$
T	**T**	**F**	**T**	**T**	**T**	**T**
T	**F**	**F**	**F**	**T**	**F**	**F**
F	**T**	**T**	**F**	**T**	**T**	**F**
F	**F**	**T**	**F**	**F**	**T**	**T**

The logical constant $\perp$ is interpreted as **F**.

The above table is what is called a *truth table*. We have introduced the function H_X to distinguish between the symbol X and its meaning H_X. This is a heavy notational burden, but it is essential to distinguish between syntax and semantics, until the reader is familiar enough with these concepts. Later on, when the reader has assimilated these concepts, we will often use X for H_X to simplify the notation.

We now define the semantics of formulae in $PROP$.

Definition 3.3.3 A *truth assignment* or *valuation* is a function $v : \mathbf{PS} \rightarrow BOOL$ assigning a truth value to all the propositional symbols. From theorem 2.4.1 (in the Appendix), since $PROP$ is freely generated by $\mathbf{PS}$, every valuation v extends to a unique function $\widehat{v} : PROP \rightarrow BOOL$ satisfying the following clauses for all $A, B \in PROP$:

$$
\begin{aligned}
\widehat{v}(\perp) &= \mathbf{F}, \\
\widehat{v}(P) &= v(P), \text{ for all } P \in \mathbf{PS}, \\
\widehat{v}(\neg A) &= H_\neg(\widehat{v}(A)), \\
\widehat{v}((A \wedge B)) &= H_\wedge(\widehat{v}(A), \widehat{v}(B)), \\
\widehat{v}((A \vee B)) &= H_\vee(\widehat{v}(A), \widehat{v}(B)), \\
\widehat{v}((A \supset B)) &= H_\supset(\widehat{v}(A), \widehat{v}(B)), \\
\widehat{v}((A \equiv B)) &= H_\equiv(\widehat{v}(A), \widehat{v}(B)).
\end{aligned}
$$

In the above definition, the *truth value* $\widehat{v}(A)$ of a proposition A is defined for a truth assignment v assigning truth values to all propositional symbols, including infinitely many symbols not occurring in A. However, for any formula A and any valuation v, the value $\widehat{v}(A)$ only depends on the propositional symbols actually occurring in A. This is justified by the following lemma.

Lemma 3.3.1 For any proposition A, for any two valuations v and v' such that $v(P) = v'(P)$ for all proposition symbols occurring in A, $\widehat{v}(A) = \widehat{v'}(A)$.

Proof: We proceed by induction. The lemma is obvious for $\perp$, since $\perp$ does not contain any propositional symbols. If A is the propositional symbol P_i, since $v(P_i) = v'(P_i)$, $\widehat{v}(P_i) = v(P_i)$, and $\widehat{v'}(P_i) = v'(P_i)$, the Lemma holds for propositional symbols.

If A is of the form $\neg B$, since the propositional symbols occurring in B are the propositional symbols occurring in A, by the induction hypothesis, $\widehat{v}(B) = \widehat{v'}(B)$. Since

$$\widehat{v}(A) = H_\neg(\widehat{v}(B)) \quad \text{and} \quad \widehat{v'}(A) = H_\neg(\widehat{v'}(B)),$$

we have

$$\widehat{v}(A) = \widehat{v'}(A).$$

If A is of the form $(B * C)$, for a connective $* \in \{\vee, \wedge, \supset, \equiv\}$, since the sets of propositional letters occurring in B and C are subsets of the set of propositional letters occurring in A, the induction hypothesis applies to B and C. Hence,

$$\widehat{v}(B) = \widehat{v'}(B) \quad \text{and} \quad \widehat{v}(C) = \widehat{v'}(C).$$

But

$$\widehat{v}(A) = H_*(\widehat{v}(B), \widehat{v}(C)) = H_*(\widehat{v'}(B), \widehat{v'}(C)) = \widehat{v'}(A),$$

showing that

$$\widehat{v}(A) = \widehat{v'}(A).$$

□

Using lemma 3.3.1, we observe that given a proposition A containing the set of propositional symbols $\{P_1, ..., P_n\}$, its truth value for any assignment v can be computed recursively and only depends on the values $v(P_1), ..., v(P_n)$.

EXAMPLE 3.3.1

Let

$$A = ((P \supset Q) \equiv (\neg Q \supset \neg P)).$$

Let v be a truth assignment whose restriction to $\{P, Q\}$ is $v(P) = \mathbf{T}$, $v(Q) = \mathbf{F}$. According to definition 3.3.3,

$$\widehat{v}(A) = H_{\equiv}(\widehat{v}((P \supset Q)), \widehat{v}((\neg Q \supset \neg P))).$$

In turn,

$$\widehat{v}((P \supset Q)) = H_{\supset}(\widehat{v}(P), \widehat{v}(Q)) \quad \text{and}$$
$$\widehat{v}((\neg Q \supset \neg P)) = H_{\supset}(\widehat{v}(\neg Q), \widehat{v}(\neg P)).$$

Since

$$\widehat{v}(P) = v(P) \quad \text{and} \quad \widehat{v}(Q) = v(Q),$$

we have

$$\widehat{v}((P \supset Q)) = H_{\supset}(\mathbf{T}, \mathbf{F}) = \mathbf{F}.$$

We also have

$$\widehat{v}(\neg Q) = H_{\neg}(\widehat{v}(Q)) = H_{\neg}(v(Q)) \quad \text{and}$$

$$\widehat{v}(\neg P) = H_{\neg}(\widehat{v}(P)) = H_{\neg}(v(P)).$$

Hence,

$$\widehat{v}(\neg Q) = H_{\neg}(\mathbf{F}) = \mathbf{T},$$
$$\widehat{v}(\neg P) = H_{\neg}(\mathbf{T}) = \mathbf{F} \quad \text{and}$$
$$\widehat{v}((\neg Q \supset \neg P)) = H_{\supset}(\mathbf{T}, \mathbf{F}) = \mathbf{F}.$$

Finally,

$$\widehat{v}(A) = H_{\equiv}(\mathbf{F}, \mathbf{F}) = \mathbf{T}.$$

The above recursive computation can be conveniently described by a truth table as follows:

P	Q	$\neg P$	$\neg Q$	$(P \supset Q)$	$(\neg Q \supset \neg P)$	$((P \supset Q) \equiv (\neg Q \supset \neg P))$
T	**F**	**F**	**T**	**F**	**F**	**T**

If $\widehat{v}(A) = \mathbf{T}$ for a valuation v and a proposition A, we say that v *satisfies* A, and this is denoted by $v \models A$. If v does not satisfy A, we say that v *falsifies* A, and this is denoted by, *not* $v \models A$, or $v \not\models A$.

An expression such as $v \models A$ (or $v \not\models A$) is merely a notation used in the meta-language to express concisely statements about satisfaction. The reader should be well aware that such a notation is not a proposition in the object language, and should be used with care. An illustration of the danger of mixing the meta-language and the object language is given by the definition (often found in texts) of the notion of satisfaction in terms of the notation $v \models A$. Using this notation, the recursive clauses of the definition of $\widehat{v}$ can be stated informally as follows:

$$
\begin{aligned}
v &\not\models \perp, \\
v \models P_i &\text{ iff } v(P_i) = \mathbf{T}, \\
v \models \neg A &\text{ iff } v \not\models A, \\
v \models A \wedge B &\text{ iff } v \models A \textit{ and } v \models B, \\
v \models A \vee B &\text{ iff } v \models A \textit{ or } v \models B, \\
v \models A \supset B &\text{ iff } v \not\models A \textit{ or } v \models B, \\
v \models A \equiv B &\text{ iff } (v \models A \; iff \; v \models B).
\end{aligned}
$$

The above definition is not really satisfactory because it mixes the object language and the meta-language too much. In particular, the meaning of the words *not*, *or*, *and* and *iff* is ambiguous. What is worse is that the legitimacy of the recursive definition of "$\models$" is far from being clear. However, the definition of $\models$ using the recursive definition of $\widehat{v}$ is rigorously justified by theorem 2.4.1 (in the Appendix).

An important subset of $PROP$ is the set of all propositions that are true in *all* valuations. These propositions are called *tautologies*.

3.3.2 Satisfiability, Unsatisfiability, Tautologies

First, we define the concept of a tautology.

Definition 3.3.4 A proposition A is *valid* iff $\widehat{v}(A) = \mathbf{T}$ for all valuations v. This is abbreviated as $\models A$, and A is also called a *tautology*.

A proposition is *satisfiable* if there is a valuation (or truth assignment) v such that $\widehat{v}(A) = \mathbf{T}$. A proposition is *unsatisfiable* if it is not satisfied by any valuation.

Given a set of propositions Γ, we say that A is a *semantic consequence* of Γ, denoted by $\Gamma \models A$, if for all valuations v, $\widehat{v}(B) = \mathbf{T}$ for all B in Γ implies that $\widehat{v}(A) = \mathbf{T}$.

The problem of determining whether any arbitrary proposition is satisfiable is called the *satisfiability problem*. The problem of determining whether any arbitrary proposition is a tautology is called the *tautology problem*.

EXAMPLE 3.3.2

The following propositions are tautologies:

$$A \supset A,$$

$$\neg\neg A \supset A,$$
$$(P \supset Q) \equiv (\neg Q \supset \neg P).$$

The proposition
$$(P \vee Q) \wedge (\neg P \vee \neg Q)$$
is satisfied by the assignment $v(P) = \mathbf{F}$, $v(Q) = \mathbf{T}$.

The proposition
$$(\neg P \vee Q) \wedge (\neg P \vee \neg Q) \wedge P$$
is unsatisfiable. The following are valid consequences.

$$A, (A \supset B) \models B,$$
$$A, B \models (A \wedge B),$$
$$(A \supset B), \neg B \models \neg A.$$

Note that $P \supset Q$ is false if and only if both P is true and Q is false. In particular, observe that $P \supset Q$ is true when P is false.

The relationship between satisfiability and being a tautology is recorded in the following useful lemma.

Lemma 3.3.2 A proposition A is a tautology if and only if $\neg A$ is unsatisfiable.

Proof: Assume that A is a tautology. Hence, for all valuations v,
$$\widehat{v}(A) = \mathbf{T}.$$
Since
$$\widehat{v}(\neg A) = H_\neg(\widehat{v}(A)),$$
$$\widehat{v}(A) = \mathbf{T} \quad \text{if and only if} \quad \widehat{v}(\neg A) = \mathbf{F}.$$
This shows that for all valuations v,
$$\widehat{v}(\neg A) = \mathbf{F},$$
which is the definition of unsatisfiability. Conversely, if $\neg A$ is unsatisfiable, for all valuations v,
$$\widehat{v}(\neg A) = \mathbf{F}.$$
By the above reasoning, for all v,
$$\widehat{v}(A) = \mathbf{T},$$
which is the definition of being a tautology. □

The above lemma suggests two different approaches for proving that a proposition is a tautology. In the first approach, one attempts to show *directly* that A is a tautology. The method in Section 3.4 using Gentzen systems illustrates the first approach (although A is proved to be a tautology if the attempt to falsify A fails). In the second approach, one attempts to show *indirectly* that A is a tautology, by showing that $\neg A$ is unsatisfiable. The method in Chapter 4 using resolution illustrates the second approach.

As we saw in example 3.3.1, the recursive definition of the unique extension $\widehat{v}$ of a valuation v suggests an algorithm for computing the truth value $\widehat{v}(A)$ of a proposition A. The algorithm consists in computing recursively the truth tables of the parts of A. This is known as the *truth table method*.

The truth table method clearly provides an algorithm for testing whether a formula is a tautology: If A contains n propositional letters, one constructs a truth table in which the truth value of A is computed for all valuations depending on n arguments. Since there are 2^n such valuations, the size of this truth table is at least 2^n. It is also clear that there is an algorithm for deciding whether a proposition is satisfiable: Try out all possible valuations (2^n) and compute the corresponding truth table.

EXAMPLE 3.3.3

Let us compute the truth table for the proposition $A = ((P \supset Q) \equiv (\neg Q \supset \neg P))$.

P	Q	$\neg P$	$\neg Q$	$(P \supset Q)$	$(\neg Q \supset \neg P)$	$((P \supset Q) \equiv (\neg Q \supset \neg P))$
F	**F**	**T**	**T**	**T**	**T**	**T**
F	**T**	**T**	**F**	**T**	**T**	**T**
T	**F**	**F**	**T**	**F**	**F**	**T**
T	**T**	**F**	**F**	**T**	**T**	**T**

Since the last column contains only the truth value **T**, the proposition A is a tautology.

The above method for testing whether a proposition is satisfiable or a tautology is computationally expensive, in the sense that it takes an exponential number of steps. One might ask if it is possible to find a more efficient procedure. Unfortunately, the satisfiability problem happens to be what is called an NP-complete problem, which implies that there is probably no fast algorithm for deciding satisfiability. By a fast algorithm, we mean an algorithm that runs in a number of steps bounded by $p(n)$, where n is the length of the input, and p is a (fixed) polynomial. NP-complete problems and their significance will be discussed at the end of this section.

Is there a better way of testing whether a proposition A is a tautology than computing its truth table (which requires computing at least 2^n entries, where n is the number of proposition symbols occurring in A)? One possibility is to work backwards, trying to find a truth assignment which makes the proposition false. In this way, one may detect failure much earlier. This is the essence of Gentzen systems to be discussed shortly.

As we said at the beginning of Section 3.3, every proposition defines a function taking truth values as arguments, and yielding truth values as results. The truth function associated with a proposition is defined as follows.

3.3.3 Truth Functions and Functionally Complete Sets of Connectives

We now show that the logical connectives are not independent. For this, we need to define what it means for a proposition to define a truth function.

Definition 3.3.5 Let A be a formula containing exactly n distinct propositional symbols. The function $H_A : BOOL^n \to BOOL$ is defined such that, for every $(a_1, ..., a_n) \in BOOL^n$,

$$H_A(a_1, ..., a_n) = \widehat{v}(A),$$

with v any valuation such that $v(P_i) = a_i$ for every propositional symbol P_i occurring in A.

For simplicity of notation we will often name the function H_A as A. H_A is a *truth function*. In general, every function $f : BOOL^n \to BOOL$ is called an n-ary truth function.

EXAMPLE 3.3.4

The proposition

$$A = (P \wedge \neg Q) \vee (\neg P \wedge Q)$$

defines the truth function $H_\oplus$ given by the following truth table:

P	Q	$\neg P$	$\neg Q$	$(P \wedge \neg Q)$	$(\neg P \wedge Q)$	$(P \wedge \neg Q) \vee (\neg P \wedge Q))$
F	**F**	**T**	**T**	**F**	**F**	**F**
F	**T**	**T**	**F**	**F**	**T**	**T**
T	**F**	**F**	**T**	**T**	**F**	**T**
T	**T**	**F**	**F**	**F**	**F**	**F**

Note that the function $H_\oplus$ takes the value **T** if and only if its arguments have different truth values. For this reason, it is called the *exclusive OR* function.

It is natural to ask whether every truth function f can be realized by some proposition A, in the sense that $f = H_A$. This is indeed the case. We say that the boolean connectives form a *functionally complete set* of connectives.

The significance of this result is that for any truth function f of n arguments, we do not enrich the collection of assertions that we can make by adding a new symbol to the syntax, say F, and interpreting F as f. Indeed, since f is already definable in terms of $\vee$, $\wedge$, $\neg$, $\supset$, $\equiv$, every extended proposition A containing F can be converted to a proposition A' in which F does not occur, such that for every valuation v, $\widehat{v}(A) = \widehat{v}(A')$. Hence, there is no loss of generality in restricting our attention to the connectives that we have introduced. In fact, we shall prove that each of the sets $\{\vee, \neg\}$, $\{\wedge, \neg\}$, $\{\supset, \neg\}$ and $\{\supset, \perp\}$ is functionally complete.

First, we prove the following two lemmas.

Lemma 3.3.3 Let A and B be any two propositions, let $\{P_1, ..., P_n\}$ be the set of propositional symbols occurring in $(A \equiv B)$, and let H_A and H_B be the functions associated with A and B, considered as functions of the arguments in $\{P_1, ..., P_n\}$. The proposition $(A \equiv B)$ is a tautology if and only if for all valuations v, $\widehat{v}(A) = \widehat{v}(B)$, if and only if $H_A = H_B$.

Proof: For any valuation v,

$$\widehat{v}((A \equiv B)) = H_{\equiv}(\widehat{v}(A), \widehat{v}(B)).$$

Consulting the truth table for $H_{\equiv}$, we see that

$$\widehat{v}((A \equiv B)) = \mathbf{T} \quad \text{if and only if} \quad \widehat{v}(A) = \widehat{v}(B).$$

By lemma 3.3.1 and definition 3.3.5, this implies that $H_A = H_B$. $\square$

By constructing truth tables, the propositions listed in lemma 3.3.4 below can be shown to be tautologies.

Lemma 3.3.4 The following properties hold:

$$\models (A \equiv B) \equiv ((A \supset B) \wedge (B \supset A)); \tag{1}$$
$$\models (A \supset B) \equiv (\neg A \vee B); \tag{2}$$
$$\models (A \vee B) \equiv (\neg A \supset B); \tag{3}$$
$$\models (A \vee B) \equiv \neg(\neg A \wedge \neg B); \tag{4}$$
$$\models (A \wedge B) \equiv \neg(\neg A \vee \neg B); \tag{5}$$
$$\models \neg A \equiv (A \supset \perp); \tag{6}$$
$$\models \perp \equiv (A \wedge \neg A). \tag{7}$$

Proof: We prove (1), leaving the other cases as an exercise. By lemma 3.3.3, it is sufficient to verify that the truth tables for $(A \equiv B)$ and $((A \supset B) \wedge (B \supset A))$ are identical.

A	B	$A \supset B$	$B \supset A$	$(A \equiv B)$	$((A \supset B) \land (B \supset A))$
F	**F**	**T**	**T**	**T**	**T**
F	**T**	**T**	**F**	**F**	**F**
T	**F**	**F**	**T**	**F**	**F**
T	**T**	**T**	**T**	**T**	**T**

Since the columns for $(A \equiv B)$ and $((A \supset B) \land (B \supset A))$ are identical, (1) is a tautology. $\square$

We now show that $\{\lor, \land, \neg\}$ is a functionally complete set of connectives.

Theorem 3.3.1 For every n-ary truth function f, there is a proposition A only using the connectives $\land$, $\lor$ and $\neg$ such that $f = H_A$.

Proof: We proceed by induction on the arity n of f. For $n = 1$, there are four truth functions whose truth tables are:

P	1	2	3	4
F	**T**	**F**	**F**	**T**
T	**T**	**F**	**T**	**F**

Clearly, the propositions $P \lor \neg P$, $P \land \neg P$, P and $\neg P$ do the job. Let f be of arity $n + 1$ and assume the induction hypothesis for n. Let

$$f_1(x_1, ..., x_n) = f(x_1, ..., x_n, \mathbf{T}) \quad \text{and}$$
$$f_2(x_1, ..., x_n) = f(x_1, ..., x_n, \mathbf{F}).$$

Both f_1 and f_2 are n-ary. By the induction hypothesis, there are propositions B and C such that

$$f_1 = H_B \quad \text{and} \quad f_2 = H_C.$$

But then, letting A be the formula

$$(P_{n+1} \land B) \lor (\neg P_{n+1} \land C),$$

where P_{n+1} occurs neither in B nor C, it is easy to see that $f = H_A$. $\square$

Using lemma 3.3.4, it follows that $\{\lor, \neg\}$, $\{\land, \neg\}$, $\{\supset, \neg\}$ and $\{\supset, \perp\}$ are functionally complete. Indeed, using induction on propositions, $\land$ can be expressed in terms of $\lor$ and $\neg$ by (5), $\supset$ can be expressed in terms of $\neg$ and $\lor$ by (2), $\equiv$ can be expressed in terms of $\supset$ and $\land$ by (1), and $\perp$ can be expressed in terms of $\land$ and $\neg$ by (7). Hence, $\{\lor, \neg\}$ is functionally complete. Since $\lor$ can be expressed in terms of $\land$ and $\neg$ by (4), the set $\{\land, \neg\}$ is functionally complete, since $\{\lor, \neg\}$ is. Since $\lor$ can be expressed in terms of $\supset$ and $\neg$ by (3), the set $\{\supset, \neg\}$ is functionally complete, since $\{\lor, \neg\}$ is. Finally, since $\neg$

can be expressed in terms of $\supset$ and $\perp$ by (6), the set $\{\supset, \perp\}$ is functionally complete since $\{\supset, \neg\}$ is.

In view of the above theorem, we may without loss of generality restrict our attention to propositions expressed in terms of the connectives in some functionally complete set of our choice. The choice of a suitable functionally complete set of connectives is essentially a matter of convenience and taste. The advantage of using a small set of connectives is that fewer cases have to be considered in proving properties of propositions. The disadvantage is that the meaning of a proposition may not be as clear for a proposition written in terms of a smaller complete set, as it is for the same proposition expressed in terms of the full set of connectives used in definition 3.2.1. Furthermore, depending on the set of connectives chosen, the representations can have very different lengths. For example, using the set $\{\supset, \perp\}$, the proposition $(A \wedge B)$ takes the form

$$(A \supset (B \supset \perp)) \supset \perp .$$

I doubt that many readers think that the second representation is more perspicuous than the first!

In this book, we will adopt the following compromise between mathematical conciseness and intuitive clarity. The set $\{\wedge, \vee, \neg, \supset\}$ will be used. Then, $(A \equiv B)$ will be considered as an abbreviation for $((A \supset B) \wedge (B \supset A))$, and $\perp$ as an abbrevation for $(P \wedge \neg P)$.

We close this section with some results showing that the set of propositions has a remarkable structure called a boolean algebra. (See Subsection 2.4.1 in the Appendix for the definition of an algebra.)

3.3.4 Logical Equivalence and Boolean Algebras

First, we show that lemma 3.3.3 implies that a certain relation on $PROP$ is an equivalence relation.

Definition 3.3.6 The relation $\simeq$ on $PROP$ is defined so that for any two propositions A and B, $A \simeq B$ if and only if $(A \equiv B)$ is a tautology. We say that A and B are *logically equivalent*, or for short, equivalent.

From lemma 3.3.3, $A \simeq B$ if and only if $H_A = H_B$. This implies that the relation $\simeq$ is reflexive, symmetric, and transitive, and therefore it is an equivalence relation. The following additional properties show that it is a congruence in the sense of Subsection 2.4.6 (in the Appendix).

Lemma 3.3.5 For all propositions A,A',B,B', the following properties hold:

If $A \simeq A'$ and $B \simeq B'$, then for $* \in \{\wedge, \vee, \supset, \equiv\}$,

$$(A * B) \simeq (A' * B') \quad \text{and}$$
$$\neg A \simeq \neg A'.$$

Proof: By definition 3.3.6,

$$(A * B) \simeq (A' * B') \quad \text{if and only if} \quad \models (A * B) \equiv (A' * B').$$

By lemma 3.3.3, it is sufficient to show that for all valuations v,

$$\widehat{v}(A * B) = \widehat{v}(A' * B').$$

Since

$$A \simeq A' \quad \text{and} \quad B \simeq B' \quad \text{implies that}$$
$$\widehat{v}(A) = \widehat{v}(A') \quad \text{and} \quad \widehat{v}(B) = \widehat{v}(B'),$$

we have

$$\widehat{v}(A * B) = H_*(\widehat{v}(A), \widehat{v}(B)) = H_*(\widehat{v}(A'), \widehat{v}(B')) = \widehat{v}(A' * B').$$

Similarly,

$$\widehat{v}(\neg A) = H_\neg(\widehat{v}(A)) = H_\neg(\widehat{v}(A')) = \widehat{v}(\neg A').$$

□

In the rest of this section, it is assumed that the constant symbol $\top$ is added to the alphabet of definition 3.2.1, yielding the set of propositions $PROP'$, and that $\top$ is interpreted as $\mathbf{T}$. The proof of the following properties is left as an exercise.

Lemma 3.3.6 The following properties hold for all propositions in $PROP'$.

Associativity rules:

$$((A \lor B) \lor C) \simeq (A \lor (B \lor C)) \quad ((A \land B) \land C) \simeq (A \land (B \land C))$$

Commutativity rules:

$$(A \lor B) \simeq (B \lor A) \quad (A \land B) \simeq (B \land A)$$

Distributivity rules:

$$(A \lor (B \land C)) \simeq ((A \lor B) \land (A \lor C))$$
$$(A \land (B \lor C)) \simeq ((A \land B) \lor (A \land C))$$

De Morgan's rules:

$$\neg(A \lor B) \simeq (\neg A \land \neg B) \quad \neg(A \land B) \simeq (\neg A \lor \neg B)$$

Idempotency rules:

$$(A \lor A) \simeq A \quad (A \land A) \simeq A$$

Double negation rule:

$$\neg\neg A \simeq A$$

Absorption rules:

$$(A \lor (A \land B)) \simeq A \quad (A \land (A \lor B)) \simeq A$$

Laws of zero and one:

$$(A \lor \perp) \simeq A \quad (A \land \perp) \simeq \perp$$
$$(A \lor \top) \simeq \top \quad (A \land \top) \simeq A$$
$$(A \lor \neg A) \simeq \top \; (A \land \neg A) \simeq \perp$$

Let us denote the equivalence class of a proposition A modulo $\simeq$ as $[A]$, and the set of all such equivalence classes as $\mathbf{B}_{PROP}$. We define the operations $+$, $*$ and $\neg$ on $\mathbf{B}_{PROP}$ as follows:

$$\begin{aligned}[A] + [B] &= [A \vee B],\\ [A] * [B] &= [A \wedge B],\\ \neg[A] &= [\neg A].\end{aligned}$$

Also, let $0 = [\perp]$ and $1 = [\top]$.

By lemma 3.3.5, the above functions (and constants) are independent of the choice of representatives in the equivalence classes. But then, the properties of lemma 3.3.6 are identities valid on the set $\mathbf{B}_{PROP}$ of equivalence classes modulo $\simeq$. The structure $\mathbf{B}_{PROP}$ is an algebra in the sense of Subsection 2.4.1 (in the Appendix). Because it satisfies the identities of lemma 3.3.6, it is a very rich structure called a *boolean algebra*. $\mathbf{B}_{PROP}$ is called the *Lindenbaum algebra* of $PROP$. In this book, we will only make simple uses of the fact that $\mathbf{B}_{PROP}$ is a boolean algebra (the properties in lemma 3.3.6, associativity, commutativity, distributivity, and idempotence in particular) but the reader should be aware that there are important and interesting consequences of this fact. However, these considerations are beyond the scope of this text. We refer the reader to Halmos, 1974, or Birkhoff, 1973 for a comprehensive study of these algebras.

* 3.3.5 NP-Complete Problems

It has been remarked earlier that both the satisfiability problem (SAT) and the tautology problem ($TAUT$) are computationally hard problems, in the sense that known algorithms to solve them require an exponential number of steps in the length of the input. Even though modern computers are capable of performing very complex computations much faster than humans can, there are problems whose computational complexity is such that it would take too much time or memory space to solve them with a computer. Such problems are called *intractable*. It should be noted that this does not mean that we do not have algorithms to solve such problems. This means that all known algorithms solving these problems in theory either require too much time or too much memory space to solve them in practice, except perhaps in rather trivial cases. An algorithm is considered intractable if either it requires an exponential number of steps, or an exponential amount of space, in the length of the input. This is because exponential functions grow very fast. For example, $2^{10} = 1024$, but 2^{1000} is equal to $10^{1000 log_{10} 2}$, which has over 300 digits! A problem that can be solved in polynomial time and polynomial space is considered to be tractable.

It is not known whether SAT or $TAUT$ are tractable, and in fact, it is conjectured that they are not. But SAT and $TAUT$ play a special role for another reason. There is a class of problems (NP) which contains problems for which no polynomial-time algorithms are known, but for which polynomial-time solutions exist, if we are allowed to make guesses, and if we are not charged for checking wrong guesses, but only for successful guesses leading to an answer. SAT is such a problem. Indeed, given a proposition A, if one is allowed to guess valuations, it is not difficult to design a polynomial-time algorithm to check that a valuation v satisfies A. The satisfiability problem can be solved nondeterministically by guessing valuations and checking that they satisfy the given proposition. Since we are not "charged" for checking wrong guesses, such a procedure works in polynomial-time.

A more accurate way of describing such algorithms is to say that free backtracking is allowed. If the algorithm reaches a point where several choices are possible, any choice can be taken, but if the path chosen leads to a dead end, the algorithm can jump back (backtrack) to the latest choice point, with no cost of computation time (and space) consumed on a wrong path involved. Technically speaking, such algorithms are called *nondeterministic*. A nondeterministic algorithm can be simulated by a deterministic algorithm, but the deterministic algorithm needs to keep track of the nondeterministic choices explicitly (using a stack), and to use a backtracking technique to handle unsuccessful computations. Unfortunately, all known backtracking techniques yield exponential-time algorithms.

In order to discuss complexity issues rigorously, it is necessary to define a model of computation. Such a model is the *Turing machine* (invented by the mathematician Turing, circa 1935). We will not present here the theory of Turing Machines and complexity classes, but refer the interested reader to Lewis and Papadimitriou, 1981, or Davis and Weyuker, 1983. We will instead conclude with an informal discussion of the classes P and NP.

In dealing with algorithms for solving classes of problems, it is convenient to assume that problems are encoded as sets of strings over a finite alphabet Σ. Then, an algorithm for solving a problem A is an algorithm for deciding whether for any string $u \in \Sigma^*$, u is a member of A. For example, the satisfiability problem is encoded as the set of strings representing satisfiable propositions, and the tautology problem as the set of strings representing tautologies.

A Turing machine is an abstract computing device used to accept sets of strings. Roughly speaking, a Turing machine M consists of a finite set of states and of a finite set of instructions. The set of states is partitioned into two subsets of *accepting* and *rejecting* states.

To explain how a Turing machine operates, we define the notion of an instantaneous description (ID) and of a computation. An instantaneous description is a sort of snapshot of the configuration of the machine during a computation that, among other things, contains a state component. A Turing machine operates in discrete steps. Every time an instruction is executed, the ID describing the current configuration is updated. The intuitive idea is that executing an instruction I when the machine is in a configuration described by an instantaneous description C_1 yields a new configuration described by C_2. A computation is a finite or infinite sequence of instantaneous descriptions $C_0, ..., C_n$ (or $C_0, ..., C_n, C_{n+1}, ...$, if it is infinite), where C_0 is an initial instantaneous description containing the input, and each C_{i+1} is obtained from C_i by application of some instruction of M. A finite computation is called a halting computation, and the last instantaneous description C_n is called a final instantaneous description.

A Turing machine is deterministic, if for every instantaneous description C_1, at most one instantaneous description C_2 follows from C_1 by execution of some instruction of M. It is nondeterministic if for any ID C_1, there may be several successors C_2 obtained by executing (different) instructions of M.

Given a halting computation $C_0, ..., C_n$, if the final ID C_n contains an accepting state, we say that the computation is an *accepting computation*, otherwise it is a *rejecting computation*.

A set A (over Σ) is accepted deterministically in polynomial time if there is a deterministic Turing machine M and a polynomial p such that, for every input u,

(i) $u \in A$ iff the computation $C_0, ..., C_n$ on input u is an accepting computation such that $n \leq p(|u|)$ and,

(ii) $u \notin A$ iff the computation $C_0, ..., C_n$ on input u is a rejecting computation such that $n \leq p(|u|)$.

A set A (over Σ) is accepted nondeterministically in polynomial time if there is a nondeterministic Turing machine M and a polynomial p, such that, for every input u, there is *some* accepting computation $C_0, ..., C_n$ such that $n \leq p(|u|)$.

It should be noted that in the nondeterministic case, a string u is rejected by a Turing machine M (that is, $u \notin A$) iff every computation of M is either a rejecting computation $C_0, ..., C_n$ such that $n \leq p(|u|)$, or a computation that takes more than $p(|u|)$ steps on input u.

The class of sets accepted deterministically in polynomial time is denoted by P, and the class of sets accepted nondeterministically in polynomial time is denoted by NP. It is obvious from the definitions that P is a subset of NP. However, whether $P = NP$ is unknown, and in fact, is a famous open problem.

The importance of the class P resides in the widely accepted (although somewhat controversial) opinion that P consists of the problems that can be realistically solved by computers. The importance of NP lies in the fact that many problems for which efficient algorithms would be highly desirable, but are yet unknown, belong to NP. The traveling salesman problem and the integer programming problem are two such problems among many others.

For an extensive list of problems in NP, the reader should consult Garey and Johnson, 1979.

The importance of the satisfiability problem SAT is that it is *NP-complete*. This implies the remarkable fact that if SAT is in P, then $P = NP$. In other words, the existence of a polynomial-time algorithm for SAT implies that all problems in NP have polynomial-time algorithms, which includes many interesting problems apparently untractable at the moment.

In order to explain the notion of NP-completeness, we need the concept of polynomial-time reducibility. Deterministic Turing machines can also be used to compute functions. Given a function $f : \Sigma^* \to \Sigma^*$, a deterministic Turing machine M computes f if for every input u, there is a halting computation $C_0, ..., C_n$ such that C_0 contains u as input and C_n contains $f(u)$ as output. The machine M computes f in polynomial time iff there is a polynomial p such that, for every input u, $n \leq p(|u|)$, where n is the number of steps in the computation on input u. Then, we say that a set A is polynomially reducible to a set B if there is a function $f : \Sigma^* \to \Sigma^*$ computable in polynomial time such that, for every input u,

$$u \in A \quad \text{if and only if} \quad f(u) \in B.$$

A set B is *NP-hard* if every set A in NP is reducible to B. A set B is *NP-complete* if it is in NP, and it is *NP-hard*.

The significance of NP-complete problems lies in the fact that if one finds a polynomial time algorithm for any NP-complete problem B, then $P = NP$. Indeed, given any problem $A \in NP$, assuming that the deterministic Turing machine M solves B in polynomial time, we could construct a deterministic Turing machine M' solving A as follows. Let M_f be the deterministic Turing machine computing the reduction function f. Then, to decide whether any arbitrary input u is in A, run M_f on input u, producing $f(u)$, and then run M on input $f(u)$. Since $u \in A$ if and only if $f(u) \in B$, the above procedure solves A. Furthermore, it is easily shown that a deterministic Turing machine M' simulating the composition of M_f and M can be constructed, and that it runs in polynomial time (because the functional composition of polynomials is a polynomial).

The importance of SAT lies in the fact that it was shown by S. A. Cook (Cook, 1971) that SAT is an NP-complete problem. In contrast, whether $TAUT$ is in NP is an open problem. But $TAUT$ is interesting for two other reasons. First, it can be shown that if $TAUT$ is in P, then $P = NP$. This is unlikely since we do not even know whether $TAUT$ is in NP. The second reason is related to the closure of NP under complementation. NP is said to be *closed under complementation* iff for every set A in NP, its complement $\Sigma^* - A$ is also in NP.

The class P is closed under complementation, but this is an open problem for the class NP. Given a deterministic Turing machine M, in order to accept the complement of the set A accepted by M, one simply has to create the machine $\overline{M}$ obtained by swapping the accepting and the rejecting states of M. Since for every input u, the computation $C_0, ..., C_n$ of M on input u halts in $n \leq p(|u|)$ steps, the modified machine $\overline{M}$ accepts $\Sigma^* - A$ is polynomial time. However, if M is nondeterministic, M may reject some input u because *all* computations on input u exceed the polynomial time bound $p(|u|)$. Thus, for this input u, there is no computation of the modified machine $\overline{M}$ which accepts u within $p(|u|)$ steps. The trouble is not that $\overline{M}$ cannot tell that u is rejected by M, but that $\overline{M}$ cannot report this fact in fewer than $p(|u|)$ steps. This shows that in the nondeterministic case, a different construction is required. Until now, no such construction has been discovered, and is rather unlikely that it will. Indeed, it can be shown that $TAUT$ is in NP if and only if NP is closed under complementation. Furthermore, since P is closed under complementation if NP is not closed under complementation, then $NP \neq P$.

Hence, one approach for showing that $NP \neq P$ would be to show that $TAUT$ is not in NP. This explains why a lot of effort has been spent on the complexity of the tautology problem.

To summarize, the satisfiability problem SAT and the tautology problem $TAUT$ are important because of the following facts:

$$SAT \in P \quad \text{if and only if} \quad P = NP;$$
$$TAUT \in NP \quad \text{if and only if} \quad NP \text{ is closed under complementation;}$$
$$\text{If } TAUT \in P, \text{ then } P = NP;$$
$$\text{If } TAUT \notin NP, \text{ then } NP \neq P.$$

Since the two questions $P = NP$ and the closure of NP under complementation appear to be very hard to solve, and it is usually believed that their answer is negative, this gives some insight to the difficulty of finding efficient algorithms for SAT and $TAUT$. Also, the tautology problem appears to be harder than the satisfiability problem. For more details on these questions, we refer the reader to the article by S. A. Cook and R. A. Reckhow (Cook and Reckhow, 1971).

PROBLEMS

3.3.1. In this problem, it is assumed that the language of propositional logic is extended by adding the constant symbol $\top$, which is interpreted as **T**. Prove that the following propositions are tautologies by constructing truth tables.

Associativity rules:

$$((A \lor B) \lor C) \equiv (A \lor (B \lor C)) \quad ((A \land B) \land C) \equiv (A \land (B \land C))$$

Commutativity rules:

$$(A \lor B) \equiv (B \lor A) \quad (A \land B) \equiv (B \land A)$$

Distributivity rules:

$$(A \lor (B \land C)) \equiv ((A \lor B) \land (A \lor C))$$
$$(A \land (B \lor C)) \equiv ((A \land B) \lor (A \land C))$$

De Morgan's rules:

$$\neg(A \lor B) \equiv (\neg A \land \neg B) \quad \neg(A \land B) \equiv (\neg A \lor \neg B)$$

Idempotency rules:

$$(A \lor A) \equiv A \quad (A \land A) \equiv A$$

Double negation rule:

$$\neg\neg A \equiv A$$

Absorption rules:

$$(A \lor (A \land B)) \equiv A \quad (A \land (A \lor B)) \equiv A$$

Laws of zero and one:

$$(A \lor \perp) \equiv A \quad (A \land \perp) \equiv \perp$$
$$(A \lor \top) \equiv \top \quad (A \land \top) \equiv A$$
$$(A \lor \neg A) \equiv \top \ (A \land \neg A) \equiv \perp$$

3.3.2. Show that the following propositions are tautologies.

$$A \supset (B \supset A)$$
$$(A \supset B) \supset ((A \supset (B \supset C)) \supset (A \supset C))$$
$$A \supset (B \supset (A \land B))$$
$$A \supset (A \lor B) \qquad B \supset (A \lor B)$$
$$(A \supset B) \supset ((A \supset \neg B) \supset \neg A)$$
$$(A \land B) \supset A \qquad (A \land B) \supset B$$
$$(A \supset C) \supset ((B \supset C) \supset ((A \lor B) \supset C))$$
$$\neg\neg A \supset A$$

3.3.3. Show that the following propositions are tautologies.

$$(A \supset B) \supset ((B \supset A) \supset (A \equiv B))$$
$$(A \equiv B) \supset (A \supset B)$$
$$(A \equiv B) \supset (B \supset A)$$

3.3.4. Show that the following propositions are not tautologies. Are they satisfiable? If so, give a satisfying valuation.

$$(A \supset C) \supset ((B \supset D) \supset ((A \lor B) \supset C))$$
$$(A \supset B) \supset ((B \supset \neg C) \supset \neg A)$$

3.3.5. Prove that the propositions of lemma 3.3.4 are tautologies.

∗ **3.3.6.** Given a function f of m arguments and m functions $g_1, ..., g_m$ each of n arguments, the composition of f and $g_1, ..., g_m$ is the function h of n arguments such that for all $x_1, ..., x_n$,

$$h(x_1, ..., x_n) = f(g_1(x_1, ..., x_n), ..., g_m(x_1, ..., x_n)).$$

For every integer $n \geq 1$, we let P_i^n, $(1 \leq i \leq n)$, denote the projection function such that, for all $x_1, .., x_n$,

$$P_i^n(x_1, ..., x_n) = x_i.$$

Then, given any k truth functions $H_1, ..., H_k$, let $\mathbf{TF}_n$ be the inductive closure of the set of functions $\{H_1, ..., H_k, P_1^n, ..., P_n^n\}$ under composition. We say that a truth function H of n arguments is *definable* from $H_1, ..., H_k$ if H belongs to $\mathbf{TF}_n$.

Let $H_{d,n}$ be the n-ary truth function such that

$$H_{d,n}(x_1, ..., x_n) = \mathbf{F} \quad \text{if and only if} \quad x_1 = ... = x_n = \mathbf{F},$$

and $H_{c,n}$ the n-ary truth function such that

$$H_{c,n}(x_1, ..., x_n) = \mathbf{T} \quad \text{if and only if} \quad x_1 = ... = x_n = \mathbf{T}.$$

(i) Prove that every n-ary truth function is definable in terms of $H_\neg$ and some of the functions $H_{d,n}$, $H_{c,n}$.

(ii) Prove that $H_\neg$ is not definable in terms of $H_\vee$, $H_\wedge$, $H_\supset$, and $H_\equiv$.

∗ **3.3.7.** Let H_{nor} be the binary truth function such that

$$H_{nor}(x, y) = \mathbf{T} \quad \text{if and only if} \quad x = y = \mathbf{F}.$$

Show that $H_{nor} = H_A$, where A is the proposition $(\neg P \wedge \neg Q)$. Show that $\{H_{nor}\}$ is functionally complete.

∗ **3.3.8.** Let H_{nand} be the binary truth function such that $H_{nand}(x, y) = \mathbf{F}$ if and only if $x = y = \mathbf{T}$. Show that $H_{nand} = H_B$, where B is the proposition $(\neg P \vee \neg Q)$. Show that $\{H_{nand}\}$ is functionally complete.

∗ **3.3.9.** An n-ary truth function H is *singulary* if there is a unary truth function H' and some i, $1 \leq i \leq n$, such that for all $x_1, ..., x_n$, $H(x_1, ..., x_n) = H'(x_i)$.

(i) Prove that if H is singulary, then every n-ary function definable in terms of H is also singulary. (See problem 3.3.6.)

(ii) Prove that if H is a binary truth function and $\{H\}$ is functionally complete, then either $H = H_{nor}$ or $H = H_{nand}$.

Hint: Show that $H(\mathbf{T},\mathbf{T}) = \mathbf{F}$ and $H(\mathbf{F},\mathbf{F}) = \mathbf{T}$, that only four binary truth functions have that property, and use (i).

3.3.10. A *substitution* is a function $s : \mathbf{PS} \to PROP$. Since $PROP$ is freely generated by $\mathbf{PS}$ and $\perp$, every substitution s extends to a unique function $\widehat{s} : PROP \to PROP$ defined by recursion. Let A be any proposition containing the propositional symbols $\{P_1, ..., P_n\}$, and s_1 and s_2 be any two substitutions such that for every $P_i \in \{P_1, ..., P_n\}$, the propositions $s_1(P_i)$ and $s_2(P_i)$ are equivalent (that is $s_1(P_i) \equiv s_2(P_i)$ is a tautology).

Prove that the propositions $\widehat{s}_1(A)$ and $\widehat{s}_2(A)$ are equivalent.

3.3.11. Show that for every set Γ of propositions,

(i) $\Gamma, A \models B$ if and only if $\Gamma \models (A \supset B)$;
(ii) If $\Gamma, A \models B$ and $\Gamma, A \models \neg B$, then $\Gamma \models \neg A$;
(iii) If $\Gamma, A \models C$ and $\Gamma, B \models C$, then $\Gamma, (A \vee B) \models C$.

3.3.12. Assume that we consider propositions expressed only in terms of the set of connectives $\{\vee, \wedge, \neg\}$. The *dual* of a proposition A, denoted by A^*, is defined recursively as follows:

$$\begin{aligned} P^* &= P, \quad \text{for every propositional symbol } P; \\ (A \vee B)^* &= (A^* \wedge B^*); \\ (A \wedge B)^* &= (A^* \vee B^*); \\ (\neg A)^* &= \neg A^*. \end{aligned}$$

(a) Prove that, for any two propositions A and B,

$$\models A \equiv B \quad \text{if and only if} \quad \models A^* \equiv B^*.$$

(b) If we change the definition of A^* so that for every propositional letter P, $P^* = \neg P$, prove that A^* and $\neg A$ are logically equivalent (that is, $(A^* \equiv \neg A)$ is a tautology).

3.3.13. A *literal* is either a propositional symbol P, or the negation $\neg P$ of a propositional symbol. A proposition A is in *disjunctive normal form* (DNF) if it is of the form $C_1 \vee ... \vee C_n$, where each C_i is a conjunction of literals.

(i) Show that the satisfiability problem for propositions in DNF can be decided in linear time. What does this say about the complexity of any algorithm for converting a proposition to disjunctive normal form?

(ii) Using the proof of theorem 3.3.1 and some of the identities of lemma 3.3.6, prove that every proposition A containing n propositional symbols is equivalent to a proposition A' in disjunctive normal form, such that each disjunct C_i contains exactly n literals.

$*$ **3.3.14.** Let $H_\oplus$ be the truth function defined by the proposition

$$(P \wedge \neg Q) \vee (\neg P \wedge Q).$$

(i) Prove that $\oplus$ (*exclusive OR*) is commutative and associative.

(ii) In this question, assume that the constant for false is denoted by 0 and that the constant for true is denoted by 1. Prove that the following are tautologies.

$$\begin{gathered}
A \vee B \equiv A \wedge B \oplus A \oplus B \\
\neg A \equiv A \oplus 1 \\
A \oplus 0 \equiv A \\
A \oplus A \equiv 0 \\
A \wedge 1 \equiv A \\
A \wedge A \equiv A \\
A \wedge (B \oplus C) \equiv A \wedge B \oplus A \wedge C \\
A \wedge 0 \equiv 0
\end{gathered}$$

(iii) Prove that $\{\oplus, \wedge, 1\}$ is functionally complete.

$*$ **3.3.15.** Using problems 3.3.13 and 3.3.14, prove that every proposition A is equivalent to a proposition A' which is either of the form 0, 1, or $C_1 \oplus ... \oplus C_n$, where each C_i is either 1 or a conjunction of positive literals. Furthermore, show that A' can be chosen so that the C_i are distinct, and that the positive literals in each C_i are all distinct (such a proposition A' is called a *reduced exclusive-OR normal form*).

$*$ **3.3.16.** (i) Prove that if A' and A'' are reduced exclusive-OR normal forms of a same proposition A, then they are equal up to commutativity, that is:

$$\begin{gathered}
\text{Either} \quad A' = A'' = 0, \text{ or } A' = A'' = 1, \text{ or} \\
A' = C_1' \wedge ... \wedge C_n', \quad A'' = C_1'' \wedge ... \wedge C_n'',
\end{gathered}$$

where each C_i' is a permutation of some C_j'' (and conversely).

Hint: There are 2^{2^n} truth functions of n arguments.

(ii) Prove that a proposition is a tautology if and only if its reduced exclusive-OR normal form is 1. What does this say about the complexity of any algorithm for converting a proposition to reduced exclusive-OR normal form?

$*$ **3.3.17.** A set Γ of propositions is *independent* if, for every $A \in \Gamma$,

$$\Gamma - \{A\} \not\models A.$$

(a) Prove that every finite set Γ has a finite independent subset Δ such that, for every $A \in \Gamma$, $\Delta \models A$.

(b) Let Γ be ordered as the sequence $< A_1, A_2, ... >$. Find a sequence $\Gamma' =< B_1, B_2, ... >$ equivalent to Γ (that is, for every $i \geq 1$, $\Gamma \models B_i$ and $\Gamma' \models A_i$), such that, for every $i \geq 1$, $\models (B_{i+1} \supset B_i)$, but $\not\models (B_i \supset B_{i+1})$. Note that Γ' may be finite.

(c) Consider a countable sequence Γ' as in (b). Define $C_1 = B_1$, and for every $i \geq 1$, $C_{n+1} = (B_n \supset B_{n+1})$. Prove that $\Delta =< C_1, C_2, ... >$ is equivalent to Γ' and independent.

(d) Prove that every countable set Γ is equivalent to an independent set.

(e) Show that Δ need not be a subset of Γ.

Hint: Consider

$$\{P_0, P_0 \wedge P_1, P_0 \wedge P_1 \wedge P_1, ...\}.$$

$*$ **3.3.18.** See problem 3.3.13 for the definition of a *literal*. A proposition A is a *basic Horn formula* iff it is a disjunction of literals, with at most one positive literal (literal of the form P). A proposition is a *Horn formula* iff it is a conjunction of basic Horn formulae.

(a) Show that every Horn formula A is equivalent to a conjunction of distinct formulae of the form,

$$P_i, \quad \text{or}$$
$$\neg P_1 \vee .... \vee \neg P_n, (n \geq 1), \quad \text{or}$$
$$\neg P_1 \vee .... \vee \neg P_n \vee P_{n+1}, (n \geq 1),$$

where all the P_i are distinct. We say that A is *reduced*.

(b) Let A be a reduced Horn formula $A = C_1 \wedge C_2 \wedge ... \wedge C_n$, where the C_i are distinct and each C_i is reduced as in (a). Since $\vee$ is commutative and associative (see problem 3.3.1), we can view each conjunct C_i as a set.

(i) Show that if no conjunct C_i is a positive literal, or every conjunct containing a negative literal also contains a positive literal, then A is satisfiable.

(ii) Assume that A contains some conjunct C_i having a single positive literal P, and some conjunct C_j distinct from C_i, such that C_j contains $\neg P$. Let $D_{i,j}$ be obtained by deleting $\neg P$ from C_j. Let A' be the conjunction obtained from A by replacing C_j by the conjunct $D_{i,j}$, provided that $D_{i,j}$ is not empty.

Show that A is satisfiable if and only if A' is satisfiable and $D_{i,j}$ is not empty.

(iii) Using the above, prove that the satisfiability of a Horn formula A can be decided in time polynomial in the length of A.

Note: Linear-time algorithms are given in Dowling and Gallier, 1984.

3.4 Proof Theory of Propositional Logic: The Gentzen System G'

In this section, we present a proof system for propositional logic and prove some of its main properties: soundness and completeness.

3.4.1 Basic Idea: Searching for a Counter Example

As we have suggested in Section 3.3, another perhaps more effective way of testing whether a proposition A is a tautology is to search for a valuation that falsifies A. In this section, we elaborate on this idea. As we progress according to this plan, we will be dealing with a tree whose nodes are labeled with pairs of finite lists of propositions. In our attempt to falsify A, the tree is constructed in such a way that we are trying to find a valuation that makes every proposition occurring in the first component of a pair labeling a node true, and all propositions occurring in the second component of that pair false. Hence, we are naturally led to deal with pairs of finite sequences of propositions called *sequents*. The idea of using sequents originates with Gentzen, although Gentzen's motivations were quite different. A proof system using sequents is very natural because the rules reflect very clearly the semantics of the connectives. The idea of searching for a valuation falsifying the given proposition is simple, and the tree-building algorithm implementing this search is also simple. Let us first illustrate the falsification procedure by means of an example.

EXAMPLE 3.4.1

Let

$$A = (P \supset Q) \supset (\neg Q \supset \neg P).$$

Initially, we start with a one-node tree labeled with the pair

$$(<>, < (P \supset Q) \supset (\neg Q \supset \neg P) >)$$

whose first component is the empty sequence and whose second component is the sequence containing the proposition A that we are attempting to falsify. In order to make A false, we must make $P \supset Q$ true and $\neg Q \supset \neg P$ false. Hence, we build the following tree:

$$\frac{(< P \supset Q >, < \neg Q \supset \neg P >)}{(<>, < (P \supset Q) \supset (\neg Q \supset \neg P) >)}$$

Now, in order to make $P \supset Q$ true, we must *either* make P false *or* Q true. The tree must therefore split as shown:

$$\dfrac{\dfrac{(<>, < P, \neg Q \supset \neg P >) \qquad\qquad (< Q >, < \neg Q \supset \neg P >)}{(< P \supset Q >, < \neg Q \supset \neg P >)}}{(<>, < (P \supset Q) \supset (\neg Q \supset \neg P) >)}$$

We continue the same procedure with each leaf. Let us consider the leftmost leaf first. In order to make $\neg Q \supset \neg P$ false, we must make $\neg Q$ true and $\neg P$ false. We obtain the tree:

$$\dfrac{\dfrac{\dfrac{(< \neg Q >, < P, \neg P >)}{(<>, < P, \neg Q \supset \neg P >)} \qquad\qquad (< Q >, < \neg Q \supset \neg P >)}{(< P \supset Q >, < \neg Q \supset \neg P >)}}{(<>, < (P \supset Q) \supset (\neg Q \supset \neg P) >)}$$

But now, in order to falsify the leftmost leaf, we must make both P and $\neg P$ false and $\neg Q$ true. This is impossible. We say that this leaf of the tree is *closed*. We still have to continue the procedure with the rightmost leaf, since there may be a way of obtaining a falsifying valuation this way. To make $\neg Q \supset \neg P$ false, we must make $\neg Q$ true and $\neg P$ false, obtaining the tree:

$$\dfrac{\dfrac{\dfrac{(< \neg Q >, < P, \neg P >)}{(<>, < P, \neg Q \supset \neg P >)} \qquad\qquad \dfrac{(< Q, \neg Q >, < \neg P >)}{(< Q >, < \neg Q \supset \neg P >)}}{(< P \supset Q >, < \neg Q \supset \neg P >)}}{(<>, < (P \supset Q) \supset (\neg Q \supset \neg P) >)}$$

This time, we must try to make $\neg P$ false and both Q and $\neg Q$ false, which is impossible. Hence, this branch of the tree is also closed, and our attempt to falsify A has failed. However, this failure to falsify A is really a success, since, as we shall prove shortly, this demonstrates that A is valid!

Trees as above are called *deduction trees*. In order to describe precisely the algorithm we have used in our attempt to falsify the proposition A, we need to state clearly the rules that we have used in constructing the tree.

3.4.2 Sequents and the Gentzen System G'

First, we define the notion of a sequent.

Definition 3.4.1 A *sequent* is a pair (Γ, Δ) of finite (possibly empty) sequences $\Gamma =< A_1, ..., A_m >$, $\Delta =< B_1, ..., B_n >$ of propositions.

Instead of using the notation (Γ, Δ), a sequent is usually denoted as $\Gamma \rightarrow \Delta$. For simplicity, a sequence $< A_1, ..., A_m >$ is denoted as $A_1, ..., A_m$. If Γ is the empty sequence, the corresponding sequent is denoted as $\rightarrow \Delta$; if Δ is empty, the sequent is denoted as $\Gamma \rightarrow$. and if both Γ and Δ are empty, we have the special sequent $\rightarrow$ (the *inconsistent sequent*). Γ is called the *antecedent* and Δ the *succedent*.

The intuitive meaning of a sequent is that a valuation v makes a sequent $A_1, ..., A_m \rightarrow B_1, ..., B_n$ true iff

$$v \models (A_1 \land ... \land A_m) \supset (B_1 \lor ... \lor B_n).$$

Equivalently, v makes the sequent false if v makes $A_1, ..., A_m$ all true and $B_1, ..., B_n$ all false.

It should be noted that the semantics of sequents suggests that instead of using sequences, we could have used sets. We could indeed define sequents as pairs (Γ, Δ) of finite *sets* of propositions, and all the results in this section would hold. The results of Section 3.5 would also hold, but in order to present the generalization of the tree construction procedure, we would have to order the sets present in the sequents anyway. Rather than switching back and forth between sets and sequences, we think that it is preferable to stick to a single formalism. Using sets instead of sequences can be viewed as an optimization.

The rules operating on sequents fall naturally into two categories: those operating on a proposition occurring in the antecedent, and those on a proposition occurring in the succedent. Both kinds of rules break the proposition on which the rule operates into subpropositions that may also be moved from the antecedent to the succedent, or vice versa. Also, the application of a rule may cause a sequent to be split into two sequents. This causes branching in the trees. Before stating the rules, let us mention that it is traditional in logic to represent trees with their root at the bottom instead of the root at the top as it is customary in computer science. The main reason is that a tree obtained in failing to falsify a given proposition can be viewed as a formal proof of the proposition. The proposition at the root of the tree is the logical conclusion of a set of inferences, and it is more natural to draw a proof tree in such a way that each premise in a rule occurs above its conclusion. However, this may be a matter of taste (and perhaps, aesthetics).

In the rest of this section, it will be assumed that the set of connectives used is $\{\wedge, \vee, \supset, \neg\}$, and that $(A \equiv B)$ is an abbreviation for $(A \supset B) \wedge (B \supset A)$, and $\perp$ an abbreviation for $(P \wedge \neg P)$.

Definition 3.4.2 The Gentzen system G'. The symbols Γ, Δ, Λ will be used to denote arbitrary sequences of propositions and A, B to denote propositions. The *inference rules* of the sequent calculus G' are the following:

$$\frac{\Gamma, A, B, \Delta \to \Lambda}{\Gamma, A \wedge B, \Delta \to \Lambda}\ (\wedge : left) \qquad \frac{\Gamma \to \Delta, A, \Lambda \quad \Gamma \to \Delta, B, \Lambda}{\Gamma \to \Delta, A \wedge B, \Lambda}\ (\wedge : right)$$

$$\frac{\Gamma, A, \Delta \to \Lambda \quad \Gamma, B, \Delta \to \Lambda}{\Gamma, A \vee B, \Delta \to \Lambda}\ (\vee : left) \qquad \frac{\Gamma \to \Delta, A, B, \Lambda}{\Gamma \to \Delta, A \vee B, \Lambda}\ (\vee : right)$$

$$\frac{\Gamma, \Delta \to A, \Lambda \quad B, \Gamma, \Delta \to \Lambda}{\Gamma, A \supset B, \Delta \to \Lambda}\ (\supset: left) \qquad \frac{A, \Gamma \to B, \Delta, \Lambda}{\Gamma \to \Delta, A \supset B, \Lambda}\ (\supset: right)$$

$$\frac{\Gamma, \Delta \to A, \Lambda}{\Gamma, \neg A, \Delta \to \Lambda}\ (\neg : left) \qquad \frac{A, \Gamma \to \Delta, \Lambda}{\Gamma \to \Delta, \neg A, \Lambda}\ (\neg : right)$$

The name of every rule is stated immediately to its right. Every rule consists of one or two upper sequents called *premises* and of a lower sequent called the *conclusion*. The above rules are called *inference rules*. For every rule, the proposition to which the rule is applied is called the *principal formula*, the propositions introduced in the premises are called the *side formulae*, and the other propositions that are copied unchanged are called the *extra formulae*.

Note that every inference rule can be represented as a tree with two nodes if the rule has a single premise, or three nodes if the rule has two premises. In both cases, the root of the tree is labeled with the conclusion of the rule and the leaves are labeled with the premises. If the rule has a single premise, it is a tree of the form

$$\begin{array}{ll} (1) & S_1 \\ & \ \ | \\ (e) & S_2 \end{array}$$

where the premise labels the node with tree address 1, and the conclusion labels the node with tree address e. If it has two premises, it is a tree of the form

$$\begin{array}{ccc} (1)\ S_1 & & S_2\ (2) \\ & \diagdown \quad \diagup & \\ & S_3 & \\ & (e) & \end{array}$$

where the first premise labels the node with tree address 1, the second premise labels the node with tree address 2, and the conclusion labels the node with tree address e.

EXAMPLE 3.4.2

Consider the following instance of the $\supset$:left rule:

$$\frac{A, B \to P, D \qquad Q, A, B \to D}{A, (P \supset Q), B \to D}$$

In the above inference, $(P \supset Q)$ is the principal formula, P and Q are side formulae, and A, B, D are extra formulae.

A careful reader might have observed that the rules ($\supset$:left), ($\supset$:right), ($\neg$:left), and ($\neg$:right) have been designed in a special way. Notice that the side proposition added to the antecedent of an upper sequent is added at the front, and similarly for the side proposition added to the succedent of an upper sequent. We have done so to facilitate the generalization of the *search* procedure presented below to infinite sequents.

We will now prove that the above rules achieve the falsification procedure sketched in example 3.4.1.

3.4.3 Falsifiable and Valid Sequents

First, we extend the concepts of falsifiability and validity to sequents.

Definition 3.4.3 A sequent $A_1, ..., A_m \to B_1, ..., B_n$ is *falsifiable* iff there exists a valuation v such that

$$v \models (A_1 \wedge ... \wedge A_m) \wedge (\neg B_1 \wedge ... \wedge \neg B_n).$$

A sequent as above is *valid* iff for every valuation v,

$$v \models (A_1 \wedge ... \wedge A_m) \supset (B_1 \vee ... \vee B_n).$$

This is also denoted by

$$\models A_1, ..., A_m \to B_1, ..., B_n.$$

If $m = 0$, the sequent $\rightarrow B_1, ..., B_n$ is falsifiable iff the proposition $(\neg B_1 \wedge ... \wedge \neg B_n)$ is satisfiable, valid iff the proposition $(B_1 \vee ... \vee B_n)$ is valid. If $n = 0$, the sequent $A_1, ..., A_m \rightarrow$ is falsifiable iff the proposition $(A_1 \wedge ... \wedge A_m)$ is satisfiable, valid iff the proposition $(A_1 \wedge ... \wedge A_m)$ is not satisfiable. Note that a sequent $\Gamma \rightarrow \Delta$ is valid if and only if it is not falsifiable.

Lemma 3.4.1 For each of the rules given in definition 3.4.2, a valuation v falsifies the sequent occurring as the conclusion of the rule if and only if v falsifies at least one of the sequents occurring as premises. Equivalently, v makes the conclusion of a rule true if and only if v makes all premises of that rule true.

Proof: The proof consists in checking the truth tables of the logical connectives. We treat one case, leaving the others as an exercise. Consider the ($\supset$:left) rule:

$$\frac{\Gamma, \Delta \rightarrow A, \Lambda \qquad B, \Gamma, \Delta \rightarrow \Lambda}{\Gamma, (A \supset B), \Delta \rightarrow \Lambda}$$

For every valuation v, v falsifies the conclusion if and only if v satisfies all propositions in Γ and Δ, and satisfies $(A \supset B)$, and falsifies all propositions in Λ. From the truth table of $(A \supset B)$, v satisfies $(A \supset B)$ if either v falsifies A, or v satisfies B. Hence, v falsifies the conclusion if and only if, either

(1) v satisfies Γ and Δ, and falsifies A and Λ, or

(2) v satisfies B, Γ and Δ, and falsifies Λ. $\square$

3.4.4 Axioms, Deduction Trees, Proof Trees, Counter Example Trees

The central concept in any proof system is the notion of *proof*. First, we define the axioms of the system G'.

Definition 3.4.4 An *axiom* is any sequent $\Gamma \rightarrow \Delta$ such that Γ and Δ contain some common proposition.

Lemma 3.4.2 No axiom is falsifiable. Equivalently, every axiom is valid.

Proof: The lemma follows from the fact that in order to falsify an axiom, a valuation would have to make some proposition true on the left hand side, and that same proposition false on the right hand side, which is impossible. $\square$

Proof trees are given by the following inductive definition.

Definition 3.4.5 The set of *proof trees* is the least set of trees containing all one-node trees labeled with an axiom, and closed under the rules of definition 3.4.2 in the following sense:

(1) For any proof tree T_1 whose root is labeled with a sequent $\Gamma \to \Delta$, for any instance of a one-premise inference rule with premise $\Gamma \to \Delta$ and conclusion $\Lambda \to \Theta$, the tree T whose root is labeled with $\Lambda \to \Theta$ and whose subtree $T/1$ is equal to T_1 is a proof tree.

(2) For any two proof trees T_1 and T_2 whose roots are labeled with sequents $\Gamma \to \Delta$ and $\Gamma' \to \Delta'$ respectively, for every instance of a two-premise inference rule with premises $\Gamma \to \Delta$ and $\Gamma' \to \Delta'$ and conclusion $\Lambda \to \Theta$, the tree T whose root is labeled with $\Lambda \to \Theta$ and whose subtrees $T/1$ and $T/2$ are equal to T_1 and T_2 respectively is a proof tree.

The set of *deduction trees* is defined inductively as the least set of trees containing *all* one-node trees (not necessarily labeled with an axiom), and closed under (1) and (2) as above.

A deduction tree such that some leaf is labeled with a sequent $\Gamma \to \Delta$ where Γ, Δ consist of propositional letters and are disjoint is called a *counter-example tree*. The sequent labeling the root of a proof tree (deduction tree) is called the *conclusion* of the proof tree (deduction tree). A sequent is *provable* iff there exists a proof tree of which it is the conclusion. If a sequent $\Gamma \to \Delta$ is provable, this is denoted by

$$\vdash \Gamma \to \Delta.$$

EXAMPLE 3.4.3

The deduction tree below is a proof tree.

$$\dfrac{\dfrac{\dfrac{P, \neg Q \to P}{\neg Q \to \neg P, P}}{\to P, (\neg Q \supset \neg P)} \qquad \dfrac{\dfrac{Q \to Q, \neg P}{\neg Q, Q \to \neg P}}{Q \to (\neg Q \supset \neg P)}}{\dfrac{(P \supset Q) \to (\neg Q \supset \neg P)}{\to (P \supset Q) \supset (\neg Q \supset \neg P)}}$$

The above tree is a proof tree obtained from the proof tree

$$\dfrac{\dfrac{\dfrac{P, \neg Q \to P}{\neg Q \to \neg P, P}}{\to P, (\neg Q \supset \neg P)} \qquad \dfrac{\dfrac{Q \to Q, \neg P}{\neg Q, Q \to \neg P}}{Q \to (\neg Q \supset \neg P)}}{(P \supset Q) \to (\neg Q \supset \neg P)}$$

and the rule

$$\frac{(P \supset Q) \to (\neg Q \supset \neg P)}{\to (P \supset Q) \supset (\neg Q \supset \neg P)}$$

In contrast, the deduction tree below is a counter-example tree.

$$\cfrac{\cfrac{P \to Q}{\to (P \supset Q)} \qquad \cfrac{\cfrac{\cfrac{Q \to P}{Q, \neg P \to}}{\neg P \to \neg Q}}{\to (\neg P \supset \neg Q)}}{\to (P \supset Q) \land (\neg P \supset \neg Q)}$$

The above tree is obtained from the two counter-example trees

$$\cfrac{P \to Q}{\to (P \supset Q)} \qquad \cfrac{\cfrac{\cfrac{Q \to P}{Q, \neg P \to}}{\neg P \to \neg Q}}{\to (\neg P \supset \neg Q)}$$

and the rule

$$\frac{\to (P \supset Q) \qquad \to (\neg P \supset \neg Q)}{\to (P \supset Q) \land (\neg P \supset \neg Q)}$$

It is easily shown that a deduction tree T is a proof tree if and only if every leaf sequent of T is an axiom.

Since proof trees (and deduction trees) are defined inductively, the induction principle applies. As an application, we now show that every provable sequent is valid.

3.4.5 Soundness of the Gentzen System G'

Lemma 3.4.3 Soundness of the system G'. If a sequent $\Gamma \to \Delta$ is provable, then it is valid.

Proof: We use the induction principle applied to proof trees. By lemma 3.4.2, every one-node proof tree (axiom) is valid. There are two cases in the induction step.

Case 1: The root of the proof tree T has a single descendant. In this case, T is obtained from some proof tree T_1 and some instance of a rule

$$\frac{S_1}{S_2}$$

By the induction hypothesis, S_1 is valid. Since by Lemma 3.4.1, S_1 is valid if and only if S_2 is valid, Lemma 3.4.3 holds.

Case 2: The root of the proof tree T has two descendants. In this case, T is obtained from two proof trees T_1 and T_2 and some instance of a rule

$$\frac{S_1 \qquad S_2}{S_3}$$

By the induction hypothesis, both S_1 and S_2 are valid. Since by lemma 3.4.1, S_3 is valid if and only if both S_1 and S_2 are, lemma 3.4.3 holds. $\square$

Next, we shall prove the fundamental theorem for the propositional sequent calculus G'. Roughly speaking, the fundamental theorem states that there exists a procedure for constructing a candidate counter-example tree, and that this procedure always terminates (is an algorithm). If the original sequent is valid, the algorithm terminates with a tree which is in fact a proof tree. Otherwise, the counter-example tree yields a falsifying valuation (in fact, all falsifying valuations). The fundamental theorem implies immediately the completeness of the sequent calculus G'.

3.4.6 The Search Procedure

The algorithm searching for a candidate counter-example tree builds this tree in a systematic fashion. We describe an algorithm that builds the tree in a breadth-first fashion. Note that other strategies for building such a tree could be used, (depth-first, in particular). A breadth-first expansion strategy was chosen because it is the strategy that works when we generalize the *search* procedure to infinite sequents. We will name this algorithm the *search procedure*.

Let us call a leaf of a tree *finished* iff the sequent labeling it is either an axiom, or all propositions in it are propositional symbols. We assume that a boolean function named *finished* testing whether a leaf is finished is available. A proposition that is a propositional symbol will be called *atomic*, and other propositions will be called *nonatomic*. A tree is finished when all its leaves are finished.

The procedure *search* traverses all leaves of the tree from left to right as long as not all of them are finished. For every unfinished leaf, the procedure *expand* is called. Procedure *expand* builds a subtree by applying the appropriate inference rule to every nonatomic proposition in the sequent labeling that leaf (proceeding from left to right). When the tree is finished, that is when all leaves are finished, either all leaves are labeled with axioms or some of the leaves are falsifiable. In the first case we have a proof tree, and in the second, all falsifying valuations can be found.

Definition 3.4.6 Procedure *search*. The input to *search* is a one-node tree labeled with a sequent $\Gamma \rightarrow \Delta$. The output is a finished tree T called a *systematic deduction tree*.

Procedure Search

```
procedure search(Γ → Δ : sequent; var T : tree);
  begin
    let T be the one-node tree labeled with  Γ → Δ;
    while not all leaves of T are finished  do
      T0 := T;
      for each leaf node of T0
      (in lexicographic order of tree addresses) do
        if not finished(node) then
          expand(node, T)
        endif
      endfor
    endwhile;
    if all leaves are axioms
    then
      write ('T is a proof of  Γ → Δ')
    else
      write ('Γ → Δ is falsifiable')
    endif
  end
```

Procedure Expand

```
procedure expand(node : tree-address; var T : tree);
  begin
    let A1, ..., Am → B1, ..., Bn  be the label of node;
    let S be the one-node tree labeled with
      A1, ..., Am → B1, ..., Bn;
      for i := 1 to m do
        if nonatomic(Ai) then
          S := the new tree obtained from S by
          applying to the descendant of Ai in
          every nonaxiom leaf of S the
          left rule applicable to  Ai;
        endif
      endfor;
      for i := 1 to n do
        if nonatomic(Bi) then
          S := the new tree obtained from S by
          applying to the descendant of Bi in
          every nonaxiom leaf of S the
          right rule applicable to  Bi;
        endif
      endfor;
      T := dosubstitution(T, node, S)
  end
```

The function *dosubstitution* yields the tree $T[node \leftarrow S]$ obtained by substituting the tree S at the address *node* in the tree T. Since a sequent $A_1, ..., A_m \to B_1, ..., B_n$ is processed from left to right, if the propositions $A_1, ..., A_{i-1}$ have been expanded so far, since the propositions A_i, ..., A_m, B_1, ..., B_n are copied unchanged, every leaf of the tree S obtained so far is of the form $\Gamma, A_i, ..., A_m \to \Delta, B_1, ..., B_n$. The occurrence of A_i following Γ is called the descendant of A_i in the sequent. Similarly, if the propositions $A_1, ..., A_m, B_1, ..., B_{i-1}$ have been expanded so far, every leaf of S is of the form $\Gamma \to \Delta, B_i, ..., B_n$, and the occurrence of B_i following Δ is called the descendant of B_i in the sequent. Note that the two *for* loops may yield a tree S of depth $m + n$.

A call to procedure *expand* is also called an *expansion step*. A *round* is the sequence of expansion calls performed during the *for* loop in procedure *search*, in which each unfinished leaf of the tree (T_0) is expanded. Note that if we change the function *finished* so that a leaf is finished if all propositions in the sequent labeling it are propositional symbols, the procedure *search* will produce trees in which leaves labeled with axioms also consist of sequents in which all propositions are atomic. Trees obtained in this fashion are called *atomically closed*.

EXAMPLE 3.4.4

Let us trace the construction of the systematic deduction tree (which is a proof tree) for the sequent

$$(P \land \neg Q), (P \supset Q), (T \supset R), (P \land S) \to T.$$

The following tree is obtained at the end of the first round:

$$\dfrac{\Gamma \to \Delta \qquad \dfrac{\dfrac{\dfrac{Q, P, \neg Q, P, S \to T, T}{Q, P, \neg Q, (P \land S) \to T, T} \qquad \dfrac{R, Q, P, \neg Q, P, S \to T}{R, Q, P, \neg Q, (P \land S) \to T}}{Q, P, \neg Q, (T \supset R), (P \land S) \to T}}{}}{\dfrac{P, \neg Q, (P \supset Q), (T \supset R), (P \land S) \to T}{(P \land \neg Q), (P \supset Q), (T \supset R), (P \land S) \to T}}$$

where $\Gamma \to \Delta = P, \neg Q, (T \supset R), (P \land S) \to P, T$.

The leaves $Q, P, \neg Q, P, S \to T, T$ and $R, Q, P, \neg Q, P, S \to T$ are not axioms (yet). Note how the same rule was applied to $(P \land S)$ in the nodes labeled $Q, P, \neg Q, (P \land S) \to T, T$ and $R, Q, P, \neg Q, (P \land S) \to T$, because these occurrences are descendants of $(P \land S)$ in $(P \land \neg Q)$,$(P \supset Q)$,$(T \supset R)$, $(P \land S) \to T$. After the end of the second round, we have the following tree, which is a proof tree.

$$
\dfrac{\dfrac{\Gamma \to \Delta \qquad \dfrac{\dfrac{\dfrac{Q,P,P,S \to Q,T,T}{Q,P,\neg Q,P,S \to T,T}}{Q,P,\neg Q,(P \land S) \to T,T} \qquad \dfrac{\dfrac{R,Q,P,P,S \to Q,T}{R,Q,P,\neg Q,P,S \to T}}{R,Q,P,\neg Q,(P \land S) \to T}}{Q,P,\neg Q,(T \supset R),(P \land S) \to T}}{P,\neg Q,(P \supset Q),(T \supset R),(P \land S) \to T}}{(P \land \neg Q),(P \supset Q),(T \supset R),(P \land S) \to T}
$$

where $\Gamma \to \Delta = P,\neg Q,(T \supset R),(P \land S) \to P,T$.

It should also be noted that the algorithm of definition 3.4.6 could be made more efficient. For example, during a round through a sequent we could delay the application of two-premise rules. This can be achieved if a two-premise rule is applied only if a sequent does not contain propositions to which a one-premise rule applies. Otherwise the *expand* procedure is called only for those propositions to which a one-premise rule applies. In this fashion, smaller trees are usually obtained. For more details, see problem 3.4.13.

3.4.7 Completeness of the Gentzen System G'

We are now ready to prove the fundamental theorem of this section.

Theorem 3.4.1 The procedure *search* terminates for every finite input sequent. If the input sequent $\Gamma \to \Delta$ is valid, procedure *search* produces a proof tree for $\Gamma \to \Delta$; if $\Gamma \to \Delta$ is falsifiable, *search* produces a tree from which all falsifying valuations can be found.

Proof: Define the *complexity* of a proposition A as the number of logical connectives occurring in A (hence, propositional symbols have complexity 0). Given a sequent $A_1, ..., A_m \to B_1, ..., B_n$, define its *complexity* as the sum of the complexities of $A_1, ..., A_m, B_1, ..., B_n$. Then, observe that for every call to procedure *expand*, the complexity of every upper sequent involved in applying a rule is strictly smaller than the complexity of the lower sequent (to which the rule is applied). Hence, either all leaves will become axioms or their complexity will become 0, which means that the *while* loop will always terminate. This proves termination. We now prove the following claim.

Claim: Given any deduction tree T, a valuation v falsifies the sequent $\Gamma \to \Delta$ labeling the root of T if and only if v falsifies some sequent labeling a leaf of T.

Proof of claim: We use the induction principle for deduction trees. In case of a one-node tree, the claim is obvious. Otherwise, the deduction tree is either of the form

$$\frac{\begin{array}{c} T_2 \\ S_2 \end{array}}{S_1}$$

where the bottom part of the tree is a one-premise rule, or of the form

$$\frac{\begin{array}{cc} T_2 & T_3 \\ S_2 & S_3 \end{array}}{S_1}$$

where the bottom part of the tree is a two-premise rule. We consider the second case, the first one being similar. By the induction hypothesis, v falsifies S_2 if and only if v falsifies some leaf of T_2, and v falsifies S_3 if and only if v falsifies some leaf of T_3. By lemma 3.4.1, v falsifies S_1 if and only if v falsifies S_2 or S_3. Hence, v falsifies S_1 if and only if either v falsifies some leaf of T_2 or some leaf of T_3, that is, v falsifies some leaf of T. $\square$

As a consequence of the claim, the sequent $\Gamma \rightarrow \Delta$ labeling the root of the deduction tree is valid, if and only if all leaf sequents are valid. It is easy to check that *search* builds a deduction tree (in a breadth-first fashion). Now, either $\Gamma \rightarrow \Delta$ is falsifiable, or it is valid. In the first case, by the above claim, if v falsifies $\Gamma \rightarrow \Delta$, then v falsifies some leaf sequent of the deduction tree T. By the definition of *finished*, such a leaf sequent must be of the form $P_1, ..., P_m \rightarrow Q_1, ..., Q_n$ where the P_i and Q_j are propositional symbols, and the sets $\{P_1, ..., P_m\}$ and $\{Q_1, ..., Q_n\}$ are disjoint since the sequent is not an axiom. Hence, if $\Gamma \rightarrow \Delta$ is falsifiable, the deduction tree T is not a proof tree. Conversely, if T is not a proof tree, some leaf sequent of T is not an axiom. By the definition of *finished*, this sequent must be of the form $P_1, ..., P_m \rightarrow Q_1, ..., Q_n$ where the P_i and Q_j are propositional symbols, and the sets $\{P_1, ..., P_m\}$ and $\{Q_1, ..., Q_n\}$ are disjoint. The valuation v which makes every P_i true and every Q_j false falsifies the leaf sequent $P_1, ..., P_m \rightarrow Q_1, ..., Q_n$, and by the above claim, it also falsifies the sequent $\Gamma \rightarrow \Delta$. Therefore, we have shown that $\Gamma \rightarrow \Delta$ is falsifiable if and only if the deduction tree T is not a proof tree, or equivalently, that $\Gamma \rightarrow \Delta$ is valid if and only if the deduction tree T is a proof tree. Furthermore, the above proof also showed that if the deduction tree T is not a proof tree, all falsifying valuations for $\Gamma \rightarrow \Delta$ can be found by inspecting the nonaxiom leaves of T. $\square$

Corollary Completeness of G'. Every valid sequent is provable. Furthermore, there is an algorithm for deciding whether a sequent is valid and if so, a proof tree is obtained.

As an application of the main theorem we obtain an algorithm to convert a proposition to conjunctive (or disjunctive) normal form.

3.4.8 Conjunctive and Disjunctive Normal Form

Definition 3.4.7 A proposition A is in *conjunctive normal form* (for short, CNF) if it is a conjunction $C_1 \wedge ... \wedge C_m$ of disjunctions $C_i = B_{i,1} \vee ... \vee B_{i,n_i}$, where each $B_{i,j}$ is either a propositional symbol P or the negation $\neg P$ of a propositional symbol. A proposition A is in *disjunctive normal form* (for short, DNF) if it is a disjunction $C_1 \vee ... \vee C_m$ of conjunctions $C_i = B_{i,1} \wedge ... \wedge B_{i,n_i}$, where each $B_{i,j}$ is either a propositional symbol P or the negation $\neg P$ of a propositional symbol.

Theorem 3.4.2 For every proposition A, a proposition A' in conjunctive normal form can be found such that $\models A \equiv A'$. Similarly, a proposition A'' in disjunctive normal form can be found such that $\models A \equiv A''$.

Proof: Starting with the input sequent $\rightarrow A$, let T be the tree given by the algorithm *search*. By theorem 3.4.1, either A is valid in which case all leaves are axioms, or A is falsifiable. In the first case, let $A' = P \vee \neg P$. Clearly, $\models A \equiv A'$. In the second case, the proof of theorem 3.4.1 shows that a valuation v makes A true if and only if it makes every leaf sequent true. For every nonaxiom leaf sequent $A_1, ..., A_m \rightarrow B_1, ..., B_n$, let

$$C = \neg A_1 \vee ... \vee \neg A_m \vee B_1 ... \vee B_n$$

and let A' be the conjunction of these propositions. Clearly, a valuation v makes A' true if and only if it makes every nonaxiom leaf sequent $A_1, ..., A_m \rightarrow B_1, ..., B_n$ true, if and only if it makes A true. Hence, $\models A \equiv A'$.

To get an equivalent proposition in disjunctive normal form, start with the sequent $A \rightarrow$. Then, a valuation v makes $A \rightarrow$ false if and only if v makes at least some of the sequent leaves false. Also, v makes $A \rightarrow$ false if and only if v makes A true. For every nonaxiom sequent leaf $A_1, ..., A_m \rightarrow B_1, ..., B_n$, let

$$C = A_1 \wedge ... \wedge A_m \wedge \neg B_1 \wedge ... \wedge \neg B_n$$

and let A'' be the disjunction of these propositions. We leave as an exercise to check that a valuation v makes some of the non-axiom leaves false if and only if it makes the disjunction A'' true. Hence $\models A \equiv A''$. $\square$

EXAMPLE 3.4.5

Counter-example tree for

$$\rightarrow (\neg P \supset Q) \supset (\neg R \supset S).$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{P \to R, S}{\to R, S, \neg P}}{\neg R \to S, \neg P}}{\to \neg P, \neg R \supset S} \qquad \dfrac{\dfrac{Q \to R, S}{\neg R, Q \to S}}{Q \to \neg R \supset S}}{\neg P \supset Q \to \neg R \supset S}}{\to (\neg P \supset Q) \supset (\neg R \supset S)}$$

An equivalent proposition in conjunctive normal form is:

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

EXAMPLE 3.4.6

Counter-example tree for

$$(\neg P \supset Q) \supset (\neg R \supset S) \to .$$

$$\dfrac{\dfrac{\dfrac{\to P, Q}{\neg P \to Q}}{\to \neg P \supset Q} \qquad \dfrac{\dfrac{R \to}{\to \neg R} \qquad S \to}{\neg R \supset S \to}}{(\neg P \supset Q) \supset (\neg R \supset S) \to}$$

An equivalent proposition in disjunctive normal form is:

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

We present below another method for converting a proposition to conjunctive normal form that does not rely on the construction of a deduction tree. This method is also useful in conjunction with the resolution method presented in Chapter 4. First, we define the negation normal form of a proposition.

3.4.9 Negation Normal Form

The set of propositions in negation normal form is given by the following inductive definition.

Definition 3.4.8 The set of propositions in *negation normal form* (for short, NNF) is the inductive closure of the set of propositions $\{P, \neg P \mid P \in \mathbf{PS}\}$ under the constructors $C_\vee$ and $C_\wedge$.

More informally, propositions in NNF are defined as follows:

(1) For every propositional letter P, P and $\neg P$ are in NNF;

(2) If A and B are in NNF, then $(A \vee B)$ and $(A \wedge B)$ are in NNF.

Lemma 3.4.4 Every proposition is equivalent to a proposition in NNF.

Proof: By theorem 3.3.1, we can assume that the proposition A is expressed only in terms of the connectives $\vee$ and $\wedge$ and $\neg$. The rest of the proof proceeds by induction on the number of connectives. By clause (1) of definition 3.4.8, every propositional letter is in NNF. Let A be of the form $\neg B$. If B is a propositional letter, by clause (1) of definition 3.4.8, the property holds. If B is of the form $\neg C$, by lemma 3.3.6, $\neg\neg C$ is equivalent to C, by the induction hypothesis, C is equivalent to a proposition C' in NNF, and by lemma 3.3.5, A is equivalent to C', a proposition in NNF. If B is of the form $(C \vee D)$, by lemma 3.3.6, $\neg(C \vee D)$ is equivalent to $(\neg C \wedge \neg D)$. Note that both $\neg C$ and $\neg D$ have fewer connectives than A. Hence, by the induction hypothesis, $\neg C$ and $\neg D$ are equivalent to propositions C' and D' in NNF. By lemma 3.3.5, A is equivalent to $(C' \wedge D')$, which is in NNF. If B is of the form $(C \wedge D)$, by lemma 3.3.6, $\neg(C \wedge D)$ is equivalent to $(\neg C \vee \neg D)$. As in the previous case, by the induction hypothesis, $\neg C$ and $\neg D$ are equivalent to propositions C' and D' in NNF. By lemma 3.3.5, A is equivalent to $(C' \vee D')$, which is in NNF. Finally, if A is of the form $(B * C)$ where $* \in \{\wedge, \vee\}$, by the induction hypothesis, C and D are equivalent to propositions C' and D' in NNF, and by lemma 3.3.5, A is equivalent to $(C' * D')$ which is in NNF. $\square$

Lemma 3.4.5 Every proposition A (containing only the connectives $\vee$,$\wedge$,$\neg$) can be transformed into an equivalent proposition in conjunctive normal form, by application of the following identities:

$$
\begin{aligned}
\neg\neg A &\simeq A \\
\neg(A \wedge B) &\simeq (\neg A \vee \neg B) \\
\neg(A \vee B) &\simeq (\neg A \wedge \neg B) \\
A \vee (B \wedge C) &\simeq (A \vee B) \wedge (A \vee C) \\
(B \wedge C) \vee A &\simeq (B \vee A) \wedge (C \vee A) \\
(A \wedge B) \wedge C &\simeq A \wedge (B \wedge C) \\
(A \vee B) \vee C &\simeq A \vee (B \vee C)
\end{aligned}
$$

Proof: The proof of lemma 3.4.4 only uses the first three tautologies. Hence, given a proposition A, we can assume that it is already in NNF. We prove by induction on propositions that a proposition in NNF can be converted to a proposition in CNF using the last four tautologies. If A is either of the form P or $\neg P$, we are done. If A is of the form $(B \vee C)$ with B and C in NNF, by the induction hypothesis both B and C are equivalent to propositions B' and C' in CNF. If both B' and C' consist of a single conjunct, $(B' \vee C')$ is a disjunction of propositional letters or negations of propositional letters and by lemma 3.3.5, A is equivalent to $(B' \vee C')$ which is in CNF. Otherwise, let $B' = B'_1 \wedge ... \wedge B'_m$ and $C' = C'_1 \wedge ... \wedge C'_n$, with either $m > 1$ or $n > 1$. By repeated applications of the distributivity and associativity rules (to be rigorous, by induction on $m + n$),

$$\begin{aligned}(B' \vee C') &= (B'_1 \wedge ... \wedge B'_m) \vee (C'_1 \wedge ... \wedge C'_n)\\ &\simeq ((B'_1 \wedge ... \wedge B'_m) \vee C'_1) \wedge ... \wedge ((B'_1 \wedge ... \wedge B'_m) \vee C'_n)\\ &\simeq \bigwedge\{(B'_i \vee C'_j) \mid 1 \leq i \leq m,\ 1 \leq j \leq n\}.\end{aligned}$$

The resulting proposition is in CNF, and by lemma 3.3.5, A is equivalent to a proposition in CNF. If A is of the form $(B \wedge C)$ where B and C are in NNF, by the induction hypothesis, B and C are equivalent to propositions B' and C' in CNF. But then, $(B' \wedge C')$ is in CNF, and by lemma 3.4.5, A is equivalent to $(B' \wedge C')$. □

The conjunctive normal form of a proposition may be simplified by using the commutativity rules and the idempotency rules given in lemma 3.3.6. A lemma similar to lemma 3.4.5 can be shown for the disjunctive normal form of a proposition.

EXAMPLE 3.4.7

Consider the proposition

$$A = (\neg P \supset Q) \supset (\neg R \supset S).$$

First, we eliminate $\supset$ using the fact that $(\neg B \vee C)$ is equivalent to $(B \supset C)$. We get

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

Then, we put this proposition in NNF. We obtain

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

Using distributivity we obtain

$$(\neg P \vee R \vee S) \wedge (\neg Q \vee R \vee S),$$

which is the proposition obtained in example 3.4.5 (up to commutativity). However, note that the CNF (or DNF) of a proposition is generally not unique. For example, the propositions

$$(P \vee Q) \wedge (\neg P \vee R) \quad \text{and} \quad (P \vee Q) \wedge (\neg P \vee R) \wedge (Q \vee R)$$

are both in CNF and are equivalent.

PROBLEMS

3.4.1. Give proof trees for the following tautologies:

$$A \supset (B \supset A)$$
$$(A \supset B) \supset ((A \supset (B \supset C)) \supset (A \supset C))$$
$$A \supset (B \supset (A \wedge B))$$
$$A \supset (A \vee B) \qquad B \supset (A \vee B)$$
$$(A \supset B) \supset ((A \supset \neg B) \supset \neg A)$$
$$(A \wedge B) \supset A \qquad (A \wedge B) \supset B$$
$$(A \supset C) \supset ((B \supset C) \supset ((A \vee B) \supset C))$$
$$\neg\neg A \supset A$$

3.4.2. Using counter-example trees, give propositions in conjunctive and disjunctive normal form equivalent to the following propositions:

$$(A \supset C) \supset ((B \supset D) \supset ((A \vee B) \supset C))$$
$$(A \supset B) \supset ((B \supset \neg C) \supset \neg A)$$

3.4.3. Recall that $\perp$ is a constant symbol always interpreted as **F**.

(i) Show that the following equivalences are valid.

$$\neg A \equiv (A \supset \perp)$$
$$(A \vee B) \equiv ((A \supset \perp) \supset B)$$
$$(A \equiv B) \equiv (A \supset B) \wedge (B \supset A)$$
$$(A \wedge B) \equiv \neg(\neg A \vee \neg B)$$

(ii) Show that every proposition A is equivalent to a proposition A' using only the connective $\supset$ and the constant symbol $\perp$.

(iii) Consider the following Gentzen-like rules for propositions over the language consisting of the propositional letters, $\supset$ and $\perp$.

The symbols Γ, Δ, Λ denote finite arbitrary sequences of propositions (possibly empty):

$$\frac{\Gamma, \Delta \rightarrow A, \Lambda \quad B, \Gamma, \Delta \rightarrow \Lambda}{\Gamma, (A \supset B), \Delta \rightarrow \Lambda} \qquad \frac{A, \Gamma \rightarrow B, \Delta, \Lambda}{\Gamma \rightarrow \Delta, (A \supset B), \Lambda}$$

$$\frac{\Gamma \rightarrow \Delta, \Lambda}{\Gamma \rightarrow \Delta, \perp, \Lambda}$$

The axioms of this system are all sequents of the form $\Gamma \rightarrow \Delta$ where Γ and Δ contain a common proposition, and all sequents of the form $\Gamma, \perp, \Delta \rightarrow \Lambda$.

(a) Prove that for every valuation v, the conclusion of a rule is falsifiable if and only if one of the premises is falsifiable, and that the axioms are not falsifiable.

(b) Prove that the above Gentzen-like system is complete.

(c) Convert $(P \supset Q) \supset (\neg Q \supset \neg P)$ to a proposition involving only $\supset$ and $\perp$. Give a proof tree for this proposition in the above system.

3.4.4. Let C and D be propositions and P a propositional letter. Prove that the following equivalence (the resolution rule) holds by giving a proof tree:

$$(C \vee P) \wedge (D \vee \neg P) \equiv (C \vee P) \wedge (D \vee \neg P) \wedge (C \vee D)$$

Show that the above also holds if either C or D is missing.

3.4.5. Give Gentzen-like rules for equivalence ($\equiv$).

3.4.6. Give proof trees for the following tautologies:

Associativity rules:

$$((A \vee B) \vee C) \equiv (A \vee (B \vee C)) \quad ((A \wedge B) \wedge C) \equiv (A \wedge (B \wedge C))$$

Commutativity rules:

$$(A \vee B) \equiv (B \vee A) \quad (A \wedge B) \equiv (B \wedge A)$$

Distributivity rules:

$$(A \vee (B \wedge C)) \equiv ((A \vee B) \wedge (A \vee C))$$
$$(A \wedge (B \vee C)) \equiv ((A \wedge B) \vee (A \wedge C))$$

De Morgan's rules:

$$\neg(A \vee B) \equiv (\neg A \wedge \neg B) \quad \neg(A \wedge B) \equiv (\neg A \vee \neg B)$$

Idempotency rules:

$$(A \vee A) \equiv A \quad (A \wedge A) \equiv A$$

Double negation rule:

$$\neg\neg A \equiv A$$

Absorption rules:

$$(A \vee (A \wedge B)) \equiv A \quad (A \wedge (A \vee B)) \equiv A$$

Laws of zero and one:

$$(A \vee \perp) \equiv A \quad (A \wedge \perp) \equiv \perp$$
$$(A \vee \top) \equiv \top \quad (A \wedge \top) \equiv A$$
$$(A \vee \neg A) \equiv \top \; (A \wedge \neg A) \equiv \perp$$

3.4.7. Instead of defining logical equivalence ($\simeq$) semantically as in definition 3.3.6, let us define $\simeq$ proof-theoretically so that, for all $A, B \in PROP$,

$$A \simeq B \text{ if and only if } \vdash (A \supset B) \wedge (B \supset A) \text{ in } G'.$$

Prove that $\simeq$ is an equivalence relation satisfying the properties of lemma 3.3.5 (Hence, $\simeq$ is a congruence).

3.4.8. Give Gentzen-like rules for the connective $\oplus$ (exclusive-or), where $H_{\oplus}$ is the binary truth function defined by the proposition $(P \wedge \neg Q) \vee (\neg P \wedge Q)$.

$*$ **3.4.9.** The *Hilbert system* H for propositional logic is defined below. For simplicity, it is assumed that only the connectives $\wedge$, $\vee$, $\supset$ and $\neg$ are used.

The *axioms* are all propositions given below, where A,B,C denote arbitrary propositions.

$$A \supset (B \supset A)$$
$$(A \supset B) \supset ((A \supset (B \supset C)) \supset (A \supset C))$$
$$A \supset (B \supset (A \wedge B))$$
$$A \supset (A \vee B), \qquad B \supset (A \vee B)$$
$$(A \supset B) \supset ((A \supset \neg B) \supset \neg A)$$
$$(A \wedge B) \supset A, \qquad (A \wedge B) \supset B$$
$$(A \supset C) \supset ((B \supset C) \supset ((A \vee B) \supset C))$$
$$\neg\neg A \supset A$$

There is a single inference rule, called *modus ponens* given by:

$$\frac{A \qquad (A \supset B)}{B}$$

Let $\{A_1, ..., A_m\}$ be any set of propositions. The concept of a *deduction tree* (in the system H) for a proposition B from the set $\{A_1, ..., A_m\}$ is defined inductively as follows:

(i) Every one-node tree labeled with an axiom B or a proposition B in $\{A_1, ..., A_m\}$ is a deduction tree of B from $\{A_1, ..., A_m\}$.

(ii) If T_1 is a deduction tree of A from $\{A_1, ..., A_m\}$ and T_2 is a deduction tree of $(A \supset B)$ from $\{A_1, ..., A_m\}$, then the following tree is a deduction tree of B from $\{A_1, ..., A_m\}$:

$$\frac{\dfrac{T_1}{A} \qquad\qquad \dfrac{T_2}{(A \supset B)}}{B}$$

A proof tree is a deduction tree whose leaves are labeled with axioms. Given a set $\{A_1, ..., A_m\}$ of propositions and a proposition B, we use the notation $A_1, ..., A_m \vdash B$ to denote that there is deduction of B from $\{A_1, ..., A_m\}$. In particular, if the set $\{A_1, ..., A_m\}$ is empty, the tree is a proof tree and we write $\vdash B$.

(i) Prove that *modus ponens* is a sound rule, in the sense that if both premises are valid, then the conclusion is valid. Prove that the system H is sound; that is, every proposition provable in H is valid.

(ii) Prove that for every proposition A, $\vdash (A \supset A)$.

Hint: Use the axioms $A \supset (B \supset A)$ and $(A \supset B) \supset ((A \supset (B \supset C)) \supset (A \supset C))$.

(iii) Prove the following:

(a) $A_1, ..., A_m \vdash A_i$, for every $i, 1 \leq i \leq m$.
(b) If $A_1, ..., A_m \vdash B_i$ for every $i, 1 \leq i \leq m$ and $B_1, ..., B_m \vdash C$, then $A_1, ..., A_m \vdash C$.

∗ **3.4.10.** In this problem, we are also considering the proof system H of problem 3.4.9. The *deduction theorem* states that, for arbitrary propositions $A_1, ..., A_m, A, B$,

$$\text{if } A_1, ..., A_m, A \vdash B, \text{ then}$$
$$A_1, ..., A_m \vdash (A \supset B).$$

Prove the deduction theorem.

Hint: Use induction on deduction trees. The base case is relatively easy. For the induction step, assume that the deduction tree is of the form

$$\frac{\begin{matrix} T_1 & & T_2 \\ B_1 & & (B_1 \supset B) \end{matrix}}{B}$$

where the leaves are either axioms or occurrences of the propositions A_1,...,A_m,A. By the induction hypothesis, there are deduction trees T_1' for $(A \supset B_1)$ and T_2' for $(A \supset (B_1 \supset B))$, where the leaves of T_1' and T_2' are labeled with axioms or the propositions $A_1, ..., A_m$. Show how a deduction tree whose leaves are labeled with axioms or the propositions $A_1, ..., A_m$ can be obtained for $(A \supset B)$.

∗ **3.4.11.** In this problem, we are still considering the proof system H of problem 3.4.9. Prove that the following meta-rules hold about deductions in the system H: For all propositions A,B,C and finite sequence Γ of propositions (possibly empty), we have:

	Introduction	*Elimination*
$\supset$	If $\Gamma, A \vdash B$, then $\Gamma \vdash (A \supset B)$	$A, (A \supset B) \vdash B$
$\wedge$	$A, B \vdash (A \wedge B)$	$(A \wedge B) \vdash A$ $(A \wedge B) \vdash B$
$\vee$	$A \vdash (A \vee B)$	If $\Gamma, A \vdash C$ and $\Gamma, B \vdash C$ then $\Gamma, (A \vee B) \vdash C$
$\neg$	If $\Gamma, A \vdash B$ and $\Gamma, A \vdash \neg B$ then $\Gamma \vdash \neg A$ (reductio ad absurdum)	$\neg\neg A \vdash A$ (double negation elimination) $A, \neg A \vdash B$ (weak negation elimination)

Hint: Use problem 3.4.9(iii) and the deduction theorem.

∗ **3.4.12.** In this problem it is shown that the Hilbert system H is complete, by proving that for every Gentzen proof T of a sequent $\rightarrow A$, where A is any proposition, there is a proof in the system H.

(i) Prove that for arbitrary propositions $A_1, ..., A_m$, $B_1, ..., B_n$,

(a) in H, for $n > 0$,

$$A_1, ..., A_m, \neg B_1, ..., \neg B_n \vdash P \wedge \neg P \text{ if and only if}$$
$$A_1, ..., A_m, \neg B_1, ..., \neg B_{n-1} \vdash B_n, \text{ and}$$

(b) in H, for $m > 0$,

$$A_1, ..., A_m, \neg B_1, ..., \neg B_n \vdash P \wedge \neg P \text{ if and only if}$$
$$A_2, ..., A_m, \neg B_1, ..., \neg B_n \vdash \neg A_1.$$

(ii) Prove that for any sequent $A_1, ..., A_m \rightarrow B_1, ..., B_n$, if

$$A_1, ..., A_m \rightarrow B_1, ..., B_n$$

is provable in the Gentzen system G' then

$$A_1, ..., A_m, \neg B_1, ..., \neg B_n \vdash (P \wedge \neg P)$$

is a deduction in the Hilbert system H. Conclude that H is complete.

Hint: Use problem 3.4.11.

3.4.13. Consider the modification of the algorithm of definition 3.4.6 obtained by postponing applications of two-premise rules. During a round, a two-premise rule is applied only if a sequent does not contain propositions to which a one-premise rule applies. Otherwise, during a round the *expand* procedure is called only for those propositions to which a one-premise rule applies.

Show that theorem 3.4.1 still holds for the resulting algorithm. Compare the size of the proof trees obtained from both versions of the *search* algorithm, by trying a few examples.

3.4.14. Write a computer program (preferably in PASCAL or C) implementing the *search* procedure of definition 3.4.6.

3.5 Proof Theory for Infinite Sequents: Extended Completeness of G'

In this section, we obtain some important results for propositional logic (extended completeness, compactness, model existence) by generalizing the procedure *search* to infinite sequents.

3.5.1 Infinite Sequents

We extend the concept of a sequent $\Gamma \rightarrow \Delta$ by allowing Γ or Δ to be countably infinite sequences of propositions. The method of this section is very important because it can be rather easily adapted to show the completeness of a Gentzen system obtained by adding quantifier rules to G' for first-order logic (see Chapter 5).

By suitably modifying the *search* procedure, we can generalize the main theorem (theorem 3.4.1) and obtain both the extended completeness theorem and the compactness theorem. The procedure *search* is no longer an algorithm since it can go on forever in some cases. However, if the input sequent is valid, a finite proof will be obtained. Also, if the input sequent is falsifiable, a falsifying valuation will be (nonconstructively) obtained.

We will now allow sequents $\Gamma \rightarrow \Delta$ in which Γ or Δ can be countably infinite sequences. It is convenient to assume that Γ is written as $A_1, ..., A_m, ...$ (possibly infinite to the right) and that Δ is written as $B_1, ..., B_n, ...$ (possibly infinite to the right). Hence, a sequent will be denoted as

$$A_1, ..., A_m, ... \rightarrow B_1, ..., B_n, ...$$

where the lists on both sides of $\rightarrow$ are finite or (countably) infinite.

In order to generalize the *search* procedure, we need to define the functions *head* and *tail* operating on possibly infinite sequences. Let us denote the empty sequence as $<>$.

head$(<>) = <>$; otherwise *head*$(A_1, ..., A_m, ...) = A_1$.

$tail(<>) =<>$; otherwise $tail(A_1, ..., A_m, ...) = A_2, ..., A_m, ...$.

In particular, $tail(A_1) =<>$. The predicate *atomic* is defined such that $atomic(A)$ is true if and only if A is a propositional symbol.

3.5.2 The Search Procedure for Infinite Sequents

Every node of the systematic tree constructed by *search* is still labeled with a *finite* sequent $\Gamma \rightarrow \Delta$. We will also use two global variables L and R, which are possibly countably infinite sequences of propositions. The initial value of L is $tail(\Gamma_0)$ and the initial value of R is $tail(\Delta_0)$, where $\Gamma_0 \rightarrow \Delta_0$ is the initial sequent.

A leaf of the tree is an *axiom* (or is *closed*) iff its label $\Gamma \rightarrow \Delta$ is an axiom. A leaf is *finished* iff either

(1) it is closed, or

(2) the sequences L and R are empty and all propositions in Γ, and Δ are atomic. The new versions of procedures *search* and *expand* are given as follows.

Definition 3.5.1 Procedure *search*.

```
procedure search(Γ0 → Δ0 : sequent; var T : tree);
  begin
    L := tail(Γ0); Γ := head(Γ0);
    R := tail(Δ0); Δ := head(Δ0);
    let T be the one-node tree labeled with Γ → Δ;
    while not all leaves of T are finished do
      T0 := T;
      for each leaf node of T0
      (in lexicographic order of tree addresses) do
        if not finished(node) then
          expand(node, T)
        endif
      endfor;
      L := tail(L); R := tail(R)
    endwhile;
    if all leaves are closed
    then
      write ('T is a proof of Γ0 → Δ0')
    else
      write ('Γ0 → Δ0 is falsifiable')
    endif
  end
```

The input to *search* is a one-node tree labeled with a possibly infinite sequent $\Gamma \to \Delta$. Procedure *search* builds a possibly infinite systematic deduction tree using procedure *expand*.

Procedure *expand* is modified as follows: For every leaf u created during an expansion step, if $\Gamma \to \Delta$ is the label of u, the finite sequent $\Gamma \to \Delta$ is extended to $\Gamma, head(L) \to \Delta, head(R)$. At the end of a round, the heads of both L and R are deleted. Hence, every proposition will eventually be considered.

Procedure Expand

```
procedure expand(node : tree-address; var T : tree);
  begin
    let A1, ..., Am → B1, ..., Bn  be the label of node;
    let S be the one-node tree labeled with
      A1, ..., Am → B1, ..., Bn;
      for i := 1 to m do
        if nonatomic(Ai) then
          S := the new tree obtained from S by
          applying to the descendant of Ai in
          every nonaxiom leaf of S the
          left rule applicable to  Ai;
          (only the sequent part is modified,
          L and R are unchanged)
        endif
      endfor;
      for i := 1 to n do
        if nonatomic(Bi) then
          S := the new tree obtained from S by
          applying to the descendant of Bi in
          every nonaxiom leaf of S the
          right rule applicable to  Bi;
          (only the sequent part is modified,
          L and R are unchanged)
        endif
      endfor;
      for each nonaxiom leaf u of S do
        let Γ → Δ  be the label of u;
        Γ' := Γ, head(L);
        Δ' := Δ, head(R);
        create a new leaf u1, son of u,
        labeled with the sequent Γ' → Δ'
      endfor;
      T := dosubstitution(T, node, S)
  end
```

If *search* terminates with a systematic tree whose leaves are all closed, we say that *search* terminates with a *closed tree*. Note that a closed tree is finite by definition. We will also need the following definitions.

Definition 3.5.2 Given a possibly infinite sequent $\Gamma \to \Delta$, a valuation v *falsifies* $\Gamma \to \Delta$ if

$$v \models A$$

for every proposition A in Γ, and

$$v \models \neg B$$

for every proposition B in Δ. We say that $\Gamma \to \Delta$ is *falsifiable*.

A valuation v *satisfies* $\Gamma \to \Delta$ if, whenever

$$v \models A$$

for every proposition A in Γ, then there is some proposition B in Δ such that

$$v \models B.$$

We say that $\Gamma \to \Delta$ is *satisfiable*.

The sequent $\Gamma \to \Delta$ is *valid* if it is satisfied by every valuation. This is denoted by

$$\models \Gamma \to \Delta.$$

The sequent $\Gamma \to \Delta$ is *provable* if there exist finite subsequences $C_1, ..., C_m$ and $D_1, ..., D_n$ of Γ and Δ respectively, such that the finite sequent

$$C_1, ..., C_m \to D_1, ..., D_n$$

is provable. This is denoted by

$$\vdash \Gamma \to \Delta.$$

Note that if an infinite sequent is provable, then it is valid. Indeed, if $\Gamma \to \Delta$ is provable, some subsequent $C_1, ..., C_m \to D_1, ..., D_n$ is provable, and therefore valid. But this implies that $\Gamma \to \Delta$ is valid, since $D_1, ..., D_n$ is a subsequence of Δ.

EXAMPLE 3.5.1

Consider the sequent $\Gamma_0 \to \Delta_0$ where

$$\Gamma_0 =< P_0, (P_0 \supset P_1), (P_1 \supset P_2), ..., (P_i \supset P_{i+1}), ... >,$$

and

$$\Delta_0 =< Q, P_3 > .$$

Initially,

$$\begin{aligned}
&\Gamma =< P_0 >,\\
&L =< (P_0 \supset P_1), (P_1 \supset P_2), ..., (P_i \supset P_{i+1}), ... >,\\
&\Delta =< Q >, \quad \text{and}\\
&R =< P_3 > .
\end{aligned}$$

At the end of the first round, we have the following tree:

$$\frac{P_0, (P_0 \supset P_1) \rightarrow Q, P_3}{P_0 \rightarrow Q}$$

Note how $(P_0 \supset P_1)$, the head of L, was added in the premise of the top sequent, and P_3, the head of R, was added to the conclusion of the top sequent. At the end of this round,

$$L =< (P_1 \supset P_2), ..., (P_i \supset P_{i+1}), ... > \quad \text{and } R =<> .$$

At the end of the second round, we have:

$$\dfrac{\dfrac{P_0 \rightarrow P_0, Q, P_3 \qquad \dfrac{P_1, P_0, (P_1 \supset P_2) \rightarrow Q, P_3}{P_1, P_0 \rightarrow Q, P_3}}{P_0, (P_0 \supset P_1) \rightarrow Q, P_3}}{P_0 \rightarrow Q}$$

We have

$$L =< (P_2 \supset P_3), ..., (P_i \supset P_{i+1}), ... > \quad \text{and } R =<> .$$

At the end of the third round, we have the tree:

$$\dfrac{\dfrac{P_0 \rightarrow P_0, Q, P_3 \qquad \dfrac{\dfrac{P_1, P_0 \rightarrow P_1, Q, P_3 \qquad \dfrac{P_2, P_1, P_0, (P_2 \supset P_3) \rightarrow Q, P_3}{P_2, P_1, P_0 \rightarrow Q, P_3}}{P_1, P_0, (P_1 \supset P_2) \rightarrow Q, P_3}}{P_1, P_0 \rightarrow Q, P_3}}{P_0, (P_0 \supset P_1) \rightarrow Q, P_3}}{P_0 \rightarrow Q}$$

We have

$$L =< (P_3 \supset P_4), ..., (P_i \supset P_{i+1}), ... > \quad \text{and } R =<> .$$

At the end of the fourth round, we have the closed tree:

$$\dfrac{\dfrac{\Pi_1 \qquad \dfrac{\dfrac{P_1, P_0 \to P_1, Q, P_3 \qquad \dfrac{\dfrac{P_2, P_1, P_0 \to P_2, Q, P_3 \quad P_3, P_2, P_1, P_0 \to Q, P_3}{P_2, P_1, P_0, (P_2 \supset P_3) \to Q, P_3}}{P_2, P_1, P_0 \to Q, P_3}}{P_1, P_0, (P_1 \supset P_2) \to Q, P_3}}{P_1, P_0 \to Q, P_3}}{P_0, (P_0 \supset P_1) \to Q, P_3}}{P_0 \to Q}$$

where

$$\Pi_1 = P_0 \to P_0, Q, P_3.$$

The above tree is not quite a proof tree, but a proof tree can be constructed from it as follows. Starting from the root and proceeding from bottom-up, for every sequent at depth k in which a proposition of the form $head(L)$ or $head(R)$ was introduced, add $head(L)$ after the rightmost proposition of the premise of every sequent at depth less than k, and add $head(R)$ after the rightmost proposition of the conclusion of every sequent at depth less than k:

$$\dfrac{\dfrac{\Pi_2 \qquad \dfrac{\dfrac{\Pi_3 \qquad \dfrac{\dfrac{P_2, P_1, P_0 \to P_2, Q, P_3 \quad P_3, P_2, P_1, P_0 \to Q, P_3}{P_2, P_1, P_0, (P_2 \supset P_3) \to Q, P_3}}{P_2, P_1, P_0, (P_2 \supset P_3) \to Q, P_3}}{P_1, P_0, (P_1 \supset P_2), (P_2 \supset P_3) \to Q, P_3}}{P_1, P_0, (P_1 \supset P_2), (P_2 \supset P_3) \to Q, P_3}}{P_0, (P_0 \supset P_1), (P_1 \supset P_2), (P_2 \supset P_3) \to Q, P_3}}{P_0, (P_0 \supset P_1), (P_1 \supset P_2), (P_2 \supset P_3) \to Q, P_3}$$

with

$$\Pi_2 = P_0, (P_1 \supset P_2), (P_2 \supset P_3) \to P_0, Q, P_3$$
$$\text{and} \quad \Pi_3 = P_1, P_0, (P_2 \supset P_3) \to P_1, Q, P_3.$$

Then, delete duplicate nodes, obtaining a legal proof tree:

$$\dfrac{\Pi_2 \qquad \dfrac{\Pi_3 \qquad \dfrac{P_2, P_1, P_0 \rightarrow P_2, Q, P_3 \quad P_3, P_2, P_1, P_0 \rightarrow Q, P_3}{P_2, P_1, P_0, (P_2 \supset P_3) \rightarrow Q, P_3}}{P_1, P_0, (P_1 \supset P_2), (P_2 \supset P_3) \rightarrow Q, P_3}}{P_0, (P_0 \supset P_1), (P_1 \supset P_2), (P_2 \supset P_3) \rightarrow Q, P_3}$$

with

$$\Pi_2 = P_0, (P_1 \supset P_2), (P_2 \supset P_3) \rightarrow P_0, Q, P_3$$
$$\text{and} \quad \Pi_3 = P_1, P_0, (P_2 \supset P_3) \rightarrow P_1, Q, P_3.$$

EXAMPLE 3.5.2

Consider the sequent $\Gamma_0 \rightarrow \Delta_0$ where

$$\Gamma_0 =< P_0, (P_0 \supset P_1), (P_1 \supset P_2), ..., (P_i \supset P_{i+1}), ... >,$$

and

$$\Delta_0 =< Q > .$$

Note that the only difference is the absence of P_3 in the conclusion. This time, the *search* procedure does not stop. Indeed, the rightmost branch of the tree is infinite, since every sequent in it is of the form

$$P_n, P_{n-1}, ...P_1, P_0 \rightarrow Q.$$

Let U be the union of all the propositions occurring as premises in the sequents on the infinite branch of the tree, and V be the union of all the propositions occurring as conclusions in such sequents. We have

$$U = \{(P_0 \supset P_1), ..., (P_i \supset P_{i+1}), ..., P_0, P_1, ..., P_i, ...\},$$

and

$$V = \{Q\}.$$

The pair (U, V) can be encoded as a single set if we prefix every proposition in U with the letter "T" (standing for **true**), and every proposition in V with the letter "F" (standing for **false**). The resulting set

$$\{T(P_0 \supset P_1), ..., T(P_i \supset P_{i+1}), ..., TP_0, TP_1, ..., TP_i, ..., FQ\}$$

is a set having some remarkable properties, and called a *Hintikka set*. The crucial property of Hintikka sets is that they are always satisfiable. For instance, it is easy to see that the valuation such that $v(P_i) = \mathbf{T}$ for all $i \geq 0$ and $v(Q) = \mathbf{F}$ satisfies the above Hintikka set.

Roughly speaking, the new version of the *search* procedure is complete because:

(1) If the input sequent is valid, a proof tree can be constructed from the output tree (as in example 3.5.1);

(2) If the sequent is falsifiable, the output tree contains a path from which a Hintikka set can be constructed (as in example 3.5.2). Hence, a counter example exists.

In order to prove rigorously properties (1) and (2), we will need some auxiliary definitions and lemmas. First, we shall need the following result about infinite finite-branching trees known as König's lemma.

3.5.3 König's Lemma

Recall from Subsection 2.2.2 that a tree T is finite branching iff every node has finite outdegree (finite number of successors).

Lemma 3.5.1 (König's lemma) If T is a finite-branching tree with infinite domain, then there is some infinite path in T.

Proof: We show that an infinite path can be defined inductively. Let u_0 be the root of the tree. Since the domain of T is infinite and u_0 has a finite number of successors, one of the subtrees of u_0 must be infinite (otherwise, T would be finite). Let u_1 be the root of the leftmost infinite subtree of u_0. Now, assume by induction that a path $u_0, ..., u_n$ has been defined and that the subtree T/u_n is infinite. Since u_n has a finite number of successors and since T/u_n is infinite, using the same reasoning as above, u_n must have a successor which is the root of an infinite tree. Let u_{n+1} be the leftmost such node. It is clear that the above inductive construction yields an infinite path in T (in fact, the leftmost such path). $\square$

Remark: The above proof only shows the *existence* of an infinite path. In particular, since there is in general no effective way of testing whether a tree is infinite, there is generally no algorithm to find the above nodes.

In example 3.5.2, the two sets U and V play a crucial role since they yield a falsifying valuation. The union of U and V is a set having certain remarkable properties first investigated by Hintikka and that we now describe. For this, it is convenient to introduce the concept of a *signed formula* as in Smullyan, 1968.

3.5.4 Signed Formulae

Following Smullyan, we will define the concept of an *a-formula* and of a *b-formula*, and describe their components. Using this device greatly reduces the number of cases in the definition of a Hintikka set, as well as in some proofs.

Definition 3.5.3 A *signed formula* is any expression of the form TA or FA, where A is an arbitrary proposition. Given any sequent (even infinite) $\Gamma \to \Delta$, we define the *signed set of formulae*

$$\{TA \mid A \in \Gamma\} \cup \{FB \mid B \in \Delta\}.$$

Definition 3.5.4 *type-a and type-b signed formulae* and their *components* are defined in the following tables. If A is a signed formula of type a, it has two components denoted by A_1 and A_2. Similarly, if B is a formula of type b, it has two components denoted by B_1 and B_2.

Type-a formulae

A	A_1	A_2
$T(X \land Y)$	TX	TY
$F(X \lor Y)$	FX	FY
$F(X \supset Y)$	TX	FY
$T(\neg X)$	FX	FX
$F(\neg X)$	TX	TX

Type-b formulae

B	B_1	B_2
$F(X \land Y)$	FX	FY
$T(X \lor Y)$	TX	TY
$T(X \supset Y)$	FX	TY

Definition 3.5.5 A valuation v makes the signed formula TA **true** iff v makes A **true** and v makes FA **true** iff v makes A **false**. A valuation v *satisfies a signed set* S iff v makes every signed formula in S **true**.

Note that for any valuation, a signed formula A of type a is **true** if and only if *both* A_1 *and* A_2 are **true**. Accordingly, we also refer to an *a-formula* as a formula of *conjunctive* type. On the other hand, for any valuation, a signed formula B of type b is **true** if and only if *at least* one of B_1, B_2 is **true**. Accordingly, a *b-formula* is also called a formula of *disjunctive* type.

Definition 3.5.6 The *conjugate* of a signed formula is defined as follows: The conjugate of a formula TA is FA, and the conjugate of FA is TA.

3.5.5 Hintikka Sets

A Hintikka set is a set of signed formulae satisfying certain downward closure conditions that ensure that such a set is satisfiable.

Definition 3.5.7 A set S of signed formulae is a *Hintikka set* iff the following conditions hold:

(H1) No signed propositional letter and its conjugate are both in S.

(H2) If a signed *a-formula* A is in S then both A_1 and A_2 are in S.

(H3) If a signed *b-formula* B is in S then either B_1 is in S or B_2 is in S.

The following lemma shows that Hintikka sets arise when the *search* procedure does not produce a closed tree (recall that a closed tree is finite).

Lemma 3.5.2 Whenever the tree T constructed by the *search* procedure is not a closed tree, a Hintikka set can be extracted from T.

Proof: If T is not a closed tree, then either it is finite and some leaf is not an axiom, or it is infinite. If T is infinite, by lemma 3.5.1, there is an infinite path in T. In the first case, consider a path to some nonaxiom leaf, and in the second consider an infinite path. Let

$$S = \{TA \mid A \in U\} \cup \{FB \mid B \in V\}$$

be the set of signed formulae such that U is the union of all propositions occurring in the antecedent of each sequent in the chosen path, and V is the union of all propositions occurring in the succedent of each sequent in the chosen path. S is a Hintikka set.

(1) H1 holds. Since every atomic formula occurring in a sequent occurs in every path having this sequent as source, if S contains both TP and FP for some propositional letter P, some sequent in the path is an axiom. This contradicts the fact that either the path is finite and ends in a non-axiom, or is an infinite path.

(2) H2 and H3 hold. This is true because the definition of *a-components* and *b-components* mirrors the inference rules. Since all nonatomic propositions in a sequent $\Gamma \rightarrow \Delta$ on the chosen path are considered during the expansion phase, and since every proposition in the input sequent is eventually considered (as $head(L)$ or $head(R)$):

(i) For every proposition A in U, if A belongs to $\Gamma \rightarrow \Delta$ and TA is of type a, A_1 and A_2 are added to the successor of $\Gamma \rightarrow \Delta$ during the expansion step. More precisely, if A_1 (or A_2) is of the form TC_1 (or TC_2), C_1 (C_2) is added to the premise of the successor of $\Gamma \rightarrow \Delta$; if A_1 (A_2) is of the form FC_1 (FC_2), C_1 (C_2) is added to the conclusion of the successor of $\Gamma \rightarrow \Delta$. In both cases, A_1 and A_2 belong to S.

(ii) If A belongs to $\Gamma \rightarrow \Delta$ and TA is of type b, A_1 is added to the left successor of $\Gamma \rightarrow \Delta$, and A_2 is added to the right successor of $\Gamma \rightarrow \Delta$, during the expansion step. As in (i), more precisely, if A_1 (or A_2) is of the form TC_1 (TC_2), C_1 (C_2) is added to the premise of the left successor (right successor) of $\Gamma \rightarrow \Delta$; if A_1 (A_2) is of the form FC_1 (FC_2), C_1 (C_2) is added to the conclusion of the left successor (right successor) of $\Gamma \rightarrow \Delta$. Hence, either B_1 or B_2 belongs to S.

Properties (i) and (ii) also apply to the set V. This proves that S is a Hintikka set. □

The following lemma establishes the fundamental property of Hintikka sets.

Lemma 3.5.3 Every Hintikka set S is satisfiable.

Proof: We define a valuation v satisfying S as follows: For every signed propositional symbol TP in S let $v(P) = \mathbf{T}$; for every signed propositional symbol FP in S let $v(P) = \mathbf{F}$; for every propositional symbol P such that neither TP nor FP is in S, set arbitrarily $v(P) = \mathbf{T}$. By clause (H1) of a Hintikka set, v is well defined. It remains to show that v makes every signed formula TX or FX true (that is, in the first case X true and in the second case X false). This is shown by induction on the number of logical connectives in X. Since every signed formula is either of type a or of type b, there are two cases.

(1) If A of type a is in S, by (H2) both A_1 and A_2 are in S. But A_1 and A_2 have fewer connectives than A and so, the induction hypothesis applies. Hence, v makes both A_1 and A_2 true. This implies that v makes A true.

(2) If B of type b is in S, by (H3) either B_1 or B_2 is in S. Without loss of generality assume that B_1 is in S. Since B_1 has fewer connectives than B, the induction hypothesis applies. Hence, v makes B_1 true. This implies that v makes B true. □

3.5.6 Extended Completeness of the Gentzen System G'

We are now ready to prove the generalization of theorem 3.4.1.

Theorem 3.5.1 Given a sequent $\Gamma \to \Delta$, either

(1) $\Gamma \to \Delta$ is falsifiable and

 (i) If $\Gamma \to \Delta$ is infinite, then *search* runs forever building an infinite tree T, or

 (ii) If $\Gamma \to \Delta$ is finite, then *search* produces a finite tree T with some non-axiom leaf.

 In both cases, a falsifying valuation can be obtained from some path in the tree produced by procedure *search*; or

(2) $\Gamma \to \Delta$ is valid and *search* terminates with a closed tree T. In this case, there exist finite subsequences $C_1, ..., C_m$ and $D_1, ..., D_n$ of Γ and Δ respectively such that the sequent $C_1, ..., C_m \to D_1, ..., D_n$ is provable.

Proof: First, observe that if a subsequent of a sequent is valid, the sequent itself is valid. Also, if $\Gamma \to \Delta$ is infinite, *search* terminates if and only if the tree is closed. This last statement holds because any node that is not an axiom is not finished, since otherwise L and R would be empty, contradicting the fact that $\Gamma \to \Delta$ is infinite. Hence, at every step of the procedure *search*, some node is unfinished, and since the procedure *expand* adds at least one new node to the tree (when $head(L)$ is added to Γ and $head(R)$ is added to Δ), *search* builds an infinite tree. Consequently, if $\Gamma \to \Delta$ is infinite, either *search* halts with a closed tree, or it builds an infinite tree.

If *search* halts with a closed tree, let $C_1, ..., C_m$ be the initial subsequence of propositions in Γ that were deleted from Γ to obtain L, and $D_1, ..., D_n$ be the initial subsequence of propositions in Δ which were deleted from Δ to obtain R. A proof tree for a the finite sequent $C_1, ..., C_m \to D_1, ..., D_n$ can easily be obtained from T, using the technique illustrated in example 3.5.1.

First, starting from the root and proceeding bottom-up, for each node

$$\Gamma, head(L) \to \Delta, head(R)$$

at depth k created at the end of a call to procedure *expand*, add $head(L)$ after the rightmost proposition in the premise of every sequent at depth less than k, and add $head(R)$ after the rightmost proposition in the conclusion of every sequent at depth less than k, obtaining the tree T'. Then, a proof tree T'' for $C_1, ..., C_m \to D_1, ..., D_n$ is constructed from T' by deleting all duplicate nodes. The tree T'' is a proof tree because the same inference rules that have been used in T are used in T''. A proof similar to that of theorem 3.4.1 shows that $C_1, ..., C_m \to D_1, ..., D_n$ is valid and consequently that $\Gamma \to \Delta$ is valid.

Hence, if the *search* procedure halts with a closed tree, a subsequent of $\Gamma \to \Delta$ is provable, which implies that $\Gamma \to \Delta$ is provable (and consequently valid). Hence, if $\Gamma \to \Delta$ is falsifiable, either the *search* procedure halts with a finite nonclosed tree if $\Gamma \to \Delta$ is finite, or else *search* must go on forever if $\Gamma \to \Delta$ is infinite. If the tree is finite, some leaf is not an axiom, and consider the path to this leaf. Otherwise, let T be the infinite tree obtained in the limit. This tree is well defined since for every integer k, *search* will produce the subtree of depth k of T. Since T is infinite and finite branching, by König's lemma, there is an infinite path $u_0, u_1, ..., u_n, ...$ in T. By lemma 3.5.2, the set

$$S = \{TA \mid A \in U\} \cup \{FB \mid B \in V\}$$

of signed formulae such that U is the union of all propositions occurring in the antecedent of each sequent in the chosen path, and V is the union of all propositions occurring in the succedent of each sequent in the chosen path, is a Hintikka set. By lemma 3.5.3, S is satisfiable. But any valuation satisfying S falsifies $\Gamma \to \Delta$, and $\Gamma \to \Delta$ is falsifiable.

To summarize, if the *search* procedure halts with a closed tree, $\Gamma \to \Delta$ is provable, and therefore valid. Otherwise $\Gamma \to \Delta$ is falsifiable.

Conversely, if $\Gamma \to \Delta$ is valid, *search* must halt with a closed tree, since otherwise the above reasoning shows that a falsifying valuation can be found. But then, we have shown that $\Gamma \to \Delta$ is provable. If $\Gamma \to \Delta$ is falsifiable, *search* cannot halt with a closed tree, since otherwise $\Gamma \rightarrow \Delta$ would be provable, and consequently valid. But then, we have shown that a falsifying valuation can be found from the tree T. This concludes the proof of the theorem. □

We now derive some easy consequences of the main theorem. Since a provable sequent is valid, the following is an obvious corollary.

Theorem 3.5.2 (Extended completeness theorem for G') For every (possibly infinite) sequent $\Gamma \rightarrow \Delta$, $\Gamma \rightarrow \Delta$ is valid if and only if $\Gamma \rightarrow \Delta$ is provable.

3.5.7 Compactness, Model Existence, Consistency

Recall that a proposition A is satisfiable if some valuation makes it **true**.

Definition 3.5.8 A set Γ of propositions is *satisfiable* iff some valuation makes all propositions in Γ **true**.

Theorem 3.5.3 (Compactness theorem for G') For any (possibly infinite) set Γ of propositions, if every finite (nonempty) subset of Γ is satisfiable then Γ is satisfiable.

Proof: Assume Γ is not satisfiable. Viewing Γ as a sequence of propositions, it is clear that the sequent

$$\Gamma \rightarrow$$

is valid, and by theorem 3.5.1 there is a finite subsequence $A_1, ..., A_p$ of Γ such that

$$A_1, ..., A_p \rightarrow$$

is provable. But then, by lemma 3.4.3, $A_1, ..., A_p \rightarrow$ is valid, which means that $A_1, ..., A_p$ is not satisfiable contrary to the hypothesis. Hence Γ is satisfiable. □

Definition 3.5.9 A set Γ of propositions is *consistent* if there exists some proposition B such that the sequent $\Gamma \rightarrow B$ is not provable (that is, $A_1, ..., A_m \rightarrow B$ is not provable for any finite subsequence $A_1, ..., A_m$ of Γ). Otherwise, we say that Γ is *inconsistent*.

Theorem 3.5.4 (Model existence theorem for G') If a set Γ of propositions is consistent then it is satisfiable.

Proof: Assume Γ unsatisfiable. Hence, for every proposition B, the sequent

$$\Gamma \rightarrow B$$

is valid. By theorem 3.5.1, for every such B, there is a finite subsequence $A_1, ..., A_p$ of Γ such that the sequent

$$A_1, ..., A_p \rightarrow B$$

is provable. But then, Γ is not consistent, contrary to the hypothesis. $\square$

The converse of theorem 3.5.4 is also true.

Lemma 3.5.4 (Consistency lemma for G') If a set Γ of propositions is satisfiable then it is consistent.

Proof: Let v be a valuation such that

$$v \models A$$

for every proposition in Γ. Assume that Γ is inconsistent. Then,

$$\Gamma \rightarrow B$$

is provable for every proposition B, and in particular, there is a finite subsequence $A_1, ..., A_m$ of Γ such that

$$A_1, ..., A_m \rightarrow P \land \neg P$$

is provable (for some propositional symbol P). By lemma 3.4.3, $A_1, ..., A_m \rightarrow P \land \neg P$ is valid and, since the valuation v makes all propositions in Γ true, v should make $P \land \neg P$ true, which is impossible. Hence, Γ is consistent. $\square$

Note that if a set Γ of propositions is consistent, theorem 3.5.4 shows that the sequent $\Gamma \rightarrow$ is falsifiable. Hence, by theorem 3.5.1, a falsifying valuation can be obtained (in fact, all falsifying valuations can be obtained by considering all infinite paths in the counter-example tree).

One may view the goal of procedure *search* as the construction of Hintikka sets. If this goal fails, the original sequent was valid and otherwise, any Hintikka set yields a falsifying valuation. The decomposition of propositions into *a-components* or *b-components* is the basis of a variant of Gentzen systems called the tableaux system. For details, see Smullyan, 1968.

3.5.8 Maximal Consistent Sets

We conclude this section by discussing briefly the concept of maximal consistent sets. This concept is important because it can be used to give another proof of the completeness theorem (theorem 3.5.2).

Definition 3.5.10 A consistent set Γ of propositions is *maximally consistent* (or a *maximal consistent set*) iff, for every consistent set Δ, if $\Gamma \subseteq \Delta$, then $\Gamma = \Delta$. Equivalently, every proper superset of Γ is inconsistent.

The importance of maximal consistent sets lies in the following lemma.

Lemma 3.5.5 Every consistent set Γ is a subset of some maximal consistent set Δ.

Proof: If Γ is a consistent set, by theorem 3.5.4, it is satisfiable. Let v be a valuation satisfying Γ. Let Δ be the set

$$\{A \mid v \models A\}$$

of all propositions satisfied by v. Clearly, Γ is a subset of Δ. We claim that Δ is a maximal consistent set. First, by lemma 3.5.4, Δ is consistent since it is satisfied by v. It remains to show that it is maximally consistent. We proceed by contradiction. Assume that there is a consistent set Λ such that Δ is a proper subset of Λ. Since Λ is consistent, by theorem 3.5.4, it is satisfied by a valuation v'. Since Δ is a proper subset of Λ, there is a proposition A which is in Λ but not in Δ. Hence,

$$v \not\models A,$$

since otherwise A would be in Δ. But then,

$$v \models \neg A,$$

and $\neg A$ is in Δ. Since Δ is a subset of Λ, v' satisfies every proposition in Δ, and in particular

$$v' \models \neg A.$$

But since v' satisfies Λ, we also have

$$v' \models A,$$

which is impossible. Hence, Δ is indeed maximally consistent. $\square$

The above lemma was shown using theorem 3.5.4, but it can be shown more directly and without theorem 3.5.4. Actually, theorem 3.5.4 can be shown from lemma 3.5.5, and in turn, the completeness theorem can be shown from theorem 3.5.4. Such an approach to the completeness theorem is more traditional, but not as constructive, in the sense that it does not provide a procedure for constructing a deduction tree.

There is also a close relationship between maximally consistent sets and Hintikka sets. Indeed, by reformulating Hintikka sets as unsigned sets of propositions, it can be shown that every maximal consistent set is a Hintikka set. However, the converse is not true. Hintikka sets are more general (and in a sense more economical) than maximal consistent sets. For details, we refer the reader to the problems.

PROBLEMS

3.5.1. (i) Show that the infinite sequent $\Gamma \rightarrow \Delta$ where

$$\Gamma =< P_0, (P_0 \supset P_1), (P_1 \supset P_2), ..., (P_i \supset P_{i+1}), ... >$$

and

$$\Delta =< (P_1 \supset Q) >$$

is falsifiable.

(ii) Prove that for every $i > 0$, the sequent $\Gamma \rightarrow \Delta'$, where Γ is as above and $\Delta' =< (P_0 \supset P_i) >$ is provable.

3.5.2. The *cut rule* is the following inference rule:

$$\frac{\Gamma \rightarrow \Delta, A \qquad A, \Lambda \rightarrow \Theta}{\Gamma, \Lambda \rightarrow \Delta, \Theta}$$

A is called the *cut formula* of this inference.

Let $G' + \{cut\}$ be the formal system obtained by adding the cut rule to G'. The notion of a deduction tree is extended to allow the cut rule as an inference. A proof in G' is called a *cut-free* proof.

(i) Prove that for every valuation v, if v satisfies the premises of the cut rule, then it satisfies its conclusion.

(ii) Prove that if a sequent is provable in the system $G' + \{cut\}$, then it is valid.

(iii) Prove that if a sequent is provable in $G' + \{cut\}$, then it has a cut-free proof.

3.5.3. (i) Prove solely in terms of proofs in $G' + \{cut\}$ that a set Γ of propositions is inconsistent if and only if there is a proposition A such that both $\Gamma \rightarrow A$ and $\Gamma \rightarrow \neg A$ are provable in $G' + \{cut\}$. (For inspiration see Section 3.6.)

(ii) Prove solely in terms of proofs in $G' + \{cut\}$ that, $\Gamma \rightarrow A$ is not provable in $G' + \{cut\}$ if and only if $\Gamma \cup \{\neg A\}$ is consistent. (For inspiration see Section 3.6.)

Note: Properties (i) and (ii) also hold for the proof system G', but the author does not know of any proof not involving a proof-theoretic version of Gentzen's *cut elimination theorem*. The cut elimination theorem states that any proof in the system $G' + \{cut\}$ can be transformed to a proof in G' (without cut). The completeness theorem for G' provides a semantic proof of the cut elimination theorem. However, in order to show (i) and (ii) without using semantic arguments, it appears that one has to mimic Gentzen's original proof. (See Szabo, 1969.)

∗ **3.5.4.** A set Γ of propositions is said to be *complete* if, for every proposition A, either $\Gamma \to A$ or $\Gamma \to \neg A$ is provable, but not both. Prove that for any set Γ of propositions, the following are equivalent:

(i) The set

$$\{A \mid \vdash \Gamma \to A \text{ in } G'\}$$

is a maximal consistent set.

(ii) Γ is complete.

(iii) There is a single valuation v satisfying Γ.

(iv) There is a valuation v such that for all A,

$\Gamma \to A$ is provable (in G') if and only if $v \models A$.

3.5.5. Let Γ be a consistent set. Let $A_1, A_2, ..., A_n, ...$ be an enumeration of *all* propositions in $PROP$. Define the sequence Γ_n inductively as follows:

$$\Gamma_0 = \Gamma,$$

$$\Gamma_{n+1} = \begin{cases} \Gamma_n \cup \{A_{n+1}\} & \text{if } \Gamma_n \cup \{A_{n+1}\} \text{ is consistent;} \\ \Gamma_n & \text{otherwise.} \end{cases}$$

Let

$$\Delta = \bigcup_{n \geq 0} \Gamma_n.$$

Prove the following:

(a) Each Γ_n is consistent.

(b) Δ is consistent.

(c) Δ is maximally consistent.

Note that this exercise provides another proof of lemma 3.5.5 for the system $G' + \{cut\}$, not using the completeness theorem.

3.5.6. Prove that if a proposition A over the language using the logical connectives $\{\vee, \wedge, \supset, \neg\}$ is a tautology, then A contains some occurrence of either $\neg$ or $\supset$.

∗ **3.5.7.** Given a proposition A, its *immediate descendants* A_1 and A_2 are given by the following table:

Type-a formulae

A	A_1	A_2
$(X \wedge Y)$	X	Y
$\neg(X \vee Y)$	$\neg X$	$\neg Y$
$\neg(X \supset Y)$	X	$\neg Y$
$\neg(\neg X)$	X	X

Type-b formulae

B	B_1	B_2
$\neg(X \wedge Y)$	$\neg X$	$\neg Y$
$(X \vee Y)$	X	Y
$(X \supset Y)$	$\neg X$	Y

Note that neither propositional letters nor negations of propositional letters have immediate descendants.

Given a set S of propositions, let $Des(S)$ be the set of immediate descendants of propositions in S, and define S^n by induction as follows:

$$S^0 = S;$$

$$S^{n+1} = Des(S^n)$$

Let

$$S^* = \bigcup_{n \geq 0} S^n$$

be the union of all the S^n. Hintikka sets can also be defined without using signed formulae, in terms of immediate descendants:

A set S of propositions is a *Hintikka set* if the following conditions hold:

(H1) No propositional letter and its negation are both in S.

(H2) If an *a-formula* A is in S then both A_1 and A_2 are in S.

(H3) If a *b-formula* B is in S then either B_1 is in S or B_2 is in S.

In this problem and some of the following problems, given any set S of propositions and any propositions A_1,...,A_n, the set $S \cup \{A_1, ...A_n\}$ will also be denoted by $\{S, A_1, ..., A_n\}$.

Assume that S is consistent.

(a) Using a modification of the construction given in problem 3.5.5, show that S can be extended to a maximal consistent subset U of S^* (that is, to a consistent subset U of S^* containing S, such that U is not a proper subset of any consistent subset of S^*).

(b) Prove that consistent sets satisfy the following properties:

C_0: No set S containing a propositional letter and its negation is consistent.

C_1: If $\{S, A\}$ is consistent, so is $\{S, A_1, A_2\}$, where A is a proposition of type a.

C_2: If $\{S, B\}$ is consistent, then either $\{S, B_1\}$ or $\{S, B_2\}$ is consistent, where B is a proposition of type b.

(c) Prove that U is a Hintikka set.

(d) Show that U is not necessarily a maximal consistent subset of $PROP$, the set of all propositions.

∗ **3.5.8.** The purpose of this problem is to prove the compactness theorem for propositional logic without using the completeness theorem. The proof developed in the following questions is in a sense more constructive than the proof given by using the completeness theorem, since if we are given a set Γ such that every finite subset of Γ is satisfiable, we will actually construct a valuation that satisfies Γ. However, the existence of ultrafilters requires Zorn's Lemma, and so this proof is not constructive in the recursion-theoretic sense.

For this problem, you may use the following result stated below and known as Zorn's lemma. For details, the reader should consult a text on set Theory, such as Enderton, 1977; Suppes, 1972; or Kuratowski and Mostowski, 1976.

We recall the following concepts from Subsection 2.1.9. A chain in a poset $(P, \leq)$ is a totally ordered subset of P. A chain C is bounded if there exists an element $b \in P$ such that for all $p \in C$, $p \leq b$. A maximal element of P is some $m \in P$ such that for any $m' \in P$, if $m \leq m'$ then $m = m'$.

Zorn's lemma: Given a partially ordered set S, if every chain in S is bounded, then S has a maximal element.

(1) Let E be a nonempty set, and F a class of subsets of E. We say that F is a *filter* on E iff:

1. E is in F;

2. if u and v are in F, then $u \cap v$ is in F;

3. if u is in F and v is any subset of E, if $u \subseteq v$, then v is also in F.

A filter F is a *proper filter* if $\emptyset$ (the empty set) is not in F. A proper filter F is *maximal* if, for any other proper filter D, if F is a subset of D, then $D = F$.

A class $\mathcal{C}$ (even empty) of subsets of a nonempty set E has the *finite intersection property* ($f.i.p.$) iff the intersection of every finite number of sets in $\mathcal{C}$ is nonempty. Let $\mathcal{C}$ be any class of subsets of a nonempty set E. The *filter generated by* $\mathcal{C}$ is the intersection D of all filters over E which include $\mathcal{C}$.

Prove the following properties:

(i) The filter D generated by $\mathcal{C}$ is indeed a filter over E.

(ii) D is equal to the set of all subsets X of E such that either $X = E$, or for some $Y_1, ..., Y_n \in \mathcal{C}$,

$$Y_1 \cap ... \cap Y_n \subseteq X.$$

(iii) D is a proper filter if and only if $\mathcal{C}$ has the finite intersection property.

(2) A maximal proper filter is called an *ultrafilter*.

Prove that a nonempty collection U of sets with the finite intersection property is an ultrafilter over E if and only if, for every subset X of E,

$$X \in U \quad \text{if and only if} \quad (E - X) \notin U.$$

Hint: Assume that $(E - X) \notin U$. Let $D = U \cup \{X\}$, and let F be the filter generated by D (as in question 1). Show that F is a proper filter including U. Hence, $U = F$ and D is a subset of U, so that $X \in U$.

(3) Use Zorn's lemma to show that if a class $\mathcal{C}$ of subsets of a nonempty set E has the finite intersection property, then it is contained in some ultrafilter.

Hint: Show that the union of a chain of proper filters is a proper filter that bounds the chain.

$*$ **3.5.9.** Let I be a nonempty set and $V = \{v_i \mid i \in I\}$ be a set of valuations. Let U be a proper filter on I. Define the valuation v such that for each propositional symbol $P \in \mathbf{PS}$,

$$v(P) = \mathbf{T} \quad \text{iff} \quad \{i \mid v_i(P) = \mathbf{T}\} \in U.$$

(Such a valuation v is called a *reduced product*).

(a) Show that

$$v(P) = \mathbf{F} \quad \text{iff} \quad \{i \mid v_i(P) = \mathbf{T}\} \notin U,$$

and

$$\text{if} \quad \{i \mid v_i(P) = \mathbf{T}\} = \emptyset \quad \text{then} \quad v(P) = \mathbf{F}.$$

If U is an ultrafilter, show that for all propositions A,

$$v \models A \quad \text{iff} \quad \{i \mid v_i \models A\} \in U.$$

Such a valuation v is called the *ultraproduct* of V with respect to U.

(b) Show that for any Horn formula A (see problem 3.3.18), whenever U is a proper filter, if

$$\{i \mid v_i \models A\} \in U \quad \text{then} \quad v \models A.$$

As a consequence, show that for every Horn formula A,

$$\text{if } v_i \models A \text{ for all } i \in I, \text{ then } v \models A.$$

(c) Let $I = \{1, 2\}$. Give all the filters on I. Give an example showing that there exists a proper filter U on $\{1, 2\}$, a set of valuations $\{v_1, v_2\}$, and a proposition A, such that $v \models A$, but $\{i \mid v_i \models A\} \notin U$.

(d) Consider the proper filter $U = \{\{1, 2\}\}$ on $I = \{1, 2\}$, and let $A = P_1 \vee P_2$. Find two valuations v_1 and v_2 such that $v_1 \models A$ and $v_2 \models A$, but the reduced product v of v_1 and v_2 with respect to U does not satisfy A. Conclude that not every proposition is logically equivalent to a Horn formula.

$*$ **3.5.10.** (a) Let Γ be a set of propositions such that every finite subset of Γ is satisfiable. Let I be the set of all finite subsets of Γ, and for each $i \in I$, let v_i be a valuation satisfying i. For each proposition $A \in \Gamma$, let

$$A^* = \{i \in I \mid A \in i\}.$$

Let

$$\mathcal{C} = \{A^* \mid A \in \Gamma\}.$$

Note that $\mathcal{C}$ has the finite intersection property since

$$\{A_1, ..., A_n\} \in A_1^* \cap ... \cap A_n^*.$$

By problem 3.5.8, let U be an ultrafilter including $\mathcal{C}$, so that every A^* is in U. If $i \in A^*$, then $A \in i$, and so

$$v_i \models A.$$

Thus, for every A in Γ, A^* is a subset of $\{i \in I \mid v_i \models A\}$.

Show that each set $\{i \in I \mid v_i \models A\}$ is in U.

(b) Show that the ultraproduct v (defined in problem 3.5.9) of the set of valuations $\{v_i \mid i \in I\}$ with respect to the ultrafilter U satisfies Γ.

$*$ **3.5.11.** Recall the definition of a Horn formula given in problem 3.3.18. Given a countable set $\{v_i \mid i \geq 0\}$ of truth assignments, the product v of $\{v_i \mid i \geq 0\}$ is the truth assignment such that for every propositional symbol P_j,

$$v(P_j) = \begin{cases} \mathbf{T} & \text{if } v_i(P_j) = \mathbf{T}, \text{ for all } v_i, \\ \mathbf{F} & \text{otherwise.} \end{cases}$$

(a) Show that if X is a set of propositional Horn formulae and every truth assignment in $\{v_i \mid i \geq 0\}$ satisfies X, then the product v satisfies X.

(b) Let

$$X^* = \{\neg P \mid P \text{ is atomic and } X \not\vdash P\}.$$

Show that if X is a consistent set of basic Horn formulas, then $X \cup X^*$ is consistent.

Hint: Using question (a), show that there is a truth assignment v satisfying $X \cup X^*$.

$*$ **3.5.12.** In this problem, we are using the definitions given in problem 3.5.7. Given a set S, a property P about subsets of S (P is a subset of 2^S) is a *property of finite character* iff the following hold:

Given any subset U of S, P holds for U if and only if P holds for all finite subsets of U.

A property P about sets of propositions is a *consistency property* if P is of finite character and the following hold:

C_0: No set S containing a propositional letter and its negation satisfies P.

C_1: If $\{S, A\}$ satisfies P, so does $\{S, A_1, A_2\}$, where A is a proposition of type a.

C_2: If $\{S, B\}$ satisfies P, then either $\{S, B_1\}$ or $\{S, B_2\}$ satisfies P, where B is a proposition of type b.

(a) Using Zorn's lemma (see problem 3.5.7), show that for any set S, for any property P about subsets of S, if P is of finite character, then any subset U of S for which P holds is a subset of some maximal subset of S for which P holds.

(b) Prove that if P is a consistency property and P satisfies a set U of propositions, then U can be extended to a Hintikka set.

Hint: Use the technique described in problem 3.5.7.

(c) Prove that if P is a consistency property and P satisfies a set U of propositions, then U is satisfiable.

$*$ **3.5.13.** Using the definitions given in problem 3.5.7, show that a maximal consistent set S is a Hintikka set satisfying the additional property:

$$M_0: \text{ For every proposition } A, \quad A \in S \quad \text{if and only if} \quad \neg A \notin S.$$

$*$ **3.5.14.** In this problem, we also use the definitions of problem 3.5.7. A set S of propositions is *downward closed* iff the following conditions hold:

D_1: For every proposition A of type a, if $A \in S$, then both A_1 and A_2 are in S.

D_2: For every proposition B of type b, if $B \in S$, then either B_1 is in S or B_2 is in S.

A set S of propositions is *upward closed* iff:

U_1: For every proposition A of type a, if A_1 and A_2 are both in S, then A is in S.

U_2: For every proposition B of type b, if either B_1 is in S or B_2 is in S, then B is in S.

(a) Prove that any downward closed set satisfying condition M_0 (given in problem 3.5.13) is a maximal consistent set.

(b) Prove that any upward closed set satisfying condition M_0 is a maximal consistent set.

Note: Conditions D_1 and D_2 are conditions $H2$ and $H3$ for Hintikka sets. Furthermore, U_1 and U_2 state the converse of D_1 and D_2. Hence, the above problem shows that a maximal consistent set is a set satisfying condition M_0 and the "if and only if" version of $H2$ and $H3$. Consequently, this reproves that a maximal consistent set is a Hintikka set.

In the next problems, some connections between logic and the theory of boolean algebras are explored.

$*$ **3.5.15.** Recall from Section 3.3 that a *boolean algebra* is a structure $\mathbf{A} =< A, +, *, \neg, 0, 1 >$, where A is a nonempty set, $+$ and $*$ are binary functions on A, $\neg$ is a unary function on A, and 0 and 1 are distinct elements of A, such that the following axioms hold:

Associativity rules:

$$((A + B) + C) = (A + (B + C)) \quad ((A * B) * C) = (A * (B * C))$$

Commutativity rules:

$$(A + B) = (B + A) \quad (A * B) = (B * A)$$

Distributivity rules:

$$(A + (B * C)) = ((A + B) * (A + C))$$
$$(A * (B + C)) = ((A * B) + (A * C))$$

De Morgan's rules:

$$\neg(A + B) = (\neg A * \neg B) \quad \neg(A * B) = (\neg A + \neg B)$$

Idempotency rules:

$$(A + A) = A \quad (A * A) = A$$

Double negation rule:

$$\neg\neg A = A$$

Absorption rules:

$$(A + (A * B)) = A \quad (A * (A + B)) = A$$

Laws of zero and one:

$$(A + 0) = A \quad (A * 0) = 0$$
$$(A + 1) = 1 \quad (A * 1) = A$$
$$(A + \neg A) = 1 \quad (A * \neg A) = 0$$

When dealing with boolean algebras, $\neg A$ is also denoted as $\overline{A}$. Given a boolean algebra $\mathbf{A}$, a partial ordering $\leq$ is defined on A as follows:

$$a \leq b \quad \text{if and only if} \quad a + b = b.$$

A *filter* D is a subset of A such that D is nonempty, for all $x, y \in D$, $x * y \in D$, and for all $z \in A$ and $x \in D$, if $x \leq z$, then $z \in D$. A *proper filter* is a filter such that $0 \notin D$ (equivalently, $D \neq A$). (Note that this is a generalization of the notion defined in problem 3.5.8.)

An *ideal* is a nonempty subset I of A such that, for all $x, y \in I$, $x + y \in I$, and for all $z \in A$ and $x \in I$, $x * z \in I$.

(a) Show that for any (proper) filter D, the set

$$\{\overline{x} \mid x \in D\}$$

is an ideal.

Given a filter D, the relation (D) defined by

$$x(D)y \quad \text{if and only if} \quad x * y + \overline{x} * \overline{y} \in D.$$

is a congruence relation on $\mathbf{A}$. The set of equivalences classes modulo (D) is a boolean algebra denoted by $\mathbf{A}/D$, whose 1 is D, and whose 0 is $\{\overline{x} \mid x \in D\}$.

(b) Prove that $x(D)y$ if and only if there is some $z \in D$ such that

$$x * z = y * z,$$

if and only if

$$x * \overline{y} + \overline{x} * y \in \{\overline{x} \mid x \in D\},$$

if and only if

$$(\overline{x} + y) * (\overline{y} + x) \in D.$$

Note: Intuitively speaking, $(\overline{x} + y)$ corresponds to the proposition $(x \supset y)$ and $(\overline{x} + y) * (\overline{y} + x)$ to $(x \equiv y)$.

A subset E of A has the *finite intersection property* iff for any finite number of elements $x_1, ..., x_n \in E$,

$$x_1 * ... * x_n \neq 0.$$

(c) Prove that every subset E with the finite intersection property is contained in a smallest proper filter. (See problem 3.5.8.)

A filter D is *principal* iff for some $a \neq 0$ in A, $x \in D$ if and only if $a \leq x$. A proper filter is an *ultrafilter* iff it is maximal.

(d) Prove that any set with the finite intersection property can be extended to an ultrafilter.

(e) If D is an ultrafilter, then

$$x + y \in D \quad \text{iff} \quad \text{either } x \in D \text{ or } y \in D,$$

$$x \in D \quad \text{iff} \quad \overline{x} \notin D.$$

Prove that D is an ultrafilter if and only if the quotient boolean algebra $\mathbf{A}/D$ is isomorphic to the two-element boolean algebra $BOOL$.

$*$ **3.5.16.** Let $\simeq$ be the proof-theoretic version of the equivalence relation on $PROP$ defined in problem 3.4.6, so that for any two propositions A and B,

$$A \simeq B \text{ if and only if } \vdash (A \equiv B) \text{ in } G'.$$

(a) Show that the set $\mathbf{B}_0$ of equivalence classes modulo $\simeq$ is a boolean algebra if we define the operations $+$, $*$ and $\overline{}$ on $\mathbf{B}_0$ as follows:

$$[A] + [B] = [A \vee B],$$
$$[A] * [B] = [A \wedge B],$$
$$\overline{[A]} = [\neg A].$$

Also, let $0 = [\bot]$ and $1 = [\top]$. Observe that $0 \neq 1$. The algebra $\mathbf{B}_0$ is called the *Lindenbaum algebra* of $PROP$.

Hint: Use problems 3.4.6 and 3.4.7.

(b) Prove that the following statements are equivalent:

(1) Every consistent set can be extended to a maximal consistent set.

(2) Every filter on $\mathbf{B}_0$ can be extended to an ultrafilter.

$*$ **3.5.17.** Let T be any subset of propositions in $PROP$. We say that T is *finitely axiomatizable* if there is a finite set S of propositions such that for every proposition A in $PROP$,

$$\vdash T \rightarrow A \text{ in } G' + \{cut\} \quad \text{if and only if} \quad \vdash S \rightarrow A \text{ in } G' + \{cut\}.$$

Let

$$D_T = \{[A] \mid \vdash T \rightarrow A \text{ in } G' + \{cut\}\},$$

where $[A]$ denotes the equivalence class of A modulo $\simeq$. Prove the following statements:

(i) T is consistent iff D_T is a proper filter on $\mathbf{B}_0$.

(ii) T is consistent and finitely axiomatizable iff D_T is a principal filter on $\mathbf{B}_0$.

(iii) T is complete iff D_T is an ultrafilter on $\mathbf{B}_0$. (For the definition of a complete set of propositions, See problem 3.5.4).

(iv) T is complete and finitely axiomatizable iff D_T is a principal ultrafilter on $\mathbf{B}_0$.

Given a subset D of $\mathbf{B}_0$, let

$$T_D = \{A \in PROP \mid [A] \in D\}.$$

Show that the converses of (i) to (iv) each hold, with T replaced by T_D and D_T by D.

Say that a set T of propositions is *closed* if, for every $A \in PROP$, $\vdash T \rightarrow A$ implies that $A \in T$. Show that there is a one-to-one correspondence between complete closed extensions of T and ultrafilters in $\mathbf{B}_0/D_T$.

Note: In this problem the cut rule seems necessary to prove that D_T is a filter, specifically, that if $\vdash T \rightarrow A$ and $\vdash T \rightarrow (A \supset B)$ in $G' + \{cut\}$, then $\vdash T \rightarrow B$ in $G' + \{cut\}$. To prove this in G', a form of Gentzen's cut elimination theorem seems necessary.

$*$ **3.5.18.** (a) Let $\mathbf{A}_1$ and $\mathbf{A}_2$ be two boolean algebras. A function $h : \mathbf{A}_1 \rightarrow \mathbf{A}_2$ is a homomorphism if, for all $x, y \in \mathbf{A}_1$,

$$\begin{aligned} h(x \vee_1 y) &= h(x) \vee_2 h(y), \\ h(x \wedge_1 y) &= h(x) \wedge_2 h(y) \text{ and} \\ h(\overline{x}) &= \overline{h(x)}. \end{aligned}$$

Show that $h(0) = 0$ and $h(1) = 1$.

(b) Given a boolean algebra $\mathbf{A}$ and a proper filter D, show that the mapping $h_D : \mathbf{A} \rightarrow \mathbf{A}/D$ that maps every element a of $\mathbf{A}$ to its equivalence class $[a]$ modulo (D) is a homomorphism.

(c) Let T be a consistent set of propositions. The equivalence relation $\simeq_T$ on $PROP$ is defined as follows:

$$A \simeq_T B \quad \text{if and only if} \quad \vdash T \rightarrow (A \equiv B) \text{ in } G' + \{cut\}.$$

Show that the set $\mathbf{B}_T$ of equivalence classes modulo $\simeq_T$ is a boolean algebra if we define the operations $+$, $*$ and $\overline{}$ on $\mathbf{B}_T$ as follows:

$$[A]_T + [B]_T = [A \lor B]_T,$$
$$[A]_T * [B]_T = [A \land B]_T,$$
$$\overline{[A]_T} = [\neg A]_T.$$

($[A]_T$ denotes the equivalence class of A modulo $\simeq_T$.) Furthermore, the element 1 is the equivalence class

$$\{A \mid \ \vdash T \to A \text{ in } G' + \{cut\}\},$$

and the element 0 is the class

$$\{A \mid \ \vdash T \to \neg A \text{ in } G' + \{cut\}\}.$$

The boolean algebra $\mathbf{B}_T$ is called the *Lindenbaum algebra of* T. Note that the equivalence class $[A]_T$ of A modulo $\simeq_T$ is the set

$$\{B \mid \ \vdash T \to (A \equiv B) \text{ in } G' + \{cut\}\}.$$

For any homomorphism $h : \mathbf{B}_T \to BOOL$, let $v : \mathbf{PS} \to BOOL$ be defined such that for every propositional letter P,

$$v(P) = h([P]_T).$$

Show that v is a valuation satisfying T such that $\widehat{v}(A) = h([A]_T)$ for all $A \in PROP$.

(d) There is a correspondence between valuations satisfying T and ultrafilters U in $\mathbf{B}_T$ defined as follows: For every ultrafilter U in $\mathbf{B}_T$, the quotient algebra $\mathbf{B}_T/U$ is isomorphic to the boolean algebra $BOOL$ (see problem 3.5.15(e)). By questions 3.5.18(a) and 3.5.18(b), there is a valuation v_U satisfying T induced by the homomorphism from $\mathbf{B}_T$ to $\mathbf{B}_T/U$. Conversely, if v is a valuation satisfying T, show that

$$U_v = \{[A]_T \mid \widehat{v}(A) = \mathbf{T}\}$$

is an ultrafilter in $\mathbf{B}_T$.

(e) Prove the extended completeness theorem for $G' + \{cut\}$.

Hint: Assume that $T \to A$ is valid, but that A is not provable from T. Then, in the Lindenbaum algebra $\mathbf{B}_T$, $[A]_T \neq 1$, and so $[\neg A]_T \neq 0$. Using problem 3.5.15(d), there is an ultrafilter U in $\mathbf{B}_T$ containing $[\neg A]_T$. Since $\mathbf{B}_T/U$ is isomorphic to $BOOL$, by questions 3.5.18(c) and 3.5.18(d), there is a valuation v satisfying T such that

$$\widehat{v}(\neg A) = h([\neg A]_T),$$

where h is the homormophism from $\mathbf{B}_T$ to $\mathbf{B}_T/U$. Since $[\neg A]_T$ is in U,

$$h([\neg A]_T) = \mathbf{T}.$$

Hence, there is a valuation satisfying T such that $\widehat{v}(A) = \mathbf{F}$. This contradicts the validity of $T \rightarrow A$.

3.5.19. Write a computer program (preferably in PASCAL or C) implementing the extended *search* procedure of definition 3.5.1.

3.6 More on Gentzen Systems: The Cut Rule

The rules of the Gentzen system G' given in definition 3.4.2 were chosen so as to give ourselves as few choices as possible at each step *upward* in searching systematically for a falsifying valuation. The use of other Gentzen-type systems may afford simpler proofs, especially working *downward*. One such system is the system LK' due to Gentzen. The system LK' contains a rule called the *cut rule*, which is important from a historical point of view, but also from a mathematical point of view. Indeed, even though it is possible to find complete proof systems not using the cut rule, we will discover some unexpected complications when we study first-order logic with equality. Indeed, the system for first-order logic with equality not using the cut rule is not very natural, and the cut rule cannot be dispensed with easily.

3.6.1 Using Auxiliary Lemmas in Proofs

There are also "pragmatic" reasons for considering the cut rule. The cut rule is the following:

$$\frac{\Gamma \rightarrow \Delta, A \qquad A, \Lambda \rightarrow \Theta}{\Gamma, \Lambda \rightarrow \Delta, \Theta}$$

A is called the *cut formula* of this inference.

Notice that this rule formalizes the technique constantly used in practice to use an *auxiliary lemma* in a proof. This is more easily visualized if we assume that Δ is empty. Then, $\Gamma \rightarrow A$ is the auxiliary lemma, which can be assumed to belong to a catalogue of already-proven results. Now, using A as an assumption, if we can show that using other assumptions Λ, that Θ is provable, we can conclude that $\Gamma, \Lambda \rightarrow \Theta$ is provable. The conclusion does not refer to A.

One might say that a proof using the cut rule is not as "direct" and consequently, not as perspicuous as a proof not using the cut rule. On the other hand, if we already have a vast catalogue of known results, and we can use them to give short and "easy" proofs of other results, why force ourselves not to use the convenience afforded by the cut rule? We shall not try to answer these questions of a philosophical nature. Let us just make a few remarks.

Let us call a proof not using the cut rule a cut-free proof. Cut-free proofs are important in investigations regarding consistency results. The object of such investigations is to establish constructively the consistency of mathematical theories such as arithmetic or set theory, the ultimate goal being to show that mathematics formalized as a logical theory is free of contradictions. First-order logic is simple enough that Gentzen's cut elimination theorem holds constructively. This means that for every proof using the cut rule, another proof not using the cut rule can effectively be constructed. We shall give a (nonconstructive) semantic proof of this result in this chapter, and a constructive proof for a simpler system in Chapter 6. From a mathematical point of view, this shows that the cut rule can be dispensed with. For richer logics, such as second-order logic, the cut-elimination theorem also holds, but not constructively, in the sense that the argument showing that there is a method for converting a proof with cut to a proof without cut is not effective.

Another interesting issue is to examine the relative complexity of proofs with or without cut. Proofs with cuts can be much shorter than cut-free proofs. This will be shown in Chapter 6. However, from the point of view of automatic theorem proving, cut-free proofs are easier to find. For more on cut-free proofs and the cut rule, the reader is referred to Takeuti, 1975, and Pfenning's paper in Shostack, 1984a.

We now present the system LK'.

3.6.2 The Gentzen System LK'

The system LK' consists of *structural rules*, the *cut rule*, and of *logical rules*.

Definition 3.6.1 Gentzen system LK'. The letters Γ,Δ,Λ,Θ stand for arbitrary (possibly empty) sequences of propositions and A,B for arbitrary propositions.

(1) Structural rules:

(i) Weakening:

$$\frac{\Gamma \to \Delta}{A, \Gamma \to \Delta}\ (left) \qquad \frac{\Gamma \to \Delta}{\Gamma \to \Delta, A}\ (right)$$

A is called the *weakening formula*.

(ii) Contraction:

$$\frac{A, A, \Gamma \to \Delta}{A, \Gamma \to \Delta}\ (left) \qquad \frac{\Gamma \to \Delta, A, A}{\Gamma \to \Delta, A}\ (right)$$

(iii) Exchange:

$$\frac{\Gamma, A, B, \Delta \to \Lambda}{\Gamma, B, A, \Delta \to \Lambda}\ (left) \qquad \frac{\Gamma \to \Delta, A, B, \Lambda}{\Gamma \to \Delta, B, A, \Lambda}\ (right)$$

(2) Cut rule:

$$\frac{\Gamma \to \Delta, A \qquad A, \Lambda \to \Theta}{\Gamma, \Lambda \to \Delta, \Theta}$$

A is called the *cut formula* of this inference.

(3) Logical rules:

$$\frac{A, \Gamma \to \Delta}{A \wedge B, \Gamma \to \Delta}\ (\wedge : left) \quad \text{and} \quad \frac{B, \Gamma \to \Delta}{A \wedge B, \Gamma \to \Delta}\ (\wedge : left)$$

$$\frac{\Gamma \to \Delta, A \qquad \Gamma \to \Delta, B}{\Gamma \to \Delta, A \wedge B}\ (\wedge : right)$$

$$\frac{A, \Gamma \to \Delta \qquad B, \Gamma \to \Delta}{A \vee B, \Gamma \to \Delta}\ (\vee : left)$$

$$\frac{\Gamma \to \Delta, A}{\Gamma \to \Delta, A \vee B}\ (\vee : right) \quad \text{and} \quad \frac{\Gamma \to \Delta, B}{\Gamma \to \Delta, A \vee B}\ (\vee : right)$$

$$\frac{\Gamma \to \Delta, A \qquad B, \Lambda \to \Theta}{A \supset B, \Gamma, \Lambda \to \Delta, \Theta}\ (\supset : left) \qquad \frac{A, \Gamma \to \Delta, B}{\Gamma \to \Delta, A \supset B}\ (\supset : right)$$

$$\frac{\Gamma \to \Delta, A}{\neg A, \Gamma \to \Delta}\ (\neg : left) \qquad \frac{A, \Gamma \to \Delta}{\Gamma \to \Delta, \neg A}\ (\neg : right)$$

In the rules above, the propositions $A \vee B$, $A \wedge B$, $A \supset B$ and $\neg A$ are called the *principal formulae* and the propositions A, B the *side formulae*.

The *axioms* of the system LK' are all sequents of the form

$$A \to A.$$

Note that in view of the exchange rule, the order of propositions in a sequent is really irrelevant, and the system LK' could be defined using multisets as defined in problem 2.1.8.

Proof trees are defined inductively as in definition 3.4.5, but with the rules of the system LK' given in definition 3.6.1. If a sequent has a proof in the system G' we say that it is *G'-provable* and similarly, if it is provable in the system LK', we say that it is *LK'-provable*. The system obtained by removing the cut rule from LK' will be denoted as $LK' - \{cut\}$. We also say that a sequent is LK'-provable without a cut if it has a proof tree using the rules of the system $LK' - \{cut\}$. We now show that the systems G' and LK' are logically equivalent. We will in fact prove a stronger result, namely that G', $LK' - \{cut\}$ and LK' are equivalent. First, we show that the system LK' is sound.

Lemma 3.6.1 (Soundness of LK') Every axiom of LK' is valid. For every rule of LK', for every valuation v, if v makes all the premises of a rule true then v makes the conclusion of the rule true. Every LK'-provable sequent is valid.

Proof: The proof uses the induction principle for proofs and is straightforward. □

Note that lemma 3.6.1 differs from lemma 3.4.3 in the following point: It is not true that if v makes the conclusion of a rule true then v makes all premises of that rule true. This reveals a remarkable property of the system G'. The system G' is a "two way" system, in the sense that the rules can be used either from top-down or from bottom-up. However, LK' is a top-down system. In order to ensure that the inferences are sound, the rules must be used from top-down.

3.6.3 Logical Equivalence of G', LK', and $LK' - \{cut\}$

The following theorem yields a semantic version of the cut elimination theorem.

Theorem 3.6.1 Logical equivalence of G', LK', and $LK' - \{cut\}$. There is an algorithm to convert any LK'-proof of a sequent $\Gamma \to \Delta$ into a G'-proof. There is an algorithm to convert any G'-proof of a sequent $\Gamma \to \Delta$ into a proof using the rules of $LK' - \{cut\}$.

Proof: If $\Gamma \to \Delta$ has an LK'-proof, by lemma 3.6.1, $\Gamma \to \Delta$ is valid. By theorem 3.5.2, $\Gamma \to \Delta$ has a G'-proof given by the algorithm *search*. Note that if $\Gamma \to \Delta$ is infinite, then the *search* procedure gives a proof for a finite subsequent of $\Gamma \to \Delta$, but by definition 3.6.2, it is a proof of $\Gamma \to \Delta$. Conversely, using the induction principle for G'-proofs we show that every G'-proof can be converted to an $(LK' - \{cut\})$-proof. This argument also applies to infinite sequents, since a proof of an infinite sequent is in fact a proof of some finite subsequent of it.

First, every G'-axiom $\Gamma \to \Delta$ contains some common proposition A, and by application of the weakening and the exchange rules, an $(LK' - \{cut\})$-proof of $\Gamma \to \Delta$ can be obtained from the axiom $A \to A$. Next, we have to show that every application of a G'-rule can be replaced by a sequence of $(LK' - \{cut\})$-rules. There are eight cases to consider. Note that the G'-rules $\land : right$, $\lor : left$, $\supset$: $right$, $\supset$: $left$, $\neg : right$ and $\neg : left$ can easily be simulated in $LK' - \{cut\}$ using the exchange, contraction, and corresponding $(LK' - \{cut\})$-rules. We show how the G'-rule $\land : left$ can be transformed to a sequence of $(LK' - \{cut\})$-rules, leaving the transformation of the G'-rule $\vee : right$ as an exercise. The following is an $(LK' - \{cut\})$-derivation from $\Gamma, A, B, \Delta \to \Lambda$ to $\Gamma, A \wedge B, \Delta \to \Lambda$.

$$\begin{array}{cl}
\Gamma, A, B, \Delta \to \Lambda & \\ \hline
\text{(several exchanges)} & \\ \hline
A, B, \Gamma, \Delta \to \Lambda & \\ \hline
A \wedge B, B, \Gamma, \Delta \to \Lambda & (\wedge : left\ (A)) \\ \hline
B, A \wedge B, \Gamma, \Delta \to \Lambda & \text{(exchange)} \\ \hline
A \wedge B, A \wedge B, \Gamma, \Delta \to \Lambda & (\wedge : left\ (B)) \\ \hline
A \wedge B, \Gamma, \Delta \to \Lambda & \text{(contraction)} \\ \hline
\text{(several exchanges)} & \\ \hline
\Gamma, A \wedge B, \Delta \to \Lambda &
\end{array}$$

□

3.6.4 Gentzen's Hauptsatz for LK' (Cut elimination theorem for LK')

Theorem 3.6.1 has the following important corollary.

Corollary (Gentzen's Hauptsatz for LK') A sequent is LK'-provable if and only if it is LK'-provable without a cut.

Note that the *search* procedure together with the above procedure provides an algorithm to construct a cut-free LK'-proof from an LK'-proof with cut. Gentzen proved the above result by a very different method in which an LK'-proof is (recursively) transformed into an LK'-proof without cut. Gentzen's proof is more structural and syntactical than ours, since we completely forget about the LK'-proof and start from scratch using the procedure *search*. Also, Gentzen's proof generalizes to the first-order predicate calculus LK, providing an algorithm for transforming any LK-proof with cut to an LK-proof without cut. The *search* procedure will also provide a cut-free proof, but the argument used in justifying the correctness of the *search* procedure is not constructive. The nonconstructive step arises when we show that the *search* procedure terminates for a valid sequent. Gentzen's proof is difficult and can be found in either Takeuti, 1975; Kleene, 1952; or in Gentzen's original paper in Szabo, 1969. A constructive proof for a simpler system (sequents of formulae in NNF) will also be given in Chapter 6.

3.6.5 Characterization of Consistency in LK'

The following lemma gives a characterization of consistency in the system LK'.

Lemma 3.6.2 (1) A set Γ of propositions is inconsistent if and only if there is some proposition A such that both $\Gamma \to A$ and $\Gamma \to \neg A$ are LK'-provable.

(2) For any proposition A, the sequent $\Gamma \to A$ is not LK'-provable if and only if $\Gamma \cup \{\neg A\}$ is consistent.

Proof: In this proof, we will abbreviate LK'-provable as provable.

(1) If Γ is inconsistent then $\Gamma \to B$ is provable for any proposition B, showing that the second half of (1) holds. Conversely, assume that for some A, both $\vdash \Gamma \to A$ and $\vdash \Gamma \to \neg A$ in LK', with proofs T_1 and T_2. The following is a proof of $\Gamma \to B$ for any given B.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{T_1}{\Gamma \to A}}{\neg A, \Gamma \to}\;(\neg : left)}{\Gamma \to \neg\neg A}\;(\neg : right) \qquad \dfrac{\dfrac{\dfrac{T_2}{\Gamma \to \neg A}}{\neg\neg A, \Gamma \to}\;(\neg : left)}{\neg\neg A, \Gamma \to B}\;(\text{weakening})}{\Gamma, \Gamma \to B}\;(cut\ (\neg\neg A))}{\Gamma \to B}\;(\text{contractions and exchanges})$$

(2) Assume that $\Gamma \to A$ is not provable. If $\Gamma \cup \{\neg A\}$ was inconsistent, then $\neg A, \Gamma \to A$ would be provable with proof T. The following is a proof of $\Gamma \to A$, contradicting the hypothesis.

$$\dfrac{\dfrac{\dfrac{\dfrac{T}{\neg A, \Gamma \to A}}{\Gamma \to A, \neg\neg A}\;(\neg : right) \qquad \dfrac{\dfrac{A \to A}{\to A, \neg A}\;(\neg : right)}{\neg\neg A \to A}\;(\neg : left)}{\Gamma \to A, A}\;(cut\ (\neg\neg A))}{\Gamma \to A}\;(\text{contraction})$$

Conversely, assume that $\Gamma \cup \{\neg A\}$ is consistent. If $\Gamma \to A$ is provable, a fortiori $\Gamma, \neg A \to A$ is provable. But $\neg A \to \neg A$ is also provable since it is an axiom, and so $\Gamma, \neg A \to \neg A$ is provable. By (1), $\Gamma \cup \{\neg A\}$ is inconsistent. $\square$

Remark: Recall that for an infinite set of propositions Γ, $\Gamma \to A$ is provable if $\Delta \to A$ is provable for a *finite* subsequence Δ of Γ. Hence, the above proofs should really be modified to refer to finite subsequences of Γ. Using the exchange, weakening and contraction rules, we can ensure that the antecedent in the conclusion of each proof is a subsequence of Γ. We leave the details as an exercise. Also, note that the above characterizations of consistency (or inconsistency) in LK' are purely syntactic (proof theoretic), and that the cut rule was used in a crucial way.

PROBLEMS

3.6.1. Give LK'-proof trees for the following tautologies:

$$A \supset (B \supset A)$$
$$(A \supset B) \supset ((A \supset (B \supset C)) \supset (A \supset C))$$
$$A \supset (B \supset (A \wedge B))$$
$$A \supset (A \vee B) \quad B \supset (A \vee B)$$
$$(A \supset B) \supset ((A \supset \neg B) \supset \neg A)$$
$$(A \wedge B) \supset A \quad (A \wedge B) \supset B$$
$$(A \supset C) \supset ((B \supset C) \supset ((A \vee B) \supset C))$$
$$\neg\neg A \supset A$$

3.6.2. Show that the cut rule is not a two-way rule, that is, if a valuation v satisfies the conclusion of the cut rule, it does not necessarily satisfy its premises. Find the other rules of LK' that are not two-way rules.

∗ **3.6.3.** Recall that a set Γ of propositions is *maximally consistent* if Γ is consistent and for any other set Δ, if Γ is a proper subset of Δ then Δ is inconsistent.

(a) Show that if Γ is maximally consistent, for any proposition A such that $\Gamma \rightarrow A$ is provable in LK' (with cut), A is in Γ.

(b) Show that

$$\Delta = \{A \mid \ \vdash \Gamma \rightarrow A \text{ in } LK' \text{ (with cut)}\}$$

is maximally consistent if and only if for every proposition A, either $\vdash \Gamma \rightarrow A$ or $\vdash \Gamma \rightarrow \neg A$ in LK', but not both.

(c) Show that Γ is maximally consistent iff there is a single valuation v satisfying Γ.

(d) Show that Γ is maximally consistent iff there is a valuation v such that $v \models A$ if and only if A is in Γ.

3.6.4. Using the technique of problem 3.5.5, prove in LK' $(+\{cut\})$ that every consistent set can be extended to a maximal consistent set.

∗ **3.6.5.** In this problem, we are adopting the definition of a Hintikka set given in problem 3.5.6. Let Δ be a maximally consistent set. To cut down on the number of cases, in this problem, assume that $(A \supset B)$ is an abbreviation for $(\neg A \vee B)$, so that the set of connectives is $\{\wedge, \vee, \neg\}$.

(a) Show that Δ is a Hintikka set.

(b) Recall that in LK', $\Gamma \rightarrow A$ is not provable if and only if $\Gamma \cup \{\neg A\}$ is consistent. Using problem 3.6.4, prove that if $\Gamma \rightarrow A$ is valid, then it is provable in LK' (with cut).

Remark: This provides another proof of the completeness of LK' (with cut). Note that a proof tree for $\Gamma \rightarrow A$ is not produced (compare with theorem 3.4.1).

3.6.6. Prove that the extended completeness theorem and the model existence theorem are equivalent for LK'. (This is also true for $LK' - \{cut\}$, but apparently requires the cut elimination theorem).

3.6.7. Implement a computer program (preferably in PASCAL or C) converting an LK'-proof into a cut-free LK'-proof. Compare and investigate the relative lengths of proofs.

Notes and Suggestions for Further Reading

We have chosen Gentzen systems as the main vehicle for presenting propositional logic because of their algorithmic nature and their conceptual simplicity. Our treatment is inspired from Kleene, 1967 and Kleene, 1952. For more on Gentzen systems, the reader should consult Takeuti, 1975; Szabo, 1969; or Smullyan, 1968.

We believe that the use of Hintikka sets improves the clarity of the proof of the completeness theorem. For more details on Hintikka sets and related concepts such as consistency properties, the reader is referred to Smullyan, 1968.

There are other proof systems for propositional logic. The Hilbert system H discussed in problems 3.4.9 to 3.4.12 is from Kleene, 1967, as well as the natural deduction system used in problems 3.4.11 and 3.4.12. For more on natural deduction systems, the reader is referred to Van Dalen, 1980; Prawitz 1965; or Szabo, 1969. A variant of Gentzen systems called tableaux systems is discussed at length in Smullyan, 1968.

The relationship between boolean algebra and logic was investigated by Tarski, Lindenbaum, Rasiowa, and Sikorski. For more details, the reader is referred to Chang and Keisler, 1973, or Bell and Slomson, 1974. Exercise 3.5.18 is adapted from Bell and Slomson, 1974.

The proof of Gentzen's cut elimination theorem can be found in Kleene, 1952; Takeuti, 1975; and Szabo, 1969.