Chapter 5

First-Order Logic

5.1 INTRODUCTION

In propositional logic, it is not possible to express assertions about elements of a structure. The weak expressive power of propositional logic accounts for its relative mathematical simplicity, but it is a very severe limitation, and it is desirable to have more expressive logics. First-order logic is a considerably richer logic than propositional logic, but yet enjoys many nice mathematical properties. In particular, there are finitary proof systems complete with respect to the semantics.

In first-order logic, assertions about elements of structures can be expressed. Technically, this is achieved by allowing the propositional symbols to have arguments ranging over elements of structures. For convenience, we also allow symbols denoting functions and constants.

Our study of first-order logic will parallel the study of propositional logic conducted in Chapter 3. First, the *syntax* of first-order logic will be defined. The syntax is given by an inductive definition. Next, the *semantics* of first-order logic will be given. For this, it will be necessary to define the notion of a *structure*, which is essentially the concept of an algebra defined in Section 2.4, and the notion of *satisfaction*. Given a structure $\mathbf{M}$ and a formula A, for any assignment s of values in $\mathbf{M}$ to the variables (in A), we shall define the *satisfaction relation* $\models$, so that

$$\mathbf{M} \models A[s]$$

expresses the fact that the assignment s satisfies the formula A in $\mathbf{M}$.

The satisfaction relation $\models$ is defined recursively on the set of formulae. Hence, it will be necessary to prove that the set of formulae is freely generated by the atomic formulae.

A formula A is said to be *valid in a structure* $\mathbf{M}$ if

$$\mathbf{M} \models A[s]$$

for every assignment s. A formula A is *valid* (or universally valid) if A is valid in every structure $\mathbf{M}$.

Next, we shall attempt to find an algorithm for deciding whether a formula is valid. Unfortunately, there is no such algorithm. However, it is possible to find a procedure that will construct a proof for a valid formula. Contrary to the propositional case, this procedure may run forever if the input formula is not valid. We will set up a *proof system* that is a generalization of the Gentzen system of Section 3.4 and extend the *search* procedure to first-order formulae. Then, some fundamental theorems will be proved, including the completeness theorem, compactness theorem, and model existence theorem.

The two main concepts in this chapter are the *search* procedure to build proof trees and Hintikka sets. The main theme is that the *search* procedure is a *Hintikka set constructor*. If the *search* procedure fails to find a counter example, a proof is produced. Otherwise, the Hintikka set yields a counter example. This approach yields the completeness theorem in a very direct fashion. Another theme that will emerge in this chapter is that the *search* procedure is a very inefficient proof procedure. The purpose of the next chapters is to try to find more efficient proof procedures.

5.2 FIRST-ORDER LANGUAGES

First, we define alphabets for first-order languages.

5.2.1 Syntax

In contrast with propositional logic, first-order languages have a fixed part consisting of logical connectives, variables, and auxiliary symbols, and a part that depends on the intended application of the language, consisting of predicate, function, and constant symbols, called the *non logical part*.

Definition 5.2.1 The *alphabet* of a first-order language consists of the following sets of symbols:

Logical connectives: $\wedge$ (and), $\vee$ (or), $\neg$ (not), $\supset$ (implication), $\equiv$ (equivalence), $\perp$ (falsehood); quantifiers: $\forall$ (for all), $\exists$ (there exists); the equality symbol $\doteq$.

Variables: A countably infinite set $\mathbf{V} = \{x_0, x_1, x_2, ...\}$.

Auxiliary symbols: "(" and ")".

A set $\mathbf{L}$ of *nonlogical symbols* consisting of:

(i) *Function symbols*: A (countable, possibly empty) set **FS** of symbols f_0,f_1,..., and a *rank function* r assigning a positive integer $r(f)$ (called *rank* or *arity*) to every function symbol f.

(ii) *Constants*: A (countable, possibly empty) set **CS** of symbols c_0,c_1,..., each of rank zero.

(iii) *Predicate symbols*: A (countable, possibly empty) set **PS** of symbols P_0, P_1,..., and a *rank function* r assigning a nonnegative integer $r(P)$ (called rank or arity) to each predicate symbol P.

It is assumed that the sets **V**, **FS**, **CS** and **PS** are disjoint. We will refer to a first-order language with set of nonlogical symbols **L** as the language **L**. First-order languages obtained by omitting the equality symbol are referred to as *first-order languages without equality*. Note that predicate symbols of rank zero are in fact propositional symbols. Note also that we are using the symbol $\doteq$ for equality in the object language in order to avoid confusion with the symbol $=$ used for equality in the meta language.

We now give inductive definitions for the sets of terms and formulae. Note that a first-order language is in fact a two-sorted ranked alphabet in the sense of Subsection 2.5.1. The sorts are *term* and *formula*. The symbols in **V**, **CS** and **FS** are of sort *term*, and the symbols in **PS** and $\perp$ are of sort *formula*.

Definition 5.2.2 Given a first-order language **L**, let Γ be the union of the sets **V**, **CS** and **FS**. For every function symbol f of rank $n > 0$, let C_f be the function $C_f : (\Gamma^*)^n \to \Gamma^*$ such that, for all strings $t_1, ..., t_n \in \Gamma^*$,

$$C_f(t_1, ..., t_n) = ft_1...t_n.$$

The set $TERM_{\mathbf{L}}$ of **L**-*terms* (for short, terms) is the inductive closure of the union of the set **V** of variables and the set **CS** of constants under the constructors C_f.

A more informal way of stating definition 5.2.2 is the following:

(i) Every constant and every variable is a term.

(ii) If $t_1, ..., t_n$ are terms and f is a function symbol of rank $n > 0$, then $ft_1...t_n$ is a term.

EXAMPLE 5.2.1

Let **L** be the following first-order language for arithmetic where, **CS** $= \{0\}$, **FS** $= \{S, +, *\}$, and **PS** $= \{<\}$. The symbol S has rank 1, and the symbols $+$, $*$ and $<$ have rank 2. Then, the following are terms:

$$S0$$

$$+S0SS0$$

$$*x_1S + Sx_1x_2.$$

Definition 5.2.3 Given a first-order language **L**, let Δ be the union of the sets Γ and **PS** (where Γ is the union of the sets **V**, **CS**, **FS**). For every predicate symbol P of rank $n > 0$, let C_P be the function

$$C_P : (\Gamma^*)^n \to \Delta^*$$

such that, for all strings $t_1, ..., t_n \in \Gamma^*$,

$$C_P(t_1, ..., t_n) = Pt_1...t_n.$$

Also, let $C_{\doteq}$ be the function $C_{\doteq} : (\Gamma^*)^2 \to \Delta^*$ such that, for all strings $t_1, t_2 \in \Gamma^*$,

$$C_{\doteq}(t_1, t_2) = \doteq t_1t_2.$$

The set of **L**-*atomic formulae* (for short, atomic formulae) is the inductive closure of the pair of sets $TERM_{\mathbf{L}}$ (of sort *term*) and $\{P \mid P \in \mathbf{PS}, r(P) = 0\} \cup \{\perp\}$ (of sort *formula*), under the functions C_P and $C_{\doteq}$.

A less formal definition is the following:

(i) Every predicate symbol of rank 0 is an atomic formula, and so is $\perp$.

(ii) If $t_1, ..., t_n$ are terms and P is a predicate symbol of rank $n > 0$, then $Pt_1...t_n$ is an atomic formula, and so is $\doteq t_1t_2$.

Let Σ be the union of the sets **V**, **CS**, **FS**, **PS**, and $\{\wedge, \vee, \neg, \supset, \equiv, \forall, \exists, \doteq, (,), \perp\}$. The functions $C_\wedge$, $C_\vee$, $C_\supset$, $C_\equiv$, $C_\neg$ are defined (on Σ^*) as in definition 3.2.2, and the functions A_i and E_i are defined such that, for any string $A \in \Sigma^*$,

$$A_i(A) = \forall x_iA, \text{ and } E_i(A) = \exists x_iA.$$

The set $FORM_{\mathbf{L}}$ of **L**-*formulae* (for short, formulae) is the inductive closure of the set of atomic formulae under the functions $C_\wedge$, $C_\vee$, $C_\supset$, $C_\equiv$, $C_\neg$ and the functions A_i and E_i.

A more informal way to state definition 5.2.3 is to define a formula as follows:

(i) Every atomic formula is a formula.

(ii) For any two formulae A and B, $(A \wedge B)$, $(A \vee B)$, $(A \supset B)$, $(A \equiv B)$ and $\neg A$ are also formulae.

(iii) For any variable x_i and any formula A, $\forall x_i A$ and $\exists x_i A$ are also formulae.

We let the letters x, y, z subscripted or not range over variables. We also omit parentheses whenever possible, as in the propositional calculus.

EXAMPLE 5.2.2

Using the first-order language of example 5.2.1, the following are atomic formulae:

$$< 0S0$$

$$\doteq ySx$$

The following are formulae:

$$\forall x \forall y (< xy \supset \exists z \doteq y + xz)$$

$$\forall x \forall y ((< xy \vee < yx) \vee \doteq xy)$$

Next, we will show that terms and formulae are freely generated.

5.2.2 Free Generation of the Set of Terms

We define a function K such that for a symbol s (variable, constant or function symbol), $K(s) = 1 - n$, where n is the least number of terms that must follow s to obtain a term (n is the "tail deficiency").

$$K(x) = 1 - 0 = 1, \text{ for a variable } x;$$
$$K(c) = 1 - 0 = 1, \text{ for a constant } c;$$
$$K(f) = 1 - n, \text{ for a } n\text{-ary function symbol } f.$$

We extend K to strings composed of variables, constants and function symbols as follows: if $w = w_1...w_m$ then

$$K(w) = K(w_1) + ... + K(w_m),$$

and the following property holds.

Lemma 5.2.1 For any term t, $K(t) = 1$.

Proof: We use the induction principle for the set of terms. The basis of the induction holds trivially for the atoms. For a term $ft_1...t_n$ where f is of arity $n > 0$ and $t_1, ..., t_n$ are terms, since by definition $K(ft_1...t_n) = K(f) +$

$K(t_1) + ... + K(t_n)$ and by the induction hypothesis $K(t_1) = ... = K(t_n) = 1$, we have $K(ft_1...t_n) = 1 - n + (1 + ... + 1) = 1$. $\square$

Lemma 5.2.2 Every nonempty suffix of a term is a concatenation of one or more terms.

Proof: We use the induction principle for terms. The basis is obvious for the atoms. For a term $ft_1...t_n$, any proper suffix w must be of the form $st_{k+1}...t_n$, where $k \leq n$, and s is a suffix of t_k. By the induction hypothesis, s is a concatenation of terms $s_1, ..., s_m$, and $w = s_1...s_m t_{k+1}...t_n$, which is a concatenation of terms. $\square$

Lemma 5.2.3 No proper prefix of a term is a term.

Proof: Assume a term t is divided into a proper prefix t_1 and a (proper) suffix t_2. Then, $1 = K(t) = K(t_1) + K(t_2)$. By lemma 5.2.2, $K(t_2) \geq 1$. Hence $K(t_1) \leq 0$ and t_1 cannot be a term by lemma 5.2.1. $\square$

Theorem 5.2.1 The set of **L**-terms is freely generated from the variables and constants as atoms and the functions C_f as operations.

Proof: First, $f \neq g$ clearly implies that C_f and C_g have disjoint ranges and these ranges are disjoint from the set of variables and the set of constants. Since the constructors increase the length of strings, condition (3) for free generation holds. It remains to show that the restrictions of the functions C_f to $TERM_{\mathbf{L}}$ are injective. If $ft_1...t_m = fs_1...s_n$, we must have $t_1...t_m = s_1...s_n$. Then either $t_1 = s_1$, or t_1 is a proper prefix of s_1, or s_1 is a proper prefix of t_1. In the first case, $s_2...s_m = t_2...t_n$. The other two cases are ruled out by lemma 5.2.3 since both t_1 and s_1 are terms. By repeating the above reasoning, we find that $m = n$ and $t_i = s_i$, $1 \leq i \leq n$. Hence the set of terms is freely generated. $\square$

5.2.3 Free Generation of the Set of Formulae

To extend this argument to formulae, we define K on the other symbols with the following idea in mind: $K(s)$ should be $1 - n$, where n is the least number of things (right parentheses, terms or formulae) required to go along with s in order to form a formula (n is the "tail deficiency").

$$\begin{aligned}
K(\text{“(”}) &= -1;\\
K(\text{“)”}) &= 1;\\
K(\forall) &= -1;\\
K(\exists) &= -1;\\
K(\wedge) &= -1;\\
K(\vee) &= -1;\\
K(\supset) &= -1;
\end{aligned}$$

$K(\equiv) = -1;$
$K(\neg) = 0;$
$K(\doteq) = -1;$
$K(P) = 1 - n$ for any n-ary predicate symbol P;
$K(\perp) = 1.$

We also extend K to strings as usual:

$$K(w_1...w_m) = K(w_1) + ... + K(w_m).$$

A lemma analogous to lemma 5.2.1 holds for formulae.

Lemma 5.2.4 For any formula A, $K(A) = 1$.

Proof: The proof uses the induction principle for formulae. Let Y be the subset of the set of formulae A such that $K(A) = 1$. First, we show that Y contains the atomic formulae. If A is of the form $Pt_1...t_m$ where P has rank m, since by lemma 5.2.1, $K(t_i) = 1$, and by definition $K(P) = 1-m$, $K(A) = 1-m+m = 1$, as desired. If A is of the form $\doteq t_1t_2$, since $K(t_1) = K(t_2) = 1$ and $K(\doteq) = -1$, $K(A) = -1+1+1 = 1$. By definition $K(\perp) = 1$. Next, if A is of the form $\neg B$, by the induction hypothesis, $K(B) = 1$. Since $K(\neg) = 0$, $K(A) = 0+1 = 1$. If A is of the form $(B * C)$ where $* \in \{\wedge, \vee, \supset, \equiv\}$, by the induction hypothesis, $K(B) = 1$ and $K(C) = 1$, and

$$K(A) = K(\text{“(”}) + K(B) + K(*) + K(C) + K(\text{“)”}) = -1+1+-1+1+1 = 1.$$

Finally if A is of the form $\forall x_i B$ (or $\exists x_i B$), by the induction hypothesis $K(B) = 1$, and $K(A) = K(\forall) + K(x_i) + K(B) = -1+1+1 = 1$. Hence Y is closed under the connectives, which concludes the proof by induction. □

Lemma 5.2.5 For any proper prefix w of a formula A, $K(w) \leq 0$.

Proof: The proof uses the induction principle for formulae and is similar to that of lemma 5.2.4. It is left as an exercise. □

Lemma 5.2.6 No proper prefix of a formula is a formula.

Proof: Immediate from lemma 5.2.4 and lemma 5.2.5. □

Theorem 5.2.2 The set of **L**-formulae is freely generated by the atomic formulae as atoms and the functions C_X (X a logical connective), A_i and E_i.

Proof: The proof is similar to that of theorem 3.2.1 and rests on the fact that no proper prefix of a formula is a formula. The argument for atomic formulae is similar to the arguments for terms. The propositional connectives are handled as in the proof of theorem 3.2.1. For the quantifiers, observe that given two formulae, one in the range of a function A_i or E_i, the other in the range of another function, either the leftmost characters differ (for $C_\wedge$, $C_\vee$, $C_\supset$, $C_\equiv$ and $C_\neg$, "(" and "$\neg$" are different from "$\forall$" and "$\exists$"), or the substrings consisting of the two leftmost symbols differ ($\forall x_i$ is different from $\forall x_j$ for $j \neq i$ and different from $\exists x_j$ for any x_j, and similarly $\exists x_i$ is different from $\exists x_j$ for $x_j \neq x_i$ and different from $\forall x_j$ for any x_j).

The fact that the restrictions of the functions $C_\wedge$, $C_\vee$, $C_\supset$, $C_\equiv$ and $C_\neg$ to $FORM_\mathbf{L}$ are injective is shown as in lemma 3.2.1, using the property that a proper prefix of a formula is not a formula (and a proper prefix of a term is not a term). For the functions A_i, $A_i(A) = A_i(B)$ iff $\forall x_i A = \forall x_i B$ iff $A = B$, and similarly for E_i. Finally, the constructors increase the number of connectives (or quantifiers). □

Remarks:

(1) Instead of defining terms and atomic formulae in prefix notation, one can define them as follows (using parentheses):

The second clause of definition 5.2.2 is changed to: For every function symbol f of arity n and any terms $t_1,...,t_n$, $f(t_1,...,t_n)$ is a term. Also, atomic formulae are defined as follows: For every predicate symbol P of arity n and any terms $t_1,...,t_n$, $P(t_1,...,t_n)$ is an atomic formula; $t_1 \doteq t_2$ is an atomic formula.

One can still show that the terms and formulae are freely generated. In the sequel, we shall use the second notation when it is convenient. For simplicity, we shall also frequently use $=$ instead of $\doteq$ and omit parentheses whenever possible, using the conventions adopted for the propositional calculus.

(2) The sets $TERM_\mathbf{L}$ and $FORM_\mathbf{L}$ are the carriers of a two-sorted algebra $T(\mathbf{L}, \mathbf{V})$ with sorts *term* and *formula*, in the sense of Subsection 2.5.2. The operations are the functions C_f for all function symbols, $C_{\doteq}$, C_P for all predicate symbols, $C_\wedge$, $C_\vee$, $C_\supset$, $C_\equiv$, $C_\neg$, A_i, E_i ($i \geq 0$) for formulae, and the symbols of arity 0 are the propositional symbols in **PS**, the constants in **CS**, and $\perp$. This two-sorted algebra $T(\mathbf{L}, \mathbf{V})$ is free on the set of variables **V** (as defined in Section 2.5), and is isomorphic to the tree algebra $T_\mathbf{L}(\mathbf{V})$ (in $T_\mathbf{L}(\mathbf{V})$, the term $A_i(A)$ is used instead of $\forall x_i A$, and $E_i(A)$ instead of $\exists x_i A$).

5.2.4 Free and Bound Variables

In first-order logic, variables may occur *bound* by quantifiers. Free and bound occurrences of variables are defined by recursion as follows.

Definition 5.2.4 Given a term t, the set $FV(t)$ of *free variables* of t is defined by recursion as follows:

$$FV(x_i) = \{x_i\}, \text{ for a variable } x_i;$$
$$FV(c) = \emptyset, \text{ for a constant } c;$$

$$FV(ft_1...t_n) = FV(t_1) \cup ... \cup FV(t_n),$$
$$\text{for a function symbol } f \text{ of rank } n.$$

For a formula A, the set $FV(A)$ of *free variables* of A is defined by:

$$\begin{aligned}
FV(Pt_1...t_n) &= FV(t_1) \cup ... \cup FV(t_n),\\
&\qquad \text{for a predicate symbol } P \text{ of rank } n;\\
FV(\doteq t_1t_2) &= FV(t_1) \cup FV(t_2);\\
FV(\neg A) &= FV(A);\\
FV((A * B)) &= FV(A) \cup FV(B), \text{ where } * \in \{\wedge, \vee, \supset, \equiv\};\\
FV(\perp) &= \emptyset;\\
FV(\forall x_i A) &= FV(A) - \{x_i\};\\
FV(\exists x_i A) &= FV(A) - \{x_i\}.
\end{aligned}$$

A term t or a formula A is *closed* if, respectively $FV(t) = \emptyset$, or $FV(A) = \emptyset$. A closed formula is also called a *sentence*. A formula without quantifiers is called *open*.

Definition 5.2.5 Given a formula A, the set $BV(A)$ of *bound variables* in A is given by:

$$\begin{aligned}
BV(Pt_1...t_n) &= \emptyset;\\
BV(\doteq t_1t_2) &= \emptyset;\\
BV(\neg A) &= BV(A);\\
BV((A * B)) &= BV(A) \cup BV(B), \text{ where } * \in \{\wedge, \vee, \supset, \equiv\};\\
BV(\perp) &= \emptyset;\\
BV(\forall x_i A) &= BV(A) \cup \{x_i\};\\
BV(\exists x_i A) &= BV(A) \cup \{x_i\}.
\end{aligned}$$

In a formula $\forall x_i A$ (or $\exists x_i A$), we say that the variable x_i *is bound by the quantifier* $\forall$ (or $\exists$).

For a formula A, the intersection of $FV(A)$ and $BV(A)$ need not be empty, that is, the same variable may have both a free and a bound occurrence in A. As we shall see later, every formula is equivalent to a formula where the free and bound variables are disjoint.

EXAMPLE 5.2.3

Let

$$A = (\forall x(Rxy \supset Px) \wedge \forall y(\neg Rxy \wedge \forall x Px)).$$

Then

$$FV(A) = \{x, y\},\ BV(A) = \{x, y\}.$$

For

$$B = \forall x(Rxy \supset Px),$$

we have

$$FV(B) = \{y\},\ BV(B) = \{x\}.$$

For

$$C = \forall y(\neg Rxy \land \forall x Px),$$

we have

$$FV(C) = \{x\} \text{ and } BV(C) = \{x, y\}.$$

5.2.5 Substitutions

We will also need to define the substitution of a term for a free variable in a term or a formula.

Definition 5.2.6 Let s and t be terms. The result of substituting t in s for a variable x, denoted by $s[t/x]$ is defined recursively as follows:

$$\begin{aligned} y[t/x] &= \ if\ y \neq x\ then\ y\ else\ t, \text{ when } s \text{ is a variable } y; \\ c[t/x] &= c, \text{ when } s \text{ is a constant } c; \\ ft_1...t_n[t/x] &= ft_1[t/x]...t_n[t/x], \text{ when } s \text{ is a term } ft_1...t_n. \end{aligned}$$

For a formula A, $A[t/x]$ is defined recursively as follows:

$$\begin{aligned} \perp [t/x] &= \perp, \text{ when } A \text{ is } \perp; \\ Pt_1...t_n[t/x] &= Pt_1[t/x]...t_n[t/x], \text{ when } A = Pt_1...t_n; \\ \doteq t_1t_2[t/x] &= \doteq t_1[t/x]t_2[t/x], \text{ when } A = \doteq t_1t_2; \\ (B * C)[t/x] &= (B[t/x] * C[t/x]), \text{ when } A = (B * C),\ * \in \{\land, \lor, \supset, \equiv\}. \\ (\neg B)[t/x] &= \neg B[t/x], \text{ when } A = \neg B; \\ (\forall yB)[t/x] &= \ if\ x \neq y\ then\ \forall yB[t/x]\ else\ \forall yB, \text{ when } A = \forall yB; \\ (\exists yB)[t/x] &= \ if\ x \neq y\ then\ \exists yB[t/x]\ else\ \exists yB, \text{ when } A = \exists yB. \end{aligned}$$

EXAMPLE 5.2.4

Let

$$A = \forall x(P(x) \supset Q(x, f(y))), \text{ and let } t = g(y).$$

We have

$$f(x)[t/x] = f(g(y)), \quad A[t/y] = \forall x(P(x) \supset Q(x, f(g(y)))),$$

$$A[t/x] = \forall x(P(x) \supset Q(x, f(y))) = A,$$

since x is a bound variable.

The above definition prevents substitution of a term for a bound variable. When we give the semantics of the language, certain substitutions will not behave properly in the sense that they can change the truth value in a wrong way. These are substitutions in which some variable in the term t becomes bound in $A[t/x]$.

EXAMPLE 5.2.5

Let

$$A = \exists x(x < y)[x/y] = \exists x(x < x).$$

The sentence

$$\exists x(x < x)$$

is false in an ordered structure, but

$$\exists x(x < y)$$

may well be satisfied.

Definition 5.2.7 A term t is *free for x in A* if either:

(i) A is atomic or

(ii) $A = (B * C)$ and t is free for x in B and C, $* \in \{\wedge, \vee, \supset, \equiv\}$ or

(iii) $A = \neg B$ and t is free for x in B or

(iv) $A = \forall yB$ or $A = \exists yB$ and either

$x = y$ or

$x \neq y$, $y \notin FV(t)$ and t is free for x in B.

EXAMPLE 5.2.6

Let

$$A = \forall x(P(x) \supset Q(x, f(y))).$$

Then $g(y)$ is free for y in A, but $g(x)$ is not free for y in A.
If

$$B = \forall x(P(x) \supset \forall z Q(z, f(y))),$$

then $g(z)$ is not free for y in B, but $g(z)$ is free for z in B (because the substitution of $g(z)$ for z will not take place).

From now on we will assume that all substitutions satisfy the conditions of definition 5.2.7. Sometimes, if a formula A contains x as a free variable we write A as $A(x)$ and we abbreviate $A[t/x]$ as $A(t)$. In the next section, we will turn to the semantics of first-order logic.

PROBLEMS

5.2.1. Prove lemma 5.2.6.

5.2.2. Let w be a string consisting of variables, constants and function symbols.

(a) Prove that if $K(v) > 0$ for every proper suffix v of w, then w is a concatenation of $K(w)$ terms.

(b) Prove that w is a term iff $K(w) = 1$ and $K(v) > 0$ for every proper suffix v of w.

5.2.3. Given a formula A and constants a and b, show that for any two distinct variables x_i and x_j,

$$A[a/x_i][b/x_j] = A[b/x_j][a/x_i].$$

5.2.4. Prove that for every formula A and for every constant c,

$$FV(A[c/x]) = FV(A) - \{x\}.$$

5.2.5. Prove that for any formula A and term t, if y is not in $FV(A)$, then $A[t/y] = A$.

5.2.6. Prove that for every formula A and every term t, if t is free for x in A, $x \neq z$ and z is not in $FV(t)$, for every constant d,

$$A[t/x][d/z] = A[d/z][t/x].$$

5.2.7. Prove that for every formula A, if x and y are distinct variables, y is free in A, and z is a variable not occurring in A, for every term t free for x in A,

$$A[t/x][z/y] = A[z/y][t[z/y]/x],$$

and $t[z/y]$ is free for x in $A[z/y]$.

$*$ **5.2.8.** The purpose of this problem is to generalize problem 3.2.6, that is, to give a context-free grammar defining terms and formulae. For this, it is necessary to encode the variables, constants, and the function and predicate symbols, as strings over a finite alphabet. Following Lewis and Papadimitriou, 1981, each variable x_n is encoded as $xI^n\$$, each m-ary predicate symbol P_n is encoded as $PI^m\$I^n\$$ $(m, n \geq 0)$, and each m-ary function symbol f_n is encoded as $fI^m\$I^n\$$.

Then, the set of terms and formulae is the language $L(G)$ defined by the following context-free grammar $G = (V, \Sigma, R, S)$:

$\Sigma = \{P, I, f, x, \$, \wedge, \vee, \supset, \equiv, \neg, \forall, \exists, \doteq, \perp\}$,

$V = \Sigma \cup \{S, N, T, U, W\}$,

$$\begin{aligned} R = \{&N \rightarrow e,\\ &N \rightarrow NI,\\ &W \rightarrow xN\$,\\ &T \rightarrow W,\\ &T \rightarrow fU,\\ &U \rightarrow IUT,\\ &U \rightarrow \$N\$,\\ &S \rightarrow PU,\\ &S \rightarrow \perp,\\ &S \rightarrow \doteq TT,\\ &S \rightarrow (S \vee S),\\ &S \rightarrow (S \wedge S),\\ &S \rightarrow (S \supset S),\\ &S \rightarrow (S \equiv S),\\ &S \rightarrow \neg S,\\ &S \rightarrow \forall WS,\\ &S \rightarrow \exists WS\} \end{aligned}$$

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.

5.3 SEMANTICS OF FIRST-ORDER LANGUAGES

Given a first-order language **L**, the semantics of formulae is obtained by interpreting the function, constant and predicate symbols in **L** and assigning values to the free variables. For this, we need to define the concept of a *structure*.

5.3.1 First-Order Structures

A first-order structure assigns a meaning to the symbols in **L** as explained below.

Definition 5.3.1 Given a first-order language **L**, an **L**-*structure* **M** (for short, a structure) is a pair $\mathbf{M} = (M, I)$ where M is a nonempty set called the *domain* (or *carrier*) of the structure and I is a function called the *interpretation function* and which assigns functions and predicates over M to the symbols in **L** as follows:

(i) For every function symbol f of rank $n > 0$, $I(f) : M^n \to M$ is an n-ary function.

(ii) For every constant c, $I(c)$ is an element of M.

(iii) For every predicate symbol P of rank $n \geq 0$, $I(P) : M^n \to BOOL$ is an n-ary predicate. In particular, predicate symbols of rank 0 are interpreted as truth values (Hence, the interpretation function I restricted to predicate symbols of rank zero is a valuation as in propositional logic.)

We will also use the following notation in which I is omitted: $I(f)$ is denoted as $f_{\mathbf{M}}$, $I(c)$ as $c_{\mathbf{M}}$ and $I(P)$ as $P_{\mathbf{M}}$.

EXAMPLE 5.3.1

Let $\mathbf{L}$ be the language of arithmetic where, $\mathbf{CS} = \{0\}$ and $\mathbf{FS} = \{S, +, *\}$. The symbol S has rank 1, and the symbols $+$, $*$ have rank 2. The $\mathbf{L}$-structure $\mathcal{N}$ is defined such that its domain is the set $\mathbf{N}$ of natural numbers, and the constant and function symbols are interpreted as follows: 0 is interpreted as zero, S as the function such that $S(x) = x+1$ for all $x \in \mathbf{N}$ (the successor function), $+$ is interpreted as addition and $*$ as multiplication. The structure $\mathcal{Q}$ obtained by replacing $\mathbf{N}$ by the set $\mathbf{Q}$ of rational numbers and the structure $\mathcal{R}$ obtained by replacing $\mathbf{N}$ by the set $\mathbf{R}$ of real numbers and interpreting 0, S, $+$ and $*$ in the natural way are also $\mathbf{L}$-structures.

Remark: Note that a first-order $\mathbf{L}$-structure is in fact a two-sorted algebra as defined in Subsection 2.5.2 (with carrier $BOOL$ of sort *formula* and carrier M of sort *term*).

5.3.2 Semantics of Formulae

We now wish to define the semantics of formulae. Since a formula A may contain free variables, its truth value will generally depend on the specific assignment s of values from the domain M to the variables. Hence, we shall define the semantics of a formula A as a *function* $A_{\mathbf{M}}$ from the set of assignments of values in M to the variables, to the set $BOOL$ of truth values. First, we need some definitions. For more details, see Section 10.3.2.

Definition 5.3.2 Given a first-order language $\mathbf{L}$ and an $\mathbf{L}$-structure $\mathbf{M}$, an *assignment* is any function $s : \mathbf{V} \to M$ from the set of variables $\mathbf{V}$ to the domain M. The set of all such functions is denoted by $[\mathbf{V} \to M]$.

To define the meaning of a formula, we shall construct the function $A_{\mathbf{M}}$ recursively, using the fact that formulae are freely generated from the atomic formulae, and using theorem 2.4.1. Since the meaning of a formula is a function from $[\mathbf{V} \to M]$ to $BOOL$, let us denote the set of all such functions as $[[\mathbf{V} \to M] \to BOOL]$. In order to apply theorem 2.4.1, it is necessary to extend the connectives to the set of functions $[[\mathbf{V} \to M] \to BOOL]$ to make this set into an algebra. For this, the next two definitions are needed.

Definition 5.3.3 Given any nonempty domain M, given any element $a \in M$ and any assignment $s : \mathbf{V} \to M$, $s[x_i := a]$ denotes the new assignment $s' : \mathbf{V} \to M$ such that

$$s'(y) = s(y) \text{ for } y \neq x_i \text{ and } s'(x_i) = a.$$

For all $i \geq 0$, define the function $(A_i)_\mathbf{M}$ and $(E_i)_\mathbf{M}$ from $[[\mathbf{V} \to M] \to BOOL]$ to $[[\mathbf{V} \to M] \to BOOL]$ as follows: For every function $f \in [[\mathbf{V} \to M] \to BOOL]$, $(A_i)_\mathbf{M}(f)$ is the function such that: For every assignment $s \in [\mathbf{V} \to M]$,

$$(A_i)_\mathbf{M}(f)(s) = \mathbf{F} \quad \text{iff} \quad f(s[x_i := a]) = \mathbf{F} \quad \text{for some } a \in M;$$

The function $(E_i)_\mathbf{M}(f)$ is the function such that: For every assignment $s \in [\mathbf{V} \to M]$,

$$(E_i)_\mathbf{M}(f)(s) = \mathbf{T} \quad \text{iff} \quad f(s[x_i := a]) = \mathbf{T} \quad \text{for some } a \in M.$$

Note that $(A_i)_\mathbf{M}(f)(s) = \mathbf{T}$ iff the function $g_s : M \to BOOL$ such that $g_s(a) = f(s[x_i := a])$ for all $a \in M$ is the constant function whose value is $\mathbf{T}$, and that $(E_i)_\mathbf{M}(f)(s) = \mathbf{F}$ iff the function $g_s : M \to BOOL$ defined above is the constant function whose value is $\mathbf{F}$.

Definition 5.3.4 Given a nonempty domain M, the functions $\wedge_\mathbf{M}$, $\vee_\mathbf{M}$, $\supset_\mathbf{M}$ and $\equiv_\mathbf{M}$ from $[[\mathbf{V} \to M] \to BOOL] \times [[\mathbf{V} \to M] \to BOOL]$ to $[[\mathbf{V} \to M] \to BOOL]$, and the function $\neg_\mathbf{M}$ from $[[\mathbf{V} \to M] \to BOOL]$ to $[[\mathbf{V} \to M] \to BOOL]$ are defined as follows: For every two functions f and g in $[[\mathbf{V} \to M] \to BOOL]$, for all s in $[\mathbf{V} \to M]$,

$$\begin{aligned}
\wedge_\mathbf{M}(f,g)(s) &= H_\wedge(f(s), g(s));\\
\vee_\mathbf{M}(f,g)(s) &= H_\vee(f(s), g(s));\\
\supset_\mathbf{M} (f,g)(s) &= H_\supset(f(s), g(s));\\
\equiv_\mathbf{M} (f,g)(s) &= H_\equiv(f(s), g(s));\\
\neg_\mathbf{M}(f)(s) &= H_\neg(f(s)).
\end{aligned}$$

Using theorem 2.4.1, we can now define the meaning $t_\mathbf{M}$ of a term t and the meaning $A_\mathbf{M}$ of a formula A. We begin with terms.

Definition 5.3.5 Given an $\mathbf{L}$-structure $\mathbf{M}$, the function

$$t_\mathbf{M} : [\mathbf{V} \to M] \to M$$

defined by a term t is the function such that for every assignment $s \in [\mathbf{V} \to M]$, the value $t_\mathbf{M}[s]$ is defined recursively as follows:

(*i*) $x_{\mathbf{M}}[s] = s(x)$, for a variable x;

(*ii*) $c_{\mathbf{M}}[s] = c_{\mathbf{M}}$, for a constant c;

(*iii*) $(ft_1...t_n)_{\mathbf{M}}[s] = f_{\mathbf{M}}((t_1)_{\mathbf{M}}[s], ..., (t_n)_{\mathbf{M}}[s])$.

The recursive definition of the function $A_{\mathbf{M}} : [\mathbf{V} \to M] \to BOOL$ is now given.

Definition 5.3.6 The function

$$A_{\mathbf{M}} : [\mathbf{V} \to M] \to BOOL$$

is defined recursively by the following clauses:

(1) For atomic formulae: $A_{\mathbf{M}}$ is the function such that, for every assignment $s \in [\mathbf{V} \to M]$,

(*i*) $(Pt_1...t_n)_{\mathbf{M}}[s] = P_{\mathbf{M}}((t_1)_{\mathbf{M}}[s], ..., (t_n)_{\mathbf{M}}[s])$;

(*ii*) $(\doteq t_1t_2)_{\mathbf{M}}[s] = \ if\ (t_1)_{\mathbf{M}}[s] = (t_2)_{\mathbf{M}}[s]\ then\ \mathbf{T}\ else\ \mathbf{F}$;

(*iii*) $(\perp)_{\mathbf{M}}[s] = \mathbf{F}$.

(2) For nonatomic formulae:

(*i*) $$(A * B)_{\mathbf{M}} = *_{\mathbf{M}}(A_{\mathbf{M}}, B_{\mathbf{M}}),$$

where $* \in \{\wedge, \vee, \supset, \equiv\}$ and $*_{\mathbf{M}}$ is the corresponding function defined in definition 5.3.4;

(*ii*) $(\neg A)_{\mathbf{M}} = \neg_{\mathbf{M}}(A_{\mathbf{M}})$;

(*iii*) $(\forall x_i A)_{\mathbf{M}} = (A_i)_{\mathbf{M}}(A_{\mathbf{M}})$;

(*iv*) $(\exists x_i A)_{\mathbf{M}} = (E_i)_{\mathbf{M}}(A_{\mathbf{M}})$.

Note that by definitions 5.3.3, 5.3.4, 5.3.5, and 5.3.6, for every assignment $s \in [\mathbf{V} \to M]$,

$$(\forall x_i A)_{\mathbf{M}}[s] = \mathbf{T} \text{ iff } A_{\mathbf{M}}[s[x_i := m]] = \mathbf{T} \text{ for all } m \in M,$$

and

$$(\exists x_i A)_{\mathbf{M}}[s] = \mathbf{T} \text{ iff } A_{\mathbf{M}}[s[x_i := m]] = \mathbf{T} \text{ for some } m \in M.$$

Hence, if M is infinite, evaluating $(\forall x_i A)_{\mathbf{M}}[s]$ (or $(\exists x_i A)_{\mathbf{M}}[s]$) requires testing infinitely many values (all the truth values $A_{\mathbf{M}}[s[x_i := m]]$, for $m \in$

M). Hence, contrary to the propositional calculus, there does not appear to be an algorithm for computing the truth value of $(\forall x_i A)_{\mathbf{M}}[s]$ (or $(\exists x_i A)_{\mathbf{M}}[s]$) and in fact, no such algorithm exists in general.

5.3.3 Satisfaction, Validity, and Model

We can now define the notions of satisfaction, validity and model.

Definition 5.3.7 Let $\mathbf{L}$ be a first-order language and $\mathbf{M}$ be an $\mathbf{L}$-structure.

(i) Given a formula A and an assignment s, we say that $\mathbf{M}$ *satisfies* A *with* s iff

$$A_{\mathbf{M}}[s] = \mathbf{T}.$$

This is also denoted by

$$\mathbf{M} \models A[s].$$

(ii) A formula A is *satisfiable in* $\mathbf{M}$ iff there is some assignment s such that

$$A_{\mathbf{M}}[s] = \mathbf{T};$$

A is *satisfiable* iff there is some $\mathbf{M}$ in which A is satisfiable.

(iii) A formula A is *valid in* $\mathbf{M}$ (or *true* in $\mathbf{M}$) iff

$$A_{\mathbf{M}}[s] = \mathbf{T} \quad \text{for every assignment } s.$$

This is denoted by

$$\mathbf{M} \models A.$$

In this case, $\mathbf{M}$ is called a *model* of A. A formula A is *valid* (or *universally valid*) iff it is valid in every structure $\mathbf{M}$. This is denoted by

$$\models A.$$

(iv) Given a set Γ of formulae, Γ is *satisfiable* iff there exists a structure $\mathbf{M}$ and an assignment s such that

$$\mathbf{M} \models A[s] \quad \text{for every formula} \quad A \in \Gamma;$$

A structure $\mathbf{M}$ is a *model* of Γ iff $\mathbf{M}$ is a model of every formula in Γ. This is denoted by

$$\mathbf{M} \models \Gamma.$$

The set Γ is *valid* iff $\mathbf{M} \models \Gamma$ for every structure $\mathbf{M}$. This is denoted by

$$\models \Gamma.$$

(v) Given a set Γ of formulae and a formula B, B is a *semantic consequence* of Γ, denoted by

$$\Gamma \models B$$

iff, for every **L**-structure **M**, for every assignment s,

$$\text{if}\quad \mathbf{M} \models A[s] \quad \text{for every formula} \quad A \in \Gamma \quad \text{then} \quad \mathbf{M} \models B[s].$$

EXAMPLE 5.3.2

Consider the language of arithmetic defined in example 5.3.1. The following formulae are valid in the model $\mathcal{N}$ with domain the set of natural numbers defined in example 5.3.1. This set A_P is known as the axioms of *Peano's arithmetic*:

$$\forall x \neg (S(x) \doteq 0)$$
$$\forall x \forall y (S(x) \doteq S(y) \supset x \doteq y)$$
$$\forall x (x + 0 \doteq x)$$
$$\forall x \forall y (x + S(y) \doteq S(x + y))$$
$$\forall x (x * 0 \doteq 0)$$
$$\forall x \forall y (x * S(y) \doteq x * y + x)$$

For every formula A with one free variable x,

$$(A(0) \wedge \forall x (A(x) \supset A(S(x)))) \supset \forall y A(y)$$

This last axiom scheme is known as an *induction axiom*. The structure $\mathcal{N}$ is a model of these formulae. These formulae are not valid in all structures. For example, the first formula is not valid in the structure whose domain has a single element.

As in propositional logic, formulae that are universally valid are particularly interesting. These formulae will be characterized in terms of a proof system in the next section.

5.3.4 A More Convenient Semantics

It will be convenient in the sequel, especially in proofs involving Hintikka sets, to have a slightly different definition of the truth of a quantified formula. The idea behind this definition is to augment the language **L** with a set of constants naming the elements in the structure **M**, so that elements in M can be treated as constants in formulae. Instead of defining the truth value of $(\forall x_i A)_{\mathbf{M}}[s]$ using the modified assignments $s[x_i := m]$ so that

$$(\forall x_i A)_{\mathbf{M}}[s] = \mathbf{T} \text{ iff } A_{\mathbf{M}}[s[x_i := m]] = \mathbf{T} \text{ for all } m \in M,$$

we will define it using the modified formula $A[\mathbf{m}/x_i]$, so that

$$(\forall x_i A)_{\mathbf{M}}[s] = \mathbf{T} \text{ iff } A[\mathbf{m}/x_i]_{\mathbf{M}}[s] = \mathbf{T} \text{ for all } m \in M,$$

where $\mathbf{m}$ is a constant naming the element $m \in M$. Instead of computing the truth value of A in the modified assignment $s[x_i := m]$, we compute the truth value of the modified formula $A[\mathbf{m}/x_i]$ obtained by substituting the constant $\mathbf{m}$ for the variable x_i, in the assignment s itself.

Definition 5.3.8 Given a first-order language $\mathbf{L}$ and an $\mathbf{L}$-structure $\mathbf{M}$, the *extended language* $\mathbf{L}(\mathbf{M})$ is obtained by adjoining to the set $\mathbf{CS}$ of constants in $\mathbf{L}$ a set

$$\{\mathbf{m} \mid m \in M\}$$

of new constants, one for each element of M. The interpretation function I of the first-order structure $\mathbf{M}$ is extended to the constants in the set $\{\mathbf{m} \mid m \in M\}$ by defining $I(\mathbf{m}) = m$. Hence, for any assignment $s : \mathbf{V} \to M$,

$$(\mathbf{m})_{\mathbf{M}}[s] = m.$$

The resulting alphabet is not necessarily countable and the set of formulae may also not be countable. However, since a formula contains only finitely many symbols, there is no problem with inductive definitions or proofs by induction.

Next, we will prove a lemma that shows the equivalence of definition 5.3.6 and the more convenient definition of the truth of a quantified formula sketched before definition 5.3.8. First, we need the following technical lemma.

Lemma 5.3.1 Given a first-order language $\mathbf{L}$ and an $\mathbf{L}$-structure $\mathbf{M}$, the following hold:

(1) For any term t, for any assignment $s \in [\mathbf{V} \to M]$, any element $m \in M$ and any variable x_i,

$$t_{\mathbf{M}}[s[x_i := m]] = (t[\mathbf{m}/x_i])_{\mathbf{M}}[s].$$

(2) For any formula A, for any assignment $s \in [\mathbf{V} \to M]$, any element $m \in M$ and any variable x_i,

$$A_{\mathbf{M}}[s[x_i := m]] = (A[\mathbf{m}/x_i])_{\mathbf{M}}[s].$$

Proof: This proof is a typical induction on the structure of terms and formulae. Since we haven't given a proof of this kind for first-order formulae, we will give it in full. This will allow us to omit similar proofs later. First, we prove (1) by induction on terms.

If t is a variable $x_j \neq x_i$, then

$$(x_j)_{\mathbf{M}}[s[x_i := m]] = s(x_j)$$

and

$$(x_j[\mathbf{m}/x_i])_{\mathbf{M}}[s] = (x_j)_{\mathbf{M}}[s] = s(x_j),$$

establishing (1).

If t is the variable x_i, then

$$(x_i)_{\mathbf{M}}[s[x_i := m]] = m$$

and

$$(x_i[\mathbf{m}/x_i])_{\mathbf{M}}[s] = (\mathbf{m})_{\mathbf{M}}[s] = m,$$

establishing (1).

If t is a constant c, then

$$(c)_{\mathbf{M}}[s[x_i := m]] = c_{\mathbf{M}} = c_{\mathbf{M}}[s] = (c[\mathbf{m}/x_i])_{\mathbf{M}}[s],$$

establishing (1).

If t is a term $ft_1...t_k$, then

$$(ft_1...t_k)_{\mathbf{M}}[s[x_i := m]] = f_{\mathbf{M}}((t_1)_{\mathbf{M}}[s[x_i := m]], ..., (t_k)_{\mathbf{M}}[s[x_i := m]]).$$

By the induction hypothesis, for every j, $1 \leq j \leq k$,

$$(t_j)_{\mathbf{M}}[s[x_i := m]] = (t_j[\mathbf{m}/x_i])_{\mathbf{M}}[s].$$

Hence,

$$\begin{aligned}(ft_1...t_k)_{\mathbf{M}}[s[x_i := m]] &= f_{\mathbf{M}}((t_1)_{\mathbf{M}}[s[x_i := m]], ..., (t_k)_{\mathbf{M}}[s[x_i := m]])\\ &= f_{\mathbf{M}}((t_1[\mathbf{m}/x_i])_{\mathbf{M}}[s], ..., (t_k[\mathbf{m}/x_i])_{\mathbf{M}}[s])\\ &= (ft_1[\mathbf{m}/x_i]...t_k[\mathbf{m}/x_i])_{\mathbf{M}}[s]\\ &= (t[\mathbf{m}/x_i])_{\mathbf{M}}[s],\end{aligned}$$

establishing (1).

This concludes the proof of (1). Next, we prove (2) by induction on formulae.

If A is an atomic formula of the form $Pt_1...t_k$, then

$$(Pt_1...t_k)_{\mathbf{M}}[s[x_i := m]] = P_{\mathbf{M}}((t_1)_{\mathbf{M}}[s[x_i := m]], ..., (t_k)_{\mathbf{M}}[s[x_i := m]]).$$

By part (1) of the lemma, for every j, $1 \leq j \leq k$,

$$(t_j)_{\mathbf{M}}[s[x_i := m]] = (t_j[\mathbf{m}/x_i])_{\mathbf{M}}[s].$$

Hence,

$$\begin{aligned}(Pt_1...t_k)_{\mathbf{M}}[s[x_i := m]] &= P_{\mathbf{M}}((t_1)_{\mathbf{M}}[s[x_i := m]], ..., (t_k)_{\mathbf{M}}[s[x_i := m]])\\ &= P_{\mathbf{M}}((t_1[\mathbf{m}/x_i])_{\mathbf{M}}[s], ..., (t_k[\mathbf{m}/x_i])_{\mathbf{M}}[s])\\ &= (Pt_1[\mathbf{m}/x_i]...t_k[\mathbf{m}/x_i])_{\mathbf{M}}[s]\\ &= (A[\mathbf{m}/x_i])_{\mathbf{M}}[s],\end{aligned}$$

establishing (2).

It is obvious that (2) holds for the constant $\perp$.

If A is an atomic formula of the form $\doteq t_1 t_2$, then

$$(\doteq t_1 t_2)_{\mathbf{M}}[s[x_i := m]] = \mathbf{T} \text{ iff } (t_1)_{\mathbf{M}}[s[x_i := m]] = (t_2)_{\mathbf{M}}[s[x_i := m]].$$

By part (1) of the lemma, for $j = 1, 2$, we have

$$(t_j)_{\mathbf{M}}[s[x_i := m]] = (t_j[\mathbf{m}/x_i])_{\mathbf{M}}[s].$$

Hence,

$$\begin{gathered}(\doteq t_1 t_2)_{\mathbf{M}}[s[x_i := m]] = \mathbf{T} \text{ iff}\\ (t_1)_{\mathbf{M}}[s[x_i := m]] = (t_2)_{\mathbf{M}}[s[x_i := m]] \text{ iff}\\ (t_1[\mathbf{m}/x_i])_{\mathbf{M}}[s] = (t_2[\mathbf{m}/x_i])_{\mathbf{M}}[s] \text{ iff}\\ ((\doteq t_1 t_2)[\mathbf{m}/x_i])_{\mathbf{M}}[s] = \mathbf{T},\end{gathered}$$

establishing (2).

If A is a formula of the form $(B * C)$ where $* \in \{\wedge, \vee, \supset, \equiv\}$, we have

$$\begin{aligned}(B * C)_{\mathbf{M}}[s[x_i := m]] &= *_{\mathbf{M}}(B_{\mathbf{M}}, C_{\mathbf{M}})[s[x_i := m]]\\ &= H_*(B_{\mathbf{M}}[s[x_i := m]], C_{\mathbf{M}}[s[x_i := m]]).\end{aligned}$$

By the induction hypothesis,

$$\begin{gathered}B_{\mathbf{M}}[s[x_i := m]] = (B[\mathbf{m}/x_i])_{\mathbf{M}}[s] \text{ and}\\ C_{\mathbf{M}}[s[x_i := m]] = (C[\mathbf{m}/x_i])_{\mathbf{M}}[s].\end{gathered}$$

Hence,

$$\begin{aligned}&(B * C)_{\mathbf{M}}[s[x_i := m]] = H_*(B_{\mathbf{M}}[s[x_i := m]], C_{\mathbf{M}}[s[x_i := m]])\\ &= H_*(B[\mathbf{m}/x_i])_{\mathbf{M}}[s], C[\mathbf{m}/x_i])_{\mathbf{M}}[s]) = ((B * C)[\mathbf{m}/x_i])_{\mathbf{M}}[s],\end{aligned}$$

establishing (2).

If A is of the form $\neg B$, then

$$(\neg B)_{\mathbf{M}}[s[x_i := m]] = \neg_{\mathbf{M}}(B_{\mathbf{M}})[s[x_i := m]] = H_\neg(B_{\mathbf{M}}[s[x_i := m]]).$$

By the induction hypothesis,

$$B_{\mathbf{M}}[s[x_i := m]] = (B[\mathbf{m}/x_i])_{\mathbf{M}}[s].$$

Hence,

$$(\neg B)_{\mathbf{M}}[s[x_i := m]] = H_{\neg}(B_{\mathbf{M}}[s[x_i := m]])$$
$$= H_{\neg}(B[\mathbf{m}/x_i])_{\mathbf{M}}[s] = ((\neg B)[\mathbf{m}/x_i])_{\mathbf{M}}[s],$$

establishing (2).

If A is of the form $\forall x_j B$, there are two cases. If $x_i \neq x_j$, then

$$(\forall x_j B)_{\mathbf{M}}[s[x_i := m]] = \mathbf{T} \text{ iff}$$
$$B_{\mathbf{M}}[s[x_i := m][x_j := a]] = \mathbf{T} \text{ for all } a \in M.$$

By the induction hypothesis,

$$B_{\mathbf{M}}[s[x_i := m][x_j := a]] = (B[\mathbf{a}/x_j])_{\mathbf{M}}[s[x_i := m]],$$

and by one more application of the induction hypothesis,

$$(B[\mathbf{a}/x_j])_{\mathbf{M}}[s[x_i := m]] = (B[\mathbf{a}/x_j][\mathbf{m}/x_i])_{\mathbf{M}}[s].$$

By problem 5.2.3, since $x_i \neq x_j$ and $\mathbf{m}$ and $\mathbf{a}$ are constants,

$$B[\mathbf{a}/x_j][\mathbf{m}/x_i] = B[\mathbf{m}/x_i][\mathbf{a}/x_j].$$

Hence,

$$B_{\mathbf{M}}[s[x_i := m][x_j := a]] = \mathbf{T} \text{ for all } a \in M \text{ iff}$$
$$(B[\mathbf{a}/x_j][\mathbf{m}/x_i])_{\mathbf{M}}[s] = \mathbf{T} \text{ for all } a \in M \text{ iff}$$
$$((B[\mathbf{m}/x_i])[\mathbf{a}/x_j])_{\mathbf{M}}[s] = \mathbf{T} \text{ for all } a \in M.$$

By the induction hypothesis,

$$((B[\mathbf{m}/x_i])[\mathbf{a}/x_j])_{\mathbf{M}}[s] = (B[\mathbf{m}/x_i])_{\mathbf{M}}[s[x_j := a]].$$

Hence,

$$((B[\mathbf{m}/x_i])[\mathbf{a}/x_j])_{\mathbf{M}}[s] = \mathbf{T} \text{ for all } a \in M \text{ iff}$$
$$(B[\mathbf{m}/x_i])_{\mathbf{M}}[s[x_j := a]] = \mathbf{T} \text{ for all } a \in M \text{ iff}$$
$$((\forall x_j B)[\mathbf{m}/x_i])_{\mathbf{M}}[s] = \mathbf{T},$$

establishing (2).

If $x_i = x_j$, then

$$s[x_i := m][x_j := a] = s[x_i := a] \text{ and } (\forall x_j B)[\mathbf{m}/x_i] = \forall x_i B,$$

and so

$$\begin{gathered}(\forall x_j B)_{\mathbf{M}}[s[x_i := m]] = \mathbf{T} \text{ iff}\\ B_{\mathbf{M}}[s[x_i := m][x_j := a]] = \mathbf{T} \text{ for all } a \in M \text{ iff}\\ B_{\mathbf{M}}[s[x_i := a]] = \mathbf{T} \text{ for all } a \in M \text{ iff}\\ (\forall x_i B)_{\mathbf{M}}[s] = \mathbf{T} \text{ iff}\\ ((\forall x_j B)[\mathbf{m}/x_i])_{\mathbf{M}}[s] = \mathbf{T},\end{gathered}$$

establishing (2).

The case in which A is of the form $\exists x_j B$ is similar to the previous case and is left as an exercise. This concludes the induction proof for (2). □

We can now prove that the new definition of the truth of a quantified formula is equivalent to the old one.

Lemma 5.3.2 For any formula B, for any assignment $s \in [\mathbf{V} \to M]$ and any variable x_i, the following hold:

(1) $(\forall x_i B)_{\mathbf{M}}[s] = \mathbf{T}$ iff $(B[\mathbf{m}/x_i])_{\mathbf{M}}[s] = \mathbf{T}$ for all $m \in M$;

(2) $(\exists x_i B)_{\mathbf{M}}[s] = \mathbf{T}$ iff $(B[\mathbf{m}/x_i])_{\mathbf{M}}[s] = \mathbf{T}$ for some $m \in M$;

Proof: Recall that

$$(\forall x_i B)_{\mathbf{M}}[s] = \mathbf{T} \text{ iff } B_{\mathbf{M}}[s[x_i := m]] = \mathbf{T} \text{ for all } m \in M.$$

By lemma 5.3.1,

$$B_{\mathbf{M}}[s[x_i := m]] = (B[\mathbf{m}/x_i])_{\mathbf{M}}[s].$$

Hence,

$$\begin{gathered}(\forall x_i B)_{\mathbf{M}}[s] = \mathbf{T} \text{ iff}\\ (B[\mathbf{m}/x_i])_{\mathbf{M}}[s] = \mathbf{T} \text{ for all } m \in M,\end{gathered}$$

proving (1).

Also recall that

$$(\exists x_i B)_{\mathbf{M}}[s] = \mathbf{T} \text{ iff } B_{\mathbf{M}}[s[x_i := m]] = \mathbf{T} \text{ for some } m \in M.$$

By lemma 5.3.1,

$$B_{\mathbf{M}}[s[x_i := m]] = (B[\mathbf{m}/x_i])_{\mathbf{M}}[s].$$

Hence,

$$\begin{gathered}(\exists x_i B)_{\mathbf{M}}[s] = \mathbf{T} \text{ iff}\\ (B[\mathbf{m}/x_i])_{\mathbf{M}}[s] = \mathbf{T} \text{ for some } m \in M,\end{gathered}$$

proving (2). $\square$

In view of lemma 5.3.2, the recursive clauses of the definition of satisfaction can also be stated more informally as follows:

$$\begin{aligned}
\mathbf{M} &\models (\neg A)[s] \text{ iff } \mathbf{M} \not\models A[s],\\
\mathbf{M} &\models (A \land B)[s] \text{ iff } \mathbf{M} \models A[s] \textit{ and } \mathbf{M} \models B[s],\\
\mathbf{M} &\models (A \lor B)[s] \text{ iff } \mathbf{M} \models A[s] \textit{ or } \mathbf{M} \models B[s],\\
\mathbf{M} &\models (A \supset B)[s] \text{ iff } \mathbf{M} \not\models A[s] \textit{ or } \mathbf{M} \models B[s],\\
\mathbf{M} &\models (A \equiv B)[s] \text{ iff } (\mathbf{M} \models A[s] \; iff \; \mathbf{M} \models B[s]),\\
\mathbf{M} &\models (\forall x_i)A[s] \text{ iff } \mathbf{M} \models (A[\mathbf{a}/x_i])[s] \text{ for every } a \in M,\\
\mathbf{M} &\models (\exists x_i)A[s] \text{ iff } \mathbf{M} \models (A[\mathbf{a}/x_i])[s] \text{ for some } a \in M.
\end{aligned}$$

5.3.5 Free Variables and Semantics of Formulae

If A is a formula and the set $FV(A)$ of variables free in A is $\{y_1, ..., y_n\}$, for every assignment s, the truth value $A_{\mathbf{M}}[s]$ only depends on the restriction of s to $\{y_1, ..., y_n\}$. The following lemma makes the above statement precise.

Lemma 5.3.3 Given a formula A with set of free variables $\{y_1, ..., y_n\}$, for any two assignments s_1, s_2 such that $s_1(y_i) = s_2(y_i)$ for $1 \leq i \leq n$,

$$A_{\mathbf{M}}[s_1] = A_{\mathbf{M}}[s_2].$$

Proof: The lemma is proved using the induction principle for terms and formulae. First, let t be a term, and assume that s_1 and s_2 agree on $FV(t)$.

If $t = c$ (a constant), then

$$t_{\mathbf{M}}[s_1] = c_{\mathbf{M}} = t_{\mathbf{M}}[s_2].$$

If $t = y$ (a variable), then

$$t_{\mathbf{M}}[s_1] = s_1(y) = s_2(y) = t_{\mathbf{M}}[s_2],$$

since $FV(t) = \{y\}$ and s_1 and s_2 agree on $FV(t)$.

If $t = ft_1...t_n$, then

$$t_{\mathbf{M}}[s_1] = f_{\mathbf{M}}((t_1)_{\mathbf{M}}[s_1], ..., (t_n)_{\mathbf{M}}[s_1]).$$

Since every $FV(t_i)$ is a subset of $FV(t)$, $1 \leq i \leq n$, and s_1 and s_2 agree on $FV(t)$, by the induction hypothesis,

$$(t_i)_{\mathbf{M}}[s_1] = (t_i)_{\mathbf{M}}[s_2],$$

for all i, $1 \leq i \leq n$. Hence,

$$t_{\mathbf{M}}[s_1] = f_{\mathbf{M}}((t_1)_{\mathbf{M}}[s_1], ..., (t_n)_{\mathbf{M}}[s_1])$$
$$= f_{\mathbf{M}}((t_1)_{\mathbf{M}}[s_2], ..., (t_n)_{\mathbf{M}}[s_2]) = t_{\mathbf{M}}[s_2].$$

Now, let A be a formula, and assume that s_1 and s_2 agree on $FV(A)$.

If $A = Pt_1...t_n$, since $FV(t_i)$ is a subset of $FV(A)$ and s_1 and s_2 agree on $FV(A)$, we have

$$(t_i)_{\mathbf{M}}[s_1] = (t_i)_{\mathbf{M}}[s_2],$$

$1 \leq i \leq n$. But then, we have

$$A_{\mathbf{M}}[s_1] = P_{\mathbf{M}}((t_1)_{\mathbf{M}}[s_1], ..., (t_n)_{\mathbf{M}}[s_1])$$
$$= P_{\mathbf{M}}((t_1)_{\mathbf{M}}[s_2], ..., (t_n)_{\mathbf{M}}[s_2]) = A_{\mathbf{M}}[s_2].$$

If $A = \doteq t_1t_2$, since $FV(t_1)$ and $FV(t_2)$ are subsets of $FV(t)$ and s_1 and s_2 agree on $FV(A)$,

$$(t_1)_{\mathbf{M}}[s_1] = (t_1)_{\mathbf{M}}[s_2] \text{ and } (t_2)_{\mathbf{M}}[s_1] = (t_2)_{\mathbf{M}}[s_2],$$

and so

$$A_{\mathbf{M}}[s_1] \ = \ A_{\mathbf{M}}[s_2].$$

The cases in which A is of the form $(B \vee C)$, or $(B \wedge C)$, or $(B \supset C)$, or $(A \equiv B)$, or $\neg B$ are easily handled by induction as in lemma 3.3.1 (or 5.3.1), and the details are left as an exercise. Finally, we treat the case in which A is of the form $\forall xB$ (the case $\exists xB$ is similar).

If $A = \forall xB$, then $FV(A) = FV(B) - \{x\}$. First, recall the following property shown in problem 5.2.4: For every constant c,

$$FV(B[c/x]) = FV(B) - \{x\}.$$

Recall that from lemma 5.3.2,

$$A_{\mathbf{M}}[s_1] = \mathbf{T} \text{ iff } (B[\mathbf{a}/x])_{\mathbf{M}}[s_1] = \mathbf{T} \text{ for every } a \in M.$$

Since s_1 and s_2 agree on $FV(A) = FV(B) - \{x\}$ and $FV(B[\mathbf{a}/x]) = FV(B) - \{x\}$, by the induction hypothesis,

$$(B[\mathbf{a}/x])_{\mathbf{M}}[s_1] = (B[\mathbf{a}/x])_{\mathbf{M}}[s_2] \text{ for every } a \in M.$$

This shows that

$$(B[\mathbf{a}/x])_{\mathbf{M}}[s_1] = \mathbf{T} \text{ iff } (B[\mathbf{a}/x])_{\mathbf{M}}[s_2] = \mathbf{T},$$

that is,

$$A_{\mathbf{M}}[s_1] = A_{\mathbf{M}}[s_2].$$

□

As a consequence, if A is a sentence (that is $FV(A) = \emptyset$) the truth value of A in $\mathbf{M}$ is the same for all assignments. Hence for a *sentence* A, for every structure $\mathbf{M}$,

$$\text{either } \mathbf{M} \models A \ \text{ or } \ \mathbf{M} \models \neg A.$$

5.3.6 Subformulae and Rectified Formulae

In preparation for the next section in which we present a Gentzen system that is sound and complete with respect to the semantics, we need the following definitions and lemmas.

Definition 5.3.9 Given a formula A, a formula X is a *subformula* of A if either:

(i) $A = Pt_1...t_n$ and $X = A$, or

(ii) $A = \doteq t_1t_2$ and $X = A$, or

(iii) $A = \perp$ and $X = A$, or

(iv) $A = (B * C)$ and $X = A$, or X is a subformula of B, or X is a subformula of C, where $* \in \{\land, \lor, \supset, \equiv\}$, or

(v) $A = \neg B$ and $X = A$, or X is a subformula of B, or

(vi) $A = \forall xB$ and $X = A$, or X is a subformula of $B[t/x]$ for any term t free for x in B, or

(vii) $A = \exists xB$ and $X = A$, or X is a subformula of $B[t/x]$ for any term t free for x in B.

Definition 5.3.10 A quantifier $\forall$ (or $\exists$) *binds* an occurrence of a variable x in A iff A contains a subformula of the form $\forall xB$ (or $\exists xB$).

A formula A is *rectified* if

(i) $FV(A)$ and $BV(A)$ are disjoint and

(ii) Distinct quantifiers in A bind occurrences of distinct variables.

The following lemma for *renaming variables apart* will be needed later.

Lemma 5.3.4 (i) For every formula A,

$$\models \forall xA \equiv \forall yA[y/x]$$

and

$$\models \exists xA \equiv \exists yA[y/x],$$

for every variable y free for x in A and not in $FV(A) - \{x\}$.

(ii) There is an algorithm such that, for any input formula A, the algorithm produces a rectified formula A' such that

$$\models A \equiv A'$$

(that is, A' is semantically equivalent to A).

Proof: We prove (i), leaving (ii) as an exercise. First, we show that for every term t and every term r,

$$t[r/x] = t[y/x][r/y], \text{ if } y \text{ is not in } FV(t) - \{x\}.$$

The proof is by induction on the structure of terms and it is left as an exercise.

Now, let A be a formula, $\mathbf{M}$ be a structure, s an assignment, and a any element in M. We show that

$$(A[\mathbf{a}/x])_{\mathbf{M}}[s] = (A[y/x][\mathbf{a}/y])_{\mathbf{M}}[s],$$

provided that y is free for x in A, and that $y \notin FV(A) - \{x\}$.

This proof is by induction on formulae. We only treat the case of quantifiers, leaving the other cases as an exercise. We consider the case $A = \forall zB$, the case $\exists zB$ being similar. Recall that the following property was shown in problem 5.2.5:

If y is not in $FV(A)$, then $A[t/y] = A$ for every term t. Also recall that

$$FV(A) = FV(B) - \{z\}.$$

If $z = x$, since $y \notin FV(A) - \{x\} = FV(A) - \{z\}$ and z is not free in A, we know that x and y are not free in A, and

$$A[\mathbf{a}/x] = A, \; A[y/x] = A, \text{ and } A[\mathbf{a}/y] = A.$$

If $z \neq x$, then

$$A[\mathbf{a}/x] = \forall zB[\mathbf{a}/x].$$

Since y is free for x in A, we know that $y \neq z$, y is free for x in B, and

$$A[y/x][\mathbf{a}/y] = \forall zB[y/x][\mathbf{a}/y]$$

(a constant is always free for a substitution). Furthermore, since $y \notin FV(A) - \{x\}$, $FV(A) = FV(B) - \{z\}$ and $z \neq y$, we have $y \notin FV(B) - \{x\}$. By the induction hypothesis,

$$(B[\mathbf{a}/x])_{\mathbf{M}}[s] = (B[y/x][\mathbf{a}/y])_{\mathbf{M}}[s] \text{ for every } a \in M,$$

which proves that

$$(A[\mathbf{a}/x])_{\mathbf{M}}[s] = (A[y/x][\mathbf{a}/y])_{\mathbf{M}}[s] \text{ for every } a \in M.$$

(ii) This is proved using the induction principle for formulae and repeated applications of (i). □

EXAMPLE 5.3.3

Let

$$A = (\forall x(Rxy \supset Px) \wedge \forall y(\neg Rxy \wedge \forall x Px)).$$

A rectified formula equivalent to A is

$$B = (\forall u(Ruy \supset Pu) \wedge \forall v(\neg Rxv \wedge \forall z Pz)).$$

From now on, we will assume that we are dealing with rectified formulae.

5.3.7 Valid Formulae Obtained by Substitution in Tautologies

As in propositional logic we will be particularly interested in those formulae that are valid. An easy way to generate valid formulae is to substitute formulae for the propositional symbols in a propositional formula.

Definition 5.3.11 Let $\mathbf{L}$ be a first-order language. Let $\mathbf{PS}_0$ be the subset of $\mathbf{PS}$ consisting of all predicate symbols of rank 0 (the propositional letters). Consider the set $PROP_{\mathbf{L}}$ of all $\mathbf{L}$-formulae obtained as follows: $PROP_{\mathbf{L}}$ consists of all $\mathbf{L}$-formulae A such that for some tautology B and some substitution

$$\sigma : \mathbf{PS}_0 \to FORM_{\mathbf{L}}$$

assigning an arbitrary formula to each propositional letter in $\mathbf{PS}_0$,

$$A = \widehat{\sigma}(B),$$

the result of performing the substitution σ on B.

EXAMPLE 5.3.4

Let

$$B = (P \equiv Q) \equiv ((P \supset Q) \wedge (Q \supset P)).$$

If $\sigma(P) = \forall x C$ and $\sigma(Q) = \exists y(C \wedge D)$, then

$$A = (\forall x C \equiv \exists y(C \wedge D)) \equiv ((\forall x C \supset \exists y(C \wedge D)) \wedge (\exists y(C \wedge D) \supset \forall x C))$$

is of the above form.

However, note that not all **L**-formulae can be obtained by substituting formulae for propositional letters in tautologies. For example, the formulae $\forall xC$ and $\exists y(C \wedge D)$ cannot be obtained in this fashion.

The main property of the formulae in $PROP_{\mathbf{L}}$ is that they are valid. Note that not all valid formulae can be obtained in this fashion. For example, we will prove shortly that the formula $\neg\forall xA \equiv \exists x\neg A$ is valid.

Lemma 5.3.5 Let A be a formula obtained by substitution into a tautology as explained in definition 5.3.11. Then A is valid.

Proof: Let $\sigma : \mathbf{PS}_0 \rightarrow FORM_{\mathbf{L}}$ be the substitution and B the proposition such that $\widehat{\sigma}(B) = A$. First, we prove by induction on propositions that for every **L**-structure **M** and every assignment s, for every formula A in $PROP_{\mathbf{L}}$, if v is the valuation defined such that for every propositional letter P,

$$v(P) = (\sigma(P))_{\mathbf{M}}[s],$$

then

$$\widehat{v}(B) = A_{\mathbf{M}}[s].$$

The base case $B = P$ is obvious by definition of v. If B is of the form $(C * D)$ with $* \in \{\vee, \wedge, \supset, \equiv\}$, we have

$$\begin{aligned}
\widehat{v}((C * D)) &= H_*(\widehat{v}(C), \widehat{v}(D)),\\
\widehat{\sigma}((C * D)) &= (\widehat{\sigma}(C) * \widehat{\sigma}(D)),\\
(\widehat{\sigma}((C * D)))_{\mathbf{M}}[s] &= (\widehat{\sigma}(C) * \widehat{\sigma}(D))_{\mathbf{M}}[s]\\
&= *_{\mathbf{M}}(\widehat{\sigma}(C)_{\mathbf{M}}, \widehat{\sigma}(D)_{\mathbf{M}})[s]\\
&= H_*(\widehat{\sigma}(C)_{\mathbf{M}}[s], \widehat{\sigma}(D)_{\mathbf{M}}[s]).
\end{aligned}$$

By the induction hypothesis,

$$\widehat{v}(C) = (\widehat{\sigma}(C))_{\mathbf{M}}[s] \text{ and } \widehat{v}(D) = (\widehat{\sigma}(D))_{\mathbf{M}}[s].$$

Hence,

$$\begin{gathered}
\widehat{v}((C * D)) = H_*(\widehat{v}(C), \widehat{v}(D))\\
= H_*((\widehat{\sigma}(C))_{\mathbf{M}}[s], (\widehat{\sigma}(D))_{\mathbf{M}}[s]) = (\widehat{\sigma}((C * D)))_{\mathbf{M}}[s],
\end{gathered}$$

as desired.

If B is of the form $\neg C$, then we have

$$\widehat{v}(\neg C) = H_\neg(\widehat{v}(C)),\ \widehat{\sigma}(\neg C) = \neg\widehat{\sigma}(C),$$

and

$$(\widehat{\sigma}(\neg C))_{\mathbf{M}}[s] = (\neg\widehat{\sigma}(C))_{\mathbf{M}}[s] = \neg_{\mathbf{M}}((\widehat{\sigma}(C))_{\mathbf{M}})[s] = H_\neg((\widehat{\sigma}(C))_{\mathbf{M}}[s]).$$

By the induction hypothesis,

$$\widehat{v}(C) = (\widehat{\sigma}(C))_{\mathbf{M}}[s].$$

Hence,

$$\widehat{v}(\neg C) = H_{\neg}(\widehat{v}(C)) = H_{\neg}((\widehat{\sigma}(C))_{\mathbf{M}}[s]) = (\widehat{\sigma}(\neg C))_{\mathbf{M}}[s],$$

as desired. This completes the induction proof. To conclude the lemma, observe that if B is a tautology,

$$\widehat{v}(B) = \mathbf{T}.$$

Hence, by the above property, for every structure $\mathbf{M}$ and every assignment s, there is a valuation v such that $\widehat{v}(B) = A_{\mathbf{M}}[s]$, and so $A_{\mathbf{M}}[s] = \mathbf{T}$. $\square$

5.3.8 Complete Sets of Connectives

As in the propositional case, the logical connectives are not independent. The following lemma shows how they are related.

Lemma 5.3.6 The following formulae are valid for all formulae A,B.

$$(A \equiv B) \equiv ((A \supset B) \wedge (B \supset A)) \quad (1)$$
$$(A \supset B) \equiv (\neg A \vee B) \quad (2)$$
$$(A \vee B) \equiv (\neg A \supset B) \quad (3)$$
$$(A \vee B) \equiv \neg(\neg A \wedge \neg B) \quad (4)$$
$$(A \wedge B) \equiv \neg(\neg A \vee \neg B) \quad (5)$$
$$\neg A \equiv (A \supset \perp) \quad (6)$$
$$\perp \equiv (A \wedge \neg A) \quad (7)$$
$$\neg \forall x A \equiv \exists x \neg A \quad (8)$$
$$\neg \exists x A \equiv \forall x \neg A \quad (9)$$
$$\forall x A \equiv \neg \exists x \neg A \quad (10)$$
$$\exists x A \equiv \neg \forall x \neg A \quad (11)$$

Proof: The validity of (1) to (7) follows from lemma 5.3.5, since these formulae are obtained by substitution in tautologies. To prove (8), recall that for any formula B,

$$(\forall x B)_{\mathbf{M}}[s] = \mathbf{T} \text{ iff } B_{\mathbf{M}}[s[x := m]] = \mathbf{T} \text{ for all } m \in M,$$

and

$$(\exists x B)_{\mathbf{M}}[s] = \mathbf{T} \text{ iff } B_{\mathbf{M}}[s[x := m]] = \mathbf{T} \text{ for some } m \in M.$$

Furthermore,

$$(\neg B)_{\mathbf{M}}[s] = \mathbf{T} \text{ iff } B_{\mathbf{M}}[s] = \mathbf{F}.$$

Hence,

$$\begin{gathered}
(\neg\forall x A)_{\mathbf{M}}[s] = \mathbf{T} \text{ iff}\\
(\forall x A)_{\mathbf{M}}[s] = \mathbf{F} \text{ iff}\\
A_{\mathbf{M}}[s[x := m]] = \mathbf{F} \text{ for some } m \in M \text{ iff}\\
(\neg A)_{\mathbf{M}}[s[x := m]] = \mathbf{T} \text{ for some } m \in M \text{ iff}\\
(\exists x \neg A)_{\mathbf{M}}[s] = \mathbf{T}.
\end{gathered}$$

The proof of (9) is similar.

To prove (11), observe that

$$\begin{gathered}
(\neg\forall x \neg A)_{\mathbf{M}}[s] = \mathbf{T} \text{ iff}\\
(\forall x \neg A)_{\mathbf{M}}[s] = \mathbf{F} \text{ iff}\\
(\neg A)_{\mathbf{M}}[s[x := m]] = \mathbf{F} \text{ for some } m \in M, \text{ iff}\\
A_{\mathbf{M}}[s[x := m]] = \mathbf{T} \text{ for some } m \in M, \text{ iff}\\
(\exists x A)_{\mathbf{M}}[s] = \mathbf{T}.
\end{gathered}$$

The proof of (10) is similar. □

The above lemma shows that, as in the propositional case, we can restrict our attention to any functionally complete set of connectives. But it also shows that we can either dispense with $\exists$ or with $\forall$, taking $\exists x A$ as an abbreviation for $\neg\forall x \neg A$, or $\forall x A$ as an abbreviation for $\neg\exists x \neg A$.

In the rest of this text, we shall use mostly the set $\{\wedge, \vee, \neg, \supset, \forall, \exists\}$ even though it is not minimal, because it is particularly convenient and natural. Hence, $(A \equiv B)$ will be viewed as an abbreviation for $((A \supset B) \wedge (B \supset A))$ and $\perp$ as an abbreviation for $(P \wedge \neg P)$.

5.3.9 Logical Equivalence and Boolean Algebras

As in the propositional case, we can also define the notion of logical equivalence for formulae.

Definition 5.3.12 The relation $\simeq$ on $FORM_{\mathbf{L}}$ is defined so that for any two formulae A and B,

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

We say that A and B are *logically equivalent*, or for short, *equivalent*.

It is immediate to show that $\simeq$ 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 5.3.7 For all formulae A, A', B, B', the following properties hold: If $A \simeq A'$ and $B \simeq B'$ then, for $* \in \{\wedge, \vee, \supset, \equiv\}$,

$$\begin{aligned}(A * B) &\simeq (A' * B'),\\ \neg A &\simeq \neg A',\\ \forall x A &\simeq \forall x A' \text{ and}\\ \exists x A &\simeq \exists x A'.\end{aligned}$$

Proof: First, we note that a formula $(A \equiv B)$ is valid iff for every structure $\mathbf{M}$ and every assignment s,

$$A_{\mathbf{M}}[s] = B_{\mathbf{M}}[s].$$

Then, the proof is similar to the proof of the propositional case (lemma 3.3.5) for formulae of the form $(A * B)$ or $\neg A$.

Since

$$(\forall x A)_{\mathbf{M}}[s] = \mathbf{T} \text{ iff } A_{\mathbf{M}}[s[x := m]] = \mathbf{T} \text{ for all } m \in M,$$

and $A \simeq A'$ implies that

$$A_{\mathbf{M}}[s] = A'_{\mathbf{M}}[s] \text{ for all } \mathbf{M} \text{ and } s,$$

then

$$\begin{aligned}A_{\mathbf{M}}[s[x := m]] = \mathbf{T} &\text{ for all } m \in M \text{ iff}\\ A'_{\mathbf{M}}[s[x := m]] = \mathbf{T} &\text{ for all } m \in M \text{ iff}\\ (\forall x A')_{\mathbf{M}}[s] = \mathbf{T}&.\end{aligned}$$

Hence $\forall x A$ and $\forall x A'$ are equivalent. The proof that if $A \simeq A'$ then $\exists x A \simeq \exists x A'$ is similar. $\square$

In the rest of this section, it is assumed that the constant symbol $\top$ is added to the alphabet of definition 5.2.1, and that $\top$ is interpreted as $\mathbf{T}$. By lemma 5.3.5 and lemma 3.3.6, the following identities hold.

Lemma 5.3.8 The following identities hold.

$$\begin{gathered}\text{Associativity rules:}\\ ((A \vee B) \vee C) \simeq (A \vee (B \vee C)) \quad ((A \wedge B) \wedge C) \simeq (A \wedge (B \wedge C))\\ \text{Commutativity rules:}\\ (A \vee B) \simeq (B \vee A) \quad (A \wedge B) \simeq (B \wedge A)\\ \text{Distributivity rules:}\\ (A \vee (B \wedge C)) \simeq ((A \vee B) \wedge (A \vee C))\end{gathered}$$

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

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 formula A modulo $\simeq$ as $[A]$, and the set of all such equivalence classes as $\mathbf{B_L}$. As in Chapter 3, we define the operations $+$, $*$ and $\neg$ on $\mathbf{B_L}$ as follows:

$$[A] + [B] = [A \lor B],$$
$$[A] * [B] = [A \land B],$$
$$\neg[A] = [\neg A].$$

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

By lemma 5.3.7, the above functions (and constants) are independent of the choice of representatives in the equivalence classes, and the properties of lemma 5.3.8 are identities valid on the set $\mathbf{B_L}$ of equivalence classes modulo $\simeq$. The structure $\mathbf{B_L}$ is a *boolean algebra* called the *Lindenbaum algebra* of $\mathbf{L}$. This algebra is important for studying algebraic properties of formulae. Some of these properties will be investigated in the problems of the next section.

Remark: Note that properties of the quantifiers and of equality are not captured in the axioms of a boolean algebra. There is a generalization of the notion of a boolean algebra, the *cylindric algebra*, due to Tarski. Cylindric algebras have axioms for the existential quantifier and for equality. However, this topic is beyond the scope of this text. The interested reader is referred to Henkin, Monk, and Tarski, 1971.

PROBLEMS

5.3.1. Prove that the following formulae are valid:

$$\forall xA \supset A[t/x], \quad A[t/x] \supset \exists xA,$$
where t is free for x in A.

5.3.2. Let x, y be any distinct variables. Let A be any formula, C any formula not containing the variable x free, and let E be any formula such that x is free for y in E. Prove that the following formulae are valid:

$$\begin{array}{cc}
\forall xC \equiv C & \exists xC \equiv C \\
\forall x\forall yA \equiv \forall y\forall xA & \exists x\exists yA \equiv \exists y\exists xA \\
\forall x\forall yE \supset \forall xE[x/y] & \exists xE[x/y] \supset \exists x\exists yE \\
\forall xA \supset \exists xA & \\
\exists x\forall yA \supset \forall y\exists xA &
\end{array}$$

5.3.3. Let A, B be any formulae, and C any formula not containing the variable x free. Prove that the following formulae are valid:

$$\begin{array}{cc}
\neg\exists xA \equiv \forall x\neg A & \neg\forall xA \equiv \exists x\neg A \\
\exists xA \equiv \neg\forall x\neg A & \forall xA \equiv \neg\exists x\neg A \\
\forall xA \wedge \forall xB \equiv \forall x(A \wedge B) & \exists xA \vee \exists xB \equiv \exists x(A \vee B) \\
C \wedge \forall xA \equiv \forall x(C \wedge A) & C \vee \exists xA \equiv \exists x(C \vee A) \\
C \wedge \exists xA \equiv \exists x(C \wedge A) & C \vee \forall xA \equiv \forall x(C \vee A) \\
\exists x(A \wedge B) \supset \exists xA \wedge \exists xB & \forall xA \vee \forall xB \supset \forall x(A \vee B)
\end{array}$$

5.3.4. Let A, B be any formulae, and C any formula not containing the variable x free. Prove that the following formulae are valid:

$$\begin{array}{cc}
(C \supset \forall xA) \equiv \forall x(C \supset A) & (C \supset \exists xA) \equiv \exists x(C \supset A) \\
(\forall xA \supset C) \equiv \exists x(A \supset C) & (\exists xA \supset C) \equiv \forall x(A \supset C) \\
(\forall xA \supset \exists xB) \equiv \exists x(A \supset B) & \\
(\exists xA \supset \forall xB) \supset \forall x(A \supset B) &
\end{array}$$

5.3.5. Prove that the following formulae are not valid:

$$A[t/x] \supset \forall xA, \quad \exists xA \supset A[t/x],$$
where t is free for x in A.

5.3.6. Show that the following formulae are not valid:

$$\exists xA \supset \forall xA$$
$$\forall y\exists xA \supset \exists x\forall yA$$
$$\exists xA \wedge \exists xB \supset \exists x(A \wedge B)$$
$$\forall x(A \vee B) \supset \forall xA \vee \forall xB$$

5.3.7. Prove that the formulae of problem 5.3.4 are not necessarily valid if x is free in C.

5.3.8. Let A be any formula and B any formula in which the variable x does not occur free.

(a) Prove that

$$\text{if } \models (B \supset A), \text{ then } \models (B \supset \forall x A),$$

and

$$\text{if } \models (A \supset B), \text{ then } \models (\exists x A \supset B).$$

(b) Prove that the above may be false if x is free in B.

5.3.9. Let **L** be a first-order language, and **M** be an **L**-structure. For any formulae A, B, prove that:

(a) $\mathbf{M} \models A \supset B$ implies that ($\mathbf{M} \models A$ implies $\mathbf{M} \models B$),

but not vice versa.

(b) $A \models B$ implies that ($\models A$ implies $\models B$),

but not vice versa.

5.3.10. Given a formula A with set of free variables $FV(A) = \{x_1, ..., x_n\}$, the *universal closure* A' of A is the sentence $\forall x_1 ... \forall x_n A$. Prove that A is valid iff $\forall x_1 ... \forall x_n A$ is valid.

5.3.11. Let **L** be a first-order language, Γ a set of formulae, and B some formula. Let Γ' be the set of universal closures of formulae in Γ.

Prove that

$$\Gamma \models B \text{ implies that } \Gamma' \models B,$$

but not vice versa.

5.3.12. Let **L** be the first-order language with equality consisting of one unary function symbol f. Write formulae asserting that for every **L**-structure **M**:

(a) f is injective

(b) f is surjective

(c) f is bijective

5.3.13. Let **L** be a first-order language with equality.

Find a formula asserting that any **L**-structure **M** has at least n elements.

5.3.14. Prove that the following formulae are valid:

$$\forall x \exists y (x \doteq y)$$
$$A[t/x] \equiv \forall x((x \doteq t) \supset A), \text{ if } x \notin Var(t) \text{ and } t \text{ is free for } x \text{ in } A.$$
$$A[t/x] \equiv \exists x((x \doteq t) \wedge A), \text{ if } x \notin Var(t) \text{ and } t \text{ is free for } x \text{ in } A.$$

$*$ **5.3.15.** Let **A** and **B** be two **L**-structures. A surjective function $h : A \to B$ is a *homomorphism* of **A** onto **B** if:

(i) For every n-ary function symbol f, for every $(a_1, ..., a_n) \in A^n$,

$$h(f_{\mathbf{A}}(a_1, ..., a_n)) = f_{\mathbf{B}}(h(a_1), ..., h(a_n));$$

(ii) For every constant symbol c, $h(c_{\mathbf{A}}) = c_{\mathbf{B}}$;

(iii) For every n-ary predicate symbol P, for every $(a_1, ..., a_n) \in A^n$,

$$\text{if} \quad P_{\mathbf{A}}(a_1, ..., a_n) = \mathbf{T} \quad \text{then} \quad P_{\mathbf{B}}(h(a_1), ..., h(a_n)) = \mathbf{T}.$$

Let t be a term containing the free variables $\{y_1, ..., y_n\}$, and A be a formula with free variables $\{y_1, ..., y_n\}$. For any assignment s whose restriction to $\{y_1, ..., y_n\}$ is given by $s(y_i) = a_i$, the notation $t[a_1, ..., a_n]$ is equivalent to $t[s]$, and $A[a_1, ..., a_n]$ is equivalent to $A[s]$.

(a) Prove that for any term t, if t contains the variables $\{y_1, ..., y_n\}$, for every $(a_1, ..., a_n) \in A^n$, if h is a homomorphism from **A** onto **B**, then

$$h(t_{\mathbf{A}}[a_1, ..., a_n]) = t_{\mathbf{B}}[h(a_1), ..., h(a_n)],$$

even if h is not surjective.

A *positive formula* is a formula built up from atomic formulae (excluding $\perp$), using only the connectives $\wedge$, $\vee$ and the quantifiers $\forall$, $\exists$.

(b) Prove that for any positive formula X, for every $(a_1, ..., a_n) \in A^n$,

$$\text{if} \quad \mathbf{A} \models X[a_1, ..., a_n] \quad \text{then} \quad \mathbf{B} \models X[h(a_1), ..., h(a_n)],$$

where h is a homomorphism from **A** onto **B** (we say that positive formulae are preserved under homomorphisms).

A *strong homomorphism* between **A** and **B** is a homomorphism $h : \mathbf{A} \to \mathbf{B}$ such that for any n-ary predicate P and any $(a_1, ..., a_n) \in A^n$,

$$P_{\mathbf{A}}(a_1, ..., a_n) = \mathbf{T} \quad \text{iff} \quad P_{\mathbf{B}}(h(a_1), ..., h(a_n)) = \mathbf{T}.$$

An *isomorphism* is a bijective strong homomorphism.

(c) If **L** is a first-order language without equality, prove that for any formula X with free variables $\{y_1, ..., y_n\}$, for any $(a_1, ..., a_n) \in A^n$, if h is a strong homomorphism of **A** onto **B** then

$$\mathbf{A} \models X[a_1, ..., a_n] \quad \text{iff} \quad \mathbf{B} \models X[h(a_1), ..., h(a_n)].$$

(d) If **L** is a first-order language with or without equality, prove that for any formula X with free variables $\{y_1, ..., y_n\}$, for any $(a_1, ..., a_n) \in A^n$, if h is an isomorphism between **A** and **B** then

$$\mathbf{A} \models X[a_1, ..., a_n] \quad \text{iff} \quad \mathbf{B} \models X[h(a_1), ..., h(a_n)].$$

(e) Find two structures, a homomorphism h between them, and a nonpositive formula X that is not preserved under h.

5.3.16. Let A be the sentence:

$$\forall x \neg R(x, x) \land \forall x \forall y \forall z (R(x, y) \land R(y, z) \supset R(x, z)) \land \\ \forall x \exists y R(x, y).$$

Give an infinite model for A and prove that A has *no* finite model.

∗ **5.3.17.** The *monadic predicate calculus* is the language **L** with no equality having only unary predicate symbols, and no function or constant symbols. Let **M** be an **L**-structure having n distinct unary predicates Q_1,...,Q_n on M. Define the relation $\cong$ on M as follows:

$$a \cong b \quad \text{iff} \\ Q_i(a) = \mathbf{T} \quad \text{iff} \quad Q_i(b) = \mathbf{T}, \text{ for all } i, 1 \leq i \leq n.$$

(a) Prove that $\cong$ is an equivalence relation having at most 2^n equivalence classes.

Let $\mathbf{M}/\cong$ be the structure having the set $M/\cong$ of equivalence classes modulo $\cong$ as its domain, and the unary predicates R_1,...,R_n such that, for every equivalence class $\overline{x}$,

$$R_i(\overline{x}) = \mathbf{T} \quad \text{iff} \quad Q_i(x) = \mathbf{T}.$$

(b) Prove that for every **L**-formula A containing only predicate symbols in the set $\{P_1, ..., P_n\}$,

$$\mathbf{M} \models A \quad \text{iff} \quad \mathbf{M}/\cong \models A,$$

and that

$$\models A \quad \text{iff} \\ \mathbf{M} \models A \quad \text{for all } \mathbf{L}\text{-structures } \mathbf{M} \text{ with at most } 2^n \text{ elements.}$$

(c) Using part (b), outline an algorithm for deciding validity in the monadic predicate calculus.

5.3.18. Give a detailed rectification algorithm (see lemma 5.3.4(ii)).

5.3.19. Let **A** and **B** be two **L**-structures. The structure **B** is a *substructure* of the structure **A** iff the following conditions hold:

(i) The domain B is a subset of the domain A;

(ii) For every constant symbol c,

$$c_{\mathbf{B}} = c_{\mathbf{A}};$$

(iii) For every function symbol f of rank $n > 0$, $f_{\mathbf{B}}$ is the restriction of $f_{\mathbf{A}}$ to B^n, and for every predicate symbol P of rank $n \geq 0$, $P_{\mathbf{B}}$ is the restriction of $P_{\mathbf{A}}$ to B^n.

A universal sentence is a sentence of the form

$$\forall x_1 ... \forall x_m B,$$

where B is quantifier free. Prove that if **A** is a model of a set Γ of universal sentences, then **B** is also a model of Γ.

$*$ **5.3.20.** In this problem, some properties of reduced products are investigated. We are considering first-order languages with or without equality.

The notion of a reduced product depends on that of a filter, and the reader is advised to consult problem 3.5.8 for the definition of filters and ultrafilters.

Let I be a nonempty set which will be used as an index set, and let D be a proper filter over I. Let $(A_i)_{i \in I}$ be an I-indexed family of nonempty sets. The Cartesian product C of these sets, denoted by

$$\prod_{i \in I} A_i$$

is the set of all I-indexed sequences

$$f : I \to \bigcup_{i \in I} A_i$$

such that, for each $i \in I$, $f(i) \in A_i$. Such I-sequences will also be denoted as $< f(i) \mid i \in I >$.

The relation $=_D$ on C is defined as follows:

For any two functions $f, g \in C$,

$$f =_D g \quad \text{iff} \quad \{i \in I \mid f(i) = g(i)\} \in D.$$

In words, f and g are related iff the set of indices on which they "agree" belongs to the proper filter D.

(a) Prove that $=_D$ is an equivalence relation.

(b) Let $\mathbf{L}$ be a first-order language (with or without equality). For each $i \in I$, let $\mathbf{A}_i$ be an $\mathbf{L}$-structure. We define the *reduced product* $\mathbf{B}$ of the $(A_i)_{i\in I}$ modulo D, as the $\mathbf{L}$-structure defined as follows:

(i) The domain of $\mathbf{B}$ the set of equivalence classes of

$$\prod_{i\in I} A_i$$

modulo $=_D$.

(ii) For every constant symbol c, c is interpreted in $\mathbf{B}$ as the equivalence class

$$[< c_{\mathbf{A}_i} \mid i \in I >]_D.$$

(iii) For every function symbol f of rank $n > 0$, f is interpreted as the function such that, for any n equivalence classes $G^1 = [< g^1(i) \mid i \in I >]_D$,...,$G^n = [< g^n(i) \mid i \in I >]_D$,

$$f_{\mathbf{B}}(G^1, ..., G^n) = [< f_{\mathbf{A}_i}(g^1(i), ..., g^n(i)) \mid i \in I >]_D.$$

(iv) For every predicate symbol P of rank $n \geq 0$, P is interpreted as the predicate such that, for any n equivalence classes $G^1 = [< g^1(i) \mid i \in I >]_D$,...,$G^n = [< g^n(i) \mid i \in I >]_D$,

$$P_{\mathbf{B}}(G^1, ..., G^n) = \mathbf{T} \quad \text{iff} \quad \{i \in I \mid P_{\mathbf{A}_i}(g^1(i), ..., g^n(i)) = \mathbf{T}\} \in D.$$

The reduced product $\mathbf{B}$ is also denoted by

$$\prod_D (A_i)_{i\in I}.$$

(c) Prove that $=_D$ is a congruence; that is, that definitions (ii) to (iv) are independent of the representatives chosen in the equivalence classes G^1,...,G^n.

$**$ **5.3.21.** Let $\mathbf{B}$ be the reduced product

$$\prod_D (A_i)_{i\in I}$$

as defined in problem 5.3.20. When D is an ultrafilter, $\mathbf{B}$ is called an *ultraproduct*.

(a) Prove that for every term t with free variables $\{y_1, ..., y_n\}$, for any n equivalence classes $G^1 = [< g^1(i) \mid i \in I >]_D$,...,$G^n = [< g^n(i) \mid i \in I >]_D$ in $\mathbf{B}$,

$$t_{\mathbf{B}}[G^1, ..., G^n] = [< t_{\mathbf{A}_i}[g^1(i), ..., g^n(i)] \mid \ \in I >]_D.$$

(b) Let D be an ultrafilter. Prove that for any formula A with free variables $FV(A) = \{y_1, ..., y_n\}$, for any n equivalence classes $G^1 = [< g^1(i) \mid i \in I >]_D$,...,$G^n = [< g^n(i) \mid i \in I >]_D$ in $\mathbf{B}$,

$$\mathbf{B} \models A[G^1, ..., G^n] \quad \text{iff} \quad \{i \in I \mid \mathbf{A}_i \models A[g^1(i), ..., g^n(i)]\} \in D.$$

(c) Prove that if D is an ultrafilter, for every sentence A,

$$\mathbf{B} \models A \quad \text{iff} \quad \{i \in I \mid \mathbf{A}_i \models A\} \in D.$$

Hint: Proceed by induction on formulae. The fact that D is an ultrafilter is needed in the case where A is of the form $\neg B$.

$*$ **5.3.22.** This problem generalizes problem 3.5.10 to first-order logic. It provides a proof of the compactness theorem for first-order languages of any cardinality.

(a) Let Γ be a set of sentences 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 $\mathbf{A}_i$ be a structure satisfying i. For each sentence $A \in \Gamma$, 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, there is an ultrafilter U including $\mathcal{C}$, so that every A^* is in U. If $i \in A^*$, then $A \in i$, and so

$$\mathbf{A}_i \models A.$$

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

Show that

$$\{i \in I \mid \mathbf{A}_i \models A\} \in U.$$

(b) Show that the ultraproduct $\mathbf{B}$ (defined in problem 5.3.20) satisfies Γ.

$**$ **5.3.23.** Given a first-order language $\mathbf{L}$, a *literal* is either an atomic formula or the negation of an atomic formula. A *basic Horn formula* is a disjunction of literals, in which at most one literal is positive. The class of *Horn formulae* is the least class of formulae containing the basic Horn formulae, and such that:

(i) If A and B are Horn formulae, so is $A \land B$;

(ii) If A is a Horn formula, so is $\forall x A$;

(iii) If A is a Horn formula, so is $\exists x A$.

A Horn sentence is a closed Horn formula.

Let A be a Horn formula, and let $(\mathbf{A}_i)_{i\in I}$ be an I-indexed family of structures satisfying A. Let D be a proper filter over D, and let $\mathbf{B}$ be the reduced product of $(\mathbf{A}_i)_{i\in I}$ modulo $=_D$.

(a) Prove that for any n equivalence classes $G^1 = [< g^1(i) \mid i \in I >]_D$,...,$G^n = [< g^n(i) \mid i \in I >]_D$ in $\mathbf{B}$,

$$\text{if} \quad \{i \in I \mid \mathbf{A}_i \models A[g^1(i), ..., g^n(i)]\} \in D \quad \text{then} \quad \mathbf{B} \models A[G^1, ..., G^n].$$

(b) Given a Horn sentence A,

$$\text{if} \quad \mathbf{A}_i \models A \quad \text{for all } \in I, \text{ then } \quad \mathbf{B} \models A.$$

(We say that Horn sentences are preserved under reduced products.)

(c) If we let the filter D be the family of sets $\{I\}$, the reduced product $\mathbf{B}$ is called the *direct product* of $(A_i)_{i\in I}$.

Show that the formula A below (which is not a Horn formula) is preserved under direct products, but is not preserved under reduced products:

A is the conjunction of the boolean algebra axioms plus the sentence

$$\exists x \forall y ((x \neq 0) \land (x * y = y \supset (y = x \lor y = 0))).$$

$*$ **5.3.24.** Show that in the proof of the compactness theorem given in problem 5.3.22, if Γ is a set of Horn formulae, it is not necessary to extend $\mathcal{C}$ to an ultrafilter, but simply to take the filter generated by $\mathcal{C}$.

$*$ **5.3.25.** Let $\mathbf{L}$ be a first-order language. Two $\mathbf{L}$-structures $\mathbf{M}_1$ and $\mathbf{M}_2$ are *elementary equivalent* iff for every sentence A,

$$\mathbf{M}_1 \models A \quad \text{iff} \quad \mathbf{M}_2 \models A.$$

Prove that if there is an isomorphism between $\mathbf{M}_1$ and $\mathbf{M}_2$, then $\mathbf{M}_1$ and $\mathbf{M}_2$ are elementary equivalent.

Note: The converse of the above statement holds if one of the structures has a finite domain. However, there are infinite structures that are elementary equivalent but are not isomorphic. For example, if the language **L** has a unique binary predicate $<$, the structure $\mathcal{Q}$ of the rational numbers and the structure $\mathcal{R}$ of the real numbers (with $<$ interpreted as the strict order of each structure) are elementary equivalent, but nonisomorphic since $\mathcal{Q}$ is countable but $\mathcal{R}$ is not.

5.4 Proof Theory of First-Order Languages

In this section, the Gentzen system G' is extended to first-order logic.

5.4.1 The Gentzen System G for Languages Without Equality

We first consider the case of a first-order language **L** *without equality*. In addition to the rules for the logical connectives, we need four new rules for the quantifiers. The rules presented below are designed to make the search for a counter-example as simple as possible. In the following sections, sequents are defined as before but are composed of formulae instead of propositions. Furthermore, it is assumed that the set of connectives is $\{\wedge, \vee, \supset, \neg, \forall, \exists\}$. By lemma 5.3.7, there is no loss of generality. Hence, when $\equiv$ is used, $(A \equiv B)$ is considered as an abbreviation for $((A \supset B) \wedge (B \supset A))$.

Definition 5.4.1 (Gentzen system G for languages without equality) The symbols Γ, Δ, Λ will be used to denote arbitrary sequences of formulae and A, B to denote formulae. The rules of the sequent calculus G are the following:

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

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

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

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

In the quantifier rules below, x is any variable and y is any variable free for x in A and not free in A, unless $y = x$ ($y \notin FV(A) - \{x\}$). The term t is any term free for x in A.

$$\frac{\Gamma, A[t/x], \forall xA, \Delta \to \Lambda}{\Gamma, \forall xA, \Delta \to \Lambda}\ (\forall : left) \qquad \frac{\Gamma \to \Delta, A[y/x], \Lambda}{\Gamma \to \Delta, \forall xA, \Lambda}\ (\forall : right)$$

$$\frac{\Gamma, A[y/x], \Delta \to \Lambda}{\Gamma, \exists xA, \Delta \to \Lambda}\ (\exists : left) \qquad \frac{\Gamma \to \Delta, A[t/x], \exists xA, \Lambda}{\Gamma \to \Delta, \exists xA, \Lambda}\ (\exists : right)$$

Note that in both the $(\forall : right)$-rule and the $(\exists : left)$-rule, the variable y does *not* occur free in the lower sequent. In these rules, the variable y is called the *eigenvariable* of the inference. The condition that the eigenvariable does not occur free in the conclusion of the rule is called the *eigenvariable condition*. The formula $\forall xA$ (or $\exists xA$) is called the *principal formula* of the inference, and the formula $A[t/x]$ (or $A[y/x]$) the *side formula* of the inference.

The *axioms* of G are all sequents $\Gamma \to \Delta$ such that Γ and Δ contain a common formula.

5.4.2 Deduction Trees for the System G

First, we define when a sequent is falsifiable or valid.

Definition 5.4.2 (i) A sequent $A_1, ..., A_m \to B_1, ..., B_n$ is *falsifiable* iff for some structure $\mathbf{M}$ and some assignment s:

$$\mathbf{M} \models (A_1 \land ... \land A_m \land \neg B_1 \land ... \land \neg B_n)[s].$$

Note that when $m = 0$ the condition reduces to

$$\mathbf{M} \models (\neg B_1 \land ... \land \neg B_n)[s]$$

and when $n = 0$ the condition reduces to

$$\mathbf{M} \models (A_1 \land ... \land A_m)[s].$$

(ii) A sequent $A_1, ..., A_m \to B_1, ..., B_n$ is *valid* iff for every structure $\mathbf{M}$ and every assignment s:

$$\mathbf{M} \models (\neg A_1 \lor ... \lor \neg A_m \lor B_1 \lor ... \lor B_n)[s].$$

This is denoted by

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

Note that a sequent is valid if and only if it is not falsifiable.

Deduction trees and *proof trees* for the system G are defined as in definition 3.4.5, but for formulae and with the rules given in definition 5.4.1.

EXAMPLE 5.4.1

Let $A = \exists x(P \supset Q(x)) \supset (P \supset \exists zQ(z))$, where P is a propositional symbol and Q a unary predicate symbol.

$$\dfrac{\dfrac{\dfrac{\dfrac{P \to P, \exists zQ(z) \qquad \dfrac{Q(y_1), P \to Q(y_1), \exists zQ(z)}{Q(y_1), P \to \exists zQ(z)}}{P, P \supset Q(y_1) \to \exists zQ(z)}}{P \supset Q(y_1) \to P \supset \exists zQ(z)}}{\exists x(P \supset Q(x)) \to P \supset \exists zQ(z)}}{\to \exists x(P \supset Q(x)) \supset (P \supset \exists zQ(z))}$$

The above is a proof tree for A. Note that $P[y_1/x] = P$.

EXAMPLE 5.4.2

Let $A = (\forall xP(x) \land \exists yQ(y)) \supset (P(f(v)) \land \exists zQ(z))$, where P and Q are unary predicate symbols, and f is a unary function symbol. The tree below is a proof tree for A.

$$\dfrac{\dfrac{\dfrac{\Pi \qquad \dfrac{\dfrac{P(f(v)), \forall xP(x), Q(u) \to Q(u), \exists zQ(z)}{P(f(v)), \forall xP(x), Q(u) \to \exists zQ(z)}}{P(f(v)), \forall xP(x), \exists yQ(y) \to \exists zQ(z)}}{P(f(v)), \forall xP(x), \exists yQ(y) \to P(f(v)) \land \exists zQ(z)}}{\dfrac{\forall xP(x), \exists yQ(y) \to P(f(v)) \land \exists zQ(z)}{\forall xP(x) \land \exists yQ(y) \to P(f(v)) \land \exists zQ(z)}}}{\to (\forall xP(x) \land \exists yQ(y)) \supset (P(f(v)) \land \exists zQ(z))}$$

where $\Pi = P(f(v)), \forall xP(x), \exists yQ(y) \to P(f(v))$.

5.4.3 Soundness of the System G

In order to establish the soundness of the system G, the following lemmas will be needed.

Lemma 5.4.1 (i) For any two terms t and r,

$$(r[t/x])_{\mathbf{M}}[s] = (r[\mathbf{a}/x])_{\mathbf{M}}[s], \text{ where } a = t_{\mathbf{M}}[s].$$

(ii) If t is free for x in A, then

$$(A[t/x])_{\mathbf{M}}[s] = (A[\mathbf{a}/x])_{\mathbf{M}}[s], \text{ where } a = t_{\mathbf{M}}[s].$$

Proof: The lemma is shown using the induction principle for terms and formulae as in lemma 5.3.1. The proof of (i) is left as an exercise. Since the proof of (ii) is quite similar to the proof of lemma 5.3.1, we only treat one case, leaving the others as an exercise.

Assume $A = \forall z B$. If $z = x$, then $A[t/x] = A$, and $A[\mathbf{a}/x] = A$. If $x \neq z$, since t is free for x in A, $z \notin FV(t)$, t is free for x in B,

$$A[t/x] = \forall z B[t/x] \text{ and } A[\mathbf{a}/x] = \forall z B[\mathbf{a}/x].$$

By problem 5.2.6, the following property holds: If t is free for x in B, $x \neq z$ and z is not in $FV(t)$, for every constant d,

$$B[t/x][d/z] = B[d/z][t/x].$$

Then, for every $c \in M$, by the induction hypothesis,

$$\begin{aligned}(B[t/x][\mathbf{c}/z])_{\mathbf{M}}[s] &= (B[\mathbf{c}/z][t/x])_{\mathbf{M}}[s] \\ = (B[\mathbf{c}/z][\mathbf{a}/x])_{\mathbf{M}}[s] &= (B[\mathbf{a}/x][\mathbf{c}/z])_{\mathbf{M}}[s],\end{aligned}$$

with $a = t_{\mathbf{M}}[s]$. But this proves that

$$(\forall z B[t/x])_{\mathbf{M}}[s] = (\forall z B[\mathbf{a}/x])_{\mathbf{M}}[s],$$

as desired. □

Lemma 5.4.2 For any rule stated in definition 5.4.1, the conclusion is falsifiable in some structure **M** if and only if one of the premises is falsifiable in the structure **M**. Equivalently, the conclusion is valid if and only if all premises are valid.

Proof: We prove the lemma for the rules $\forall : right$ and $\forall : left$, leaving the others as an exercise. In this proof, the following abbreviating conventions will be used: If Γ (or Δ) is the antecedent of a sequent,

$$\mathbf{M} \models \Gamma[s]$$

means that

$$\mathbf{M} \models A[s]$$

for every formula $A \in \Gamma$, and if Δ (or Λ) is the succedent of a sequent,

$$\mathbf{M} \not\models \Delta[s]$$

means that

$$\mathbf{M} \not\models B[s]$$

for every formula $B \in \Delta$.

(i) Assume that $\Gamma \to \Delta, A[y/x], \Lambda$ is falsifiable. This means that there is a structure $\mathbf{M}$ and an assignment s such that

$$\mathbf{M} \models \Gamma[s],\ \mathbf{M} \not\models \Delta[s],$$
$$\mathbf{M} \not\models (A[y/x])[s] \text{ and } \mathbf{M} \not\models \Lambda[s].$$

Recall that

$$\mathbf{M} \models (\forall x A)[s]$$

iff

$$\mathbf{M} \models (A[\mathbf{a}/x])[s] \text{ for every } a \in M.$$

Since y is free for x in A, by lemma 5.4.1, for $a = s(y)$, we have

$$(A[y/x])_{\mathbf{M}}[s] = (A[\mathbf{a}/x])_{\mathbf{M}}[s].$$

Hence,

$$\mathbf{M} \not\models (\forall x A)[s],$$

which implies that $\Gamma \to \Delta, \forall x A, \Lambda$ is falsified by $\mathbf{M}$ and s.

Conversely, assume that $\mathbf{M}$ and s falsify $\Gamma \to \Delta, \forall x A, \Lambda$. In particular, there is some $a \in M$ such that

$$\mathbf{M} \not\models (A[\mathbf{a}/x])[s].$$

Let s' be the assignment $s(y := a)$. Since $y \notin FV(A) - \{x\}$ and y is free for x in A, by lemma 5.3.3 and lemma 5.4.1, then

$$\mathbf{M} \not\models (A[y/x])[s'].$$

Since y does not appear free in Γ, Δ or Λ, by lemma 5.3.3, we also have

$$\mathbf{M} \models \Gamma[s'],\ \mathbf{M} \not\models \Delta[s'] \text{ and } \mathbf{M} \not\models \Lambda[s'].$$

Hence, $\mathbf{M}$ and s' falsify $\Gamma \to \Delta, A[y/x], \Lambda$.

(ii) Assume that $\mathbf{M}$ and s falsify $\Gamma, A[t/x], \forall x A, \Delta \to \Lambda$. Then, it is clear that $\mathbf{M}$ and s also falsify $\Gamma, \forall x A, \Delta \to \Lambda$.

Conversely, assume that $\mathbf{M}$ and s falsify $\Gamma, \forall x A, \Delta \to \Lambda$. Then,

$$\mathbf{M} \models \Gamma[s],\ \mathbf{M} \models (\forall x A)[s],$$
$$\mathbf{M} \models \Delta[s] \text{ and } \mathbf{M} \not\models \Lambda[s].$$

In particular,

$$\mathbf{M} \models (A[\mathbf{a}/x])[s] \text{ for every } a \in M.$$

By lemma 5.4.1, for $a = t_{\mathbf{M}}[s]$, we have

$$(A[t/x])_{\mathbf{M}}[s] = (A[\mathbf{a}/x])_{\mathbf{M}}[s],$$

and so

$$\mathbf{M} \models (A[t/x])[s].$$

Hence, $\mathbf{M}$ and s also falsify $\Gamma, A[t/x], \forall xA, \Delta \rightarrow \Lambda$. $\square$

As a consequence, we obtain the soundness of the system G.

Lemma 5.4.3 (Soundness of G) Every sequent provable in G is valid.

Proof: Use the induction principle for G-proofs and the fact that the axioms are obviously valid. $\square$

We shall now turn to the completeness of the system G. For that, we shall modify the procedure *expand*. The additional complexity comes from the quantifier rules and the handling of terms, and we need to revise the definition of a Hintikka set. We shall define the concept of a Hintikka set with respect to a term algebra H defined in the next section. First, we extend the notation for signed formulae given in definition 3.5.3 to quantified formulae.

5.4.4 Signed Formulae and Term Algebras (no Equality Symbol)

We begin with signed formulae.

Definition 5.4.3 *Signed formulae of type a*, *type b*, *type c*, and *type d* and their *components* are defined in the tables below.

Type-a formulae

A	A_1	A_2
$T(X \wedge Y)$	TX	TY
$F(X \vee Y)$	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
$F(X \wedge Y)$	FX	FY
$T(X \vee Y)$	TX	TY
$T(X \supset Y)$	FX	TY

Type-c formulae

C	C_1	C_2
$T\forall xY$	$TY[t/x]$	$TY[t/x]$
$F\exists xY$	$FY[t/x]$	$FY[t/x]$

where t is free for x in Y

Type-d formulae

D	D_1	D_2
$T\exists xY$	$TY[t/x]$	$TY[t/x]$
$F\forall xY$	$FY[t/x]$	$FY[t/x]$

where t is free for x in Y

For a formula C of type c, we let $C(t)$ denote $TY[t/x]$ if $C = T\forall xY$, $FY[t/x]$ if $C = F\exists xY$, and for a formula D of type d, we let $D(t)$ denote $TY[t/x]$ if $D = T\exists xY$, $FY[t/x]$ if $D = F\forall xY$.

We define satisfaction for signed formulae as follows.

Definition 5.4.4 Given a structure $\mathbf{M}$, an assignment s, and a formula A,

$$\mathbf{M} \models (TA)[s] \quad \text{iff} \quad \mathbf{M} \models A[s],$$

and

$$\mathbf{M} \models (FA)[s] \quad \text{iff} \quad \mathbf{M} \models (\neg A)[s]$$

(or equivalently, $\mathbf{M} \not\models A[s]$).

Lemma 5.4.4 For any structure $\mathbf{M}$ and any assignment s:

(i) For any formula C of type c,

$$\mathbf{M} \models C[s] \quad \text{iff} \quad \mathbf{M} \models C(\mathbf{a})[s] \text{ for every } a \in M;$$

(ii) For any formula D of type d,

$$\mathbf{M} \models D[s] \quad \text{iff} \quad \mathbf{M} \models D(\mathbf{a})[s] \text{ for at least one } a \in M.$$

Proof: The lemma follows immediately from lemma 5.3.1. □

In view of lemma 5.4.4, formulae of type c are also called formulae of *universal type*, and formulae of type d are called formulae of *existential type*. In order to define Hintikka sets with respect to a term algebra H, some definitions are needed.

Definition 5.4.5 Given a first-order language **L**, a nonempty set H of terms in $TERM_{\mathbf{L}}$ (with variables from **V**) is a *term algebra* iff:

(i) For every n-ary function symbol f in **L** ($n > 0$), for any terms $t_1, ..., t_n \in H$, $ft_1...t_n$ is also in H.

(ii) Every constant symbol c in **L** is also in H.

Note that this definition is consistent with the definition of an algebra given in Section 2.4. Indeed, every function symbol and every constant in **L** receives an interpretation. A term algebra is simply an algebra whose carrier H is a nonempty set of terms, and whose operations are the *term constructors*. Note also that the terms in the carrier H may contain variables from the countable set **V**. However, variables are not in **L** and are not treated as constants. Also, observe that if **L** has at least one function symbol, by condition (i) any term algebra on **L** is infinite. Hence, a term algebra is finite only if **L** does not contain any function symbols.

5.4.5 Reducts, Expansions

Given a set S of signed **L**-formulae, not all symbols in **L** need occur in formulae in S. Hence, we shall define the *reduct* of **L** to S.

Definition 5.4.6 (i) Given two first-order languages (with or without equality) **L** and **L**$'$, if **L** is a subset of **L**$'$ (that is, the sets of constant symbols, function symbols, and predicate symbols of **L** are subsets of the corresponding sets of **L**$'$), we say that **L** is a *reduct* of **L**$'$ and that **L**$'$ is an *expansion* of **L**. (The set of variables is neither in **L** nor in **L**$'$ and is the given countable set **V**.)

(ii) If **L** is a reduct of **L**$'$, an **L**-structure $\mathbf{M} = (M, I)$ is a *reduct* of an **L**$'$-structure $\mathbf{M}' = (M', I')$ if $M' = M$ and I is the restriction of I' to **L**. $\mathbf{M}'$ is called an *expansion* of **M**.

(iii) Given a set S of signed **L**-formulae, the *reduct* of **L** with respect to S, denoted by $\mathbf{L}_S$, is the subset of **L** consisting of all constant, function, and predicate symbols occurring in formulae in S.

Note: If **L** is a reduct of **L**$'$, any **L**-structure **M** can be expanded to an **L**$'$-structure $\mathbf{M}'$ (in many ways). Furthermore, for any **L**-formula A and any assigment s,

$$\mathbf{M} \models A[s] \quad \text{if and only if} \quad \mathbf{M}' \models A[s].$$

5.4.6 Hintikka Sets (Languages Without Equality)

The definition of a Hintikka set is generalized to first-order languages without equality as follows.

Definition 5.4.7 A *Hintikka set* S (over a language $\mathbf{L}$ without equality) *with respect to a term algebra* H (over the reduct $\mathbf{L}_S$) is a set of signed $\mathbf{L}$-formulae such that the following conditions hold for all signed formulae A, B, C, D of type a, b, c, d:

H0: No atomic formula and its conjugate are both in S (TA or FA is atomic iff A is).

H1: If a type-a formula A is in S, then both A_1 and A_2 are in S.

H2: If a type-b formula B is in S, then either B_1 is in S or B_2 is in S.

H3: If a type-c formula C is in S, then for every term $t \in H$, $C(t)$ is in S (we require that t is free for x in C for every $t \in H$).

H4: If a type-d formula D is in S, then for at least one term $t \in H$, $D(t)$ is in S (we require that t is free for x in D for every $t \in H$).

H5: Every variable x occurring free in some formula of S is in H.

Observe that condition H5 and the fact that H is a term algebra imply that, for every term occurring in some formula in S, if that term is closed or contains only variables free in S, then it is in H.

We can now prove the generalization of lemma 3.5.3 for first-order logic (without equality). From lemma 5.3.4, we can assume without loss of generality that the set of variables occurring free in formulae in S is disjoint from the set of variables occurring bound in formulae in S.

Lemma 5.4.5 Every Hintikka set S (with respect to a term algebra H) is satisfiable in a structure $\mathbf{H_S}$ with domain H.

Proof: The $\mathbf{L}_S$-structure $\mathbf{H_S}$ is defined as follows. The domain of $\mathbf{H_S}$ is H.

Every constant c in $\mathbf{L}_S$ is interpreted as the term c;

Every function symbol f of rank n in $\mathbf{L}_S$ is interpreted as the function such that, for any terms $t_1, ..., t_n \in H$,

$$f_{\mathbf{H_S}}(t_1, ..., t_n) = ft_1...t_n.$$

For every predicate symbol P of rank n in $\mathbf{L}_S$, for any terms $t_1, ..., t_n \in H$,

$$P_{\mathbf{H_S}}(t_1, ..., t_n) = \begin{cases} \mathbf{F} & \text{if } FPt_1...t_n \in S, \\ \mathbf{T} & \text{if } TPt_1...t_n \in S, \\ & \text{or neither } TPt_1...t_n \text{ nor } FPt_1...t_n \text{ is in } S. \end{cases}$$

By conditions H0 and the fact that H is an algebra, this definition is proper. Let s be any assignment that is the identity on the variables belonging to H.

We now prove using the induction principle for formulae that

$$\mathbf{H_S} \models X[s]$$

for every signed formula $X \in S$. We will need the following claim that is proved using the induction principle for terms. (For a proof of a more general version of this claim, see claim 2 in lemma 5.6.1.)

Claim: For every term t (in H),

$$t_{\mathbf{H_S}}[s] = t.$$

Assume that $TPt_1...t_n$ is in S. By H5, the variables in t_1,..., t_n are in H, and since H is a term algebra, t_1,...,t_n are in H. By definition of $P_{\mathbf{H_S}}$ and the above claim,

$$\begin{aligned}(Pt_1...t_n)_{\mathbf{H_S}}[s] &= P_{\mathbf{H_S}}((t_1)_{\mathbf{H_S}}[s], ..., (t_n)_{\mathbf{H_S}}[s])\\ &= P_{\mathbf{H_S}}(t_1, ..., t_n) = \mathbf{T}.\end{aligned}$$

Hence,

$$\mathbf{H_S} \models Pt_1...t_n[s].$$

Similarly, it is shown that if $FPt_1...t_n$ is in S then

$$(Pt_1...t_n)_{\mathbf{H_S}}[s] = \mathbf{F}.$$

The propositional connectives are handled as in lemma 3.5.3.

If a signed formula C of type c is in S, $C(t)$ is in S for every term t in H by H3. Since $C(t)$ contains one less quantifier than C, by the induction hypothesis,

$$\mathbf{H_S} \models C(t)[s] \text{ for every } t \in H.$$

By lemma 5.4.1, for any formula A, any term t free for x in A, any structure $\mathbf{M}$ and any assignment v, we have

$$(A[t/x])_{\mathbf{M}}[v] = (A[\mathbf{a}/x])_{\mathbf{M}}[v],$$

where $a = t_{\mathbf{M}}[v]$. Since

$$t_{\mathbf{H_S}}[s] = t,$$

we have

$$(C[t/x])_{\mathbf{H_S}}[s] = (C[\mathbf{t}/x])_{\mathbf{H_S}}[s].$$

Hence,

$$\mathbf{H_S} \models C(\mathbf{t})[s] \text{ for every } t \in H,$$

which implies that

$$\mathbf{H_S} \models C[s]$$

by lemma 5.4.4.

If a signed formula D of type d is in S, $D(t)$ is in S for some t in H by H4. Since $D(t)$ contains one less quantifier than D, by the induction hypothesis,

$$\mathbf{H_S} \models D(t)[s].$$

As above, it can be shown that

$$\mathbf{H_S} \models D(\mathbf{t})[s] \text{ for some } t \in H,$$

which implies that

$$\mathbf{H_S} \models D[s]$$

(by lemma 5.4.4). Finally, using the note before definition 5.4.7, $\mathbf{H_S}$ can be expanded to an $\mathbf{L}$-structure satisfying S. $\square$

The domain H of the structure $\mathbf{H_S}$ is also called a *Herbrand universe*.

In order to prove the completeness of the system G (in case of a first-order language $\mathbf{L}$ without equality) we shall extend the methods used in sections 3.4 and 3.5 for the propositional calculus to first-order logic. Given a (possibly infinite) sequent $\Gamma \to \Delta$, our goal is to attempt to falsify it. For this, we design a *search* procedure with the following properties:

(1) If the original sequent is valid, the *search* procedure stops after a finite number of steps, yielding a proof tree.

(2) If the original sequent is falsifiable, the *search* procedure constructs a possibly infinite tree, and along some (possibly infinite) path in the tree it can be shown that a Hintikka set exists, which yields a counter example for the sequent.

The problem is to modify the *expand* procedure to deal with quantifier rules and terms. Clauses H3 and H4 in the definition of a Hintikka set suggest that a careful procedure must be designed in dealing with quantified formulae.

We first treat the *special case* in which we are dealing with a first-order language without equality, without function symbols and with a finite sequent $\Gamma \to \Delta$.

5.4.7 Completeness: Special Case of Languages Without Function Symbols and Without Equality

Given a sequent $\Gamma \to \Delta$, using lemma 5.3.4, we can assume that the set of all variables occurring free in some formula in the sequent is disjoint from the set of all variables occurring bound in some formula in the sequent. This condition ensures that terms occurring in formulae in the sequent are free for susbtitutions. Even though it is not strictly necessary to assume that all the formulae in the sequent are rectified, it is convenient to assume that they are.

First, it is convenient for proving the correctness of the *search* procedure to give a slightly more general version of the quantifier rules $\forall : left$ and $\exists : right$.

Definition 5.4.8 The *extended rules* $\forall : left$ and $\exists : right$ are the following:

$$\frac{\Gamma, A[t_1/x], ..., A[t_k/x], \forall xA, \Delta \to \Lambda}{\Gamma, \forall xA, \Delta \to \Lambda} \; (\forall : left)$$

$$\frac{\Gamma \to \Delta, A[t_1/x], ..., A[t_k/x], \exists xA, \Lambda}{\Gamma \to \Delta, \exists xA, \Lambda} \; (\exists : right)$$

where t_1,...,t_k are any k terms ($k \geq 1$) free for x in A.

It is clear that an inference using this new version of the $\forall : left$ rule (resp. $\exists : right$ rule) can be simulated by k applications of the old $\forall : left$ rule (resp. $\exists : right$ rule). Hence, there is no gain of generality. However, these new rules may reduce the size of proof trees. Consequently, we will assume from now on that the rules of definition 5.4.8 are used as the $\forall : left$ and $\exists : right$ rules of the Gentzen system G.

In order to fulfill conditions H3 and H4 we build lists of variables and constants as follows.

Let $TERM_0$ be a nonempty list of terms and variables defined as follows. If no free variables and no constants occur in any of the formulae in $\Gamma \to \Delta$,

$$TERM_0 =< y_0 >,$$

where y_0 is the first variable in $\mathbf{V}$ not occurring in any formula in $\Gamma \to \Delta$. Otherwise,

$$TERM_0 =< u_0, ..., u_p >,$$

a list of all free variables and constants occurring in formulae in $\Gamma \to \Delta$. Let

$$AVAIL_0 =< y_1, ..., y_n, ... >$$

be a countably infinite list disjoint from $TERM_0$ and consisting of variables not occurring in any formula in $\Gamma \to \Delta$.

The terms in $TERM_0$ and the variables in $AVAIL_0$ will be used as the t's and y's for our applications of quantifier rules $\forall : right$, $\forall : left$, $\exists : right$ and $\exists : left$. Because the variables in $TERM_0$ do not occur bound in $\Gamma \to \Delta$ and the variables in $AVAIL_0$ are new, the substitutions with results $A[t/x]$ and $A[y/x]$ performed using the quantifier rules will be free.

As the search for a counter example for $\Gamma \to \Delta$ progresses, we keep track step by step of which of u_0,...,u_p,y_1,y_2,y_3,... have been thus far activated. The list of activated terms is kept in $TERM_0$ and the list of available variables in $AVAIL_0$.

Every time a rule $\forall : right$ or $\exists : left$ is applied, as the variable y we use the head of the list $AVAIL_0$, we append y to the end of $TERM_0$ and delete y from the head of $AVAIL_0$.

When a rule $\forall : left$ or $\exists : right$ is applied, we use as the terms t each of the terms u_0,...,u_q in $TERM_0$ that have not previously served as a term t for that rule with the same principal formula.

To handle the $\forall : left$ rule and the $\exists : right$ rule correctly, it is necessary to keep track of the formulae $\forall xA$ (or $\exists xA$) for which the term u_i was used as a term for the rule $\forall : left$ (or $\exists : right$) with principal formula $\forall xA$ (or $\exists xA$). The first reason is economy, but the second is more crucial:

If a sequent has the property that all formulae in it are either atomic or of the form $\forall xA$ (or $\exists xA$) such that all the terms u_0,...,u_q in $TERM_0$ have already been used as terms for the rule $\forall : left$ (or $\exists : right$), and if this sequent is not an axiom, then it will never become an axiom and we can stop expanding it.

Hence, we structure $TERM_0$ as a list of records where every record $< u_i, FORM_0(i) >$ contains two fields: u_i is a term and $FORM_0(i)$ a list of the formulae $\forall xA$ (or $\exists xA$) for which u_i was used as a term t for the rule $\forall : left$ (or $\exists : right$) with principal formula $\forall xA$ (or $\exists xA$). Initially, each list $FORM_0(i)$ is the null list. The lists $FORM_0(i)$ are updated each time a term t is used in a rule $\forall : left$ or $\exists : right$. We also let $t(TERM_0)$ denote the set of terms $\{u_i \mid < u_i, FORM_0(i) > \in TERM_0\}$.

Finally, we need to take care of another technical detail: These lists must be updated only at the end of a round, so that the same substitutions are performed for all occurrences of a formula. Hence, we create another variable $TERM_1$ local to the procedure *search*. During a round, $TERM_1$ is updated but $TERM_0$ is not, and at the end of the round, $TERM_0$ is set to its updated version $TERM_1$.

A leaf of the tree constructed by procedure *search* is *finished* iff either:

(1) The sequent labeling it is an axiom, or

(2) The sequent contains only atoms or formulae $\forall xA$ (or $\exists xA$) belonging to all of the lists $FORM_0(i)$ for all $< u_i, FORM_0(i) >$ in $TERM_0$.

The *search* procedure is obtained by modifying the procedure given in definition 3.4.6 by adding the initialization of $TERM_0$ and $AVAIL_0$.

Definition 5.4.9 The *search* procedure. The input to *search* is a one-node tree labeled with a sequent $\Gamma \rightarrow \Delta$. The output is a possibly infinite tree T called a *systematic deduction tree*.

Procedure Search

```
procedure search($\Gamma \rightarrow \Delta$ : sequent; var $T$ : tree);
  begin
    let $T$ be the one-node tree labeled with $\Gamma \rightarrow \Delta$;
    Let $TERM_0 :=<< u_0, nil >, ..., < u_p, nil >>$
    and let $AVAIL_0 :=< y_1, y_2, y_3, ... >$, with $u_0, ..., u_p$
    as explained after def. 5.4.8, with $p = 1$ and $u_0 = y_0$ when
    $\Gamma \rightarrow \Delta$ contains no free variables and no constants.
    while not all leaves of $T$ are finished do
      $TERM_1 := TERM_0$; $T_0 := T$;
      for each leaf node of $T_0$
      (in lexicographic order of tree addresses) do
        if not finished(node) then
          expand(node, T)
        endif
      endfor;
      $TERM_0 := TERM_1$
    endwhile;
    if all leaves are closed
    then
      write ('T is a proof of $\Gamma \rightarrow \Delta$')
    else
      write ('$\Gamma \rightarrow \Delta$ is falsifiable')
    endif
  end
```

Procedure Expand

```
    procedure expand(node : tree-address; var $T$ : tree);
      begin
        let $A_1, ..., A_m \rightarrow B_1, ..., B_n$ be the label of node;
        let $S$ be the one-node tree labeled with
          $A_1, ..., A_m \rightarrow B_1, ..., B_n$;
        for $i := 1$ to $m$ do
          if nonatomic($A_i$) then
            grow-left($A_i$, S)
          endif
        endfor;
        for $i := 1$ to $n$ do
          if nonatomic($B_i$) then
            grow-right($B_i$, S)
          endif
        endfor;
        $T :=$ dosubstitution($T$, node, $S$)
      end
```

Procedure Grow-Left

```
procedure grow-left($A : formula$; var $S : tree$);
  begin
    case $A$ of
    $B \wedge C$, $B \vee C$,
    $B \supset C$, $\neg B$  : extend every nonaxiom leaf of $S$ using the
                    left rule corresponding to the main
                    propositional connective;
    $\forall xB$          : for every term $u_k \in t(TERM_0)$
                    such that $A$ is not in $FORM_0(k)$ do
                      extend every nonaxiom leaf of $S$ by applying
                      the $\forall : left$ rule using the term $u_k$
                      as one of the terms of the rule. In $TERM_1$
                      let $FORM_1(k) := append(FORM_1(k), A)$
                    endfor;
    $\exists xB$          : extend every nonaxiom leaf of $S$ by applying
                    the $\exists : left$ rule using $y = head(AVAIL_0)$
                    as the new variable;
                    $TERM_1 := append(TERM_1, < y, nil >)$;
                    $AVAIL_0 := tail(AVAIL_0)$
    endcase
  end
```

Procedure Grow-Right

```
procedure grow-right($A : formula$; var $S : tree$);
  begin
    case $A$ of
    $B \wedge C$, $B \vee C$,
    $B \supset C$, $\neg B$  : extend every nonaxiom leaf of $S$ using the
                    right rule corresponding to the main
                    propositional connective;
    $\exists xB$          : for every term $u_k \in t(TERM_0)$
                    such that $A$ is not in $FORM_0(k)$ do
                      extend every nonaxiom leaf of $S$ by applying
                      the $\exists : right$ rule using the term $u_k$
                      as one of the terms of the rule. In $TERM_1$
                      let $FORM_1(k) := append(FORM_1(k), A)$
                    endfor;
    $\forall xB$          : extend every nonaxiom leaf of $S$ by applying
                    the $\forall : right$ rule using $y = head(AVAIL_0)$
                    as the new variable;
                    $TERM_1 := append(TERM_1, < y, nil >)$;
                    $AVAIL_0 := tail(AVAIL_0)$
    endcase
  end
```

EXAMPLE 5.4.3

Let

$$A = \exists x(P \supset Q(x)) \supset (P \supset \exists z Q(z)),$$

where P is a propositional symbol and Q a unary predicate symbol. Let us trace the construction of the proof tree constructed by the *search* procedure. Since A does not have any free variables, $TERM_0 =<< y_0, nil >>$. After the first round, we have the following tree, and $TERM_0$ and $AVAIL_0$ have not changed.

$$\frac{\exists x(P \supset Q(x)) \to P \supset \exists z Q(z)}{\to \exists x(P \supset Q(x)) \supset (P \supset \exists z Q(z))}$$

The $\exists : left$ rule is applied using the head y_1 of $AVAIL_0$, and the following tree is obtained:

$$\dfrac{\dfrac{\dfrac{P, P \supset Q(y_1) \to \exists z Q(z)}{P \supset Q(y_1) \to P \supset \exists z Q(z)}}{\exists x(P \supset Q(x)) \to P \supset \exists z Q(z)}}{\to \exists x(P \supset Q(x)) \supset (P \supset \exists z Q(z))}$$

At the end of this round,

$$TERM_0 =<< y_0, nil >, < y_1, nil >>,$$

and

$$AVAIL_0 =< y_2, y_3, ... > .$$

During the next round, the $\exists : right$ rule is applied to the formula $\exists z Q(z)$ with the terms y_0 and y_1. The following tree is obtained:

$$\dfrac{\dfrac{\dfrac{P \to P, \exists z Q(z) \qquad \dfrac{Q(y_1), P \to Q(y_0), Q(y_1), \exists z Q(z)}{Q(y_1), P \to \exists z Q(z)}}{P, P \supset Q(y_1) \to \exists z Q(z)}}{\dfrac{P \supset Q(y_1) \to P \supset \exists z Q(z)}{\exists x(P \supset Q(x)) \to P \supset \exists z Q(z)}}}{\to \exists x(P \supset Q(x)) \supset (P \supset \exists z Q(z))}$$

At the end of the round,

$$TERM_0 =<< y_0, < \exists z Q(z) >>, < y_1, < \exists z Q(z) >>> .$$

This last tree is a proof tree for A.

EXAMPLE 5.4.4

Let

$$A = \exists x(P \supset Q(x)) \supset (P \supset \forall z Q(z)),$$

where P is a propositional letter and Q is a unary predicate symbol. Initially, $TERM_0 =<< y_0, nil >>$. After the first round, the following tree is obtained and $TERM_0$ and $AVAIL_0$ are unchanged:

$$\dfrac{\exists x(P \supset Q(x)) \to P \supset \forall z Q(z)}{\to \exists x(P \supset Q(x)) \supset (P \supset \forall z Q(z))}$$

During the second round, the $\exists : left$ rule is applied and the following tree is obtained:

$$\dfrac{\dfrac{\dfrac{P, P \supset Q(y_1) \to \forall z Q(z)}{P \supset Q(y_1) \to P \supset \forall z Q(z)}}{\exists x(P \supset Q(x)) \to P \supset \forall z Q(z)}}{\to \exists x(P \supset Q(x)) \supset (P \supset \forall z Q(z))}$$

At the end of this round,

$$TERM_0 =<< y_0, nil >, < y_1, nil >>,$$

and

$$AVAIL_0 =< y_2, y_3, ... > .$$

During the next round, the $\forall$: $right$ rule is applied to the formula $\forall z Q(z)$ with the new variable y_2. The following tree is obtained:

$$\dfrac{\dfrac{\dfrac{\dfrac{P \to P, \forall z Q(z) \qquad \dfrac{Q(y_1), P \to Q(y_2)}{Q(y_1), P \to \forall z Q(z)}}{P, P \supset Q(y_1) \to \forall z Q(z)}}{P \supset Q(y_1) \to P \supset \forall z Q(z)}}{\exists x(P \supset Q(x)) \to P \supset \forall z Q(z)}}{\to \exists x(P \supset Q(x)) \supset (P \supset \forall z Q(z))}$$

At the end of this round,

$$TERM_0 =<< y_0, nil >, < y_1, nil >, < y_2, nil >>,$$

and

$$AVAIL_0 =< y_3, y_4, ... > .$$

Since all formulae in the sequent $Q(y_1), P \rightarrow Q(y_2)$ are atomic and it is not an axiom, this last tree yields a counter example with domain $\{y_0, y_1, y_2\}$, by making Q arbitrarily **true** for y_0 and y_1, **false** for y_2, and P **true**. Note that $\{y_1, y_2\}$ is also the domain of a counter example.

The following theorem is a generalization of theorem 3.4.1 to finite sequents in first-order languages without function symbols (and without equality). Recall that a closed tree is finite by definition.

Theorem 5.4.1 Let $\Gamma \rightarrow \Delta$ be an input sequent in which no free variable occurs bound. (i) If $\Gamma \rightarrow \Delta$ is valid, then the procedure *search* halts with a closed tree T which is a proof tree for $\Gamma \rightarrow \Delta$.

(ii) If $\Gamma \rightarrow \Delta$ is falsifiable, then either *search* halts with a finite counter-example tree T and $\Gamma \rightarrow \Delta$ can be falsified in a finite structure, or *search* generates an infinite tree T and $\Gamma \rightarrow \Delta$ can be falsified in a countably infinite structure.

Proof: First, assume that the sequent $\Gamma \rightarrow \Delta$ is falsifiable. If the tree T was closed, by lemma 5.4.3, $\Gamma \rightarrow \Delta$ would be valid, a contradiction. Hence, either T is finite and contains some path to a nonaxiom leaf, or T is infinite and by König's lemma contains an infinite path. In either case, we show as in theorem 3.5.1 that a Hintikka set can be found along that path. Let U be the union of all formulae occurring in the left-hand side of each sequent along that path, and V the union of all formulae occurring in the right-hand side of any such sequent. Let

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

We prove the following claim:

Claim: S is a Hintikka set with respect to the term algebra consisting of the set H of terms in $t(TERM_0)$.

Conditions H0, H1, and H2 are proved as in the propositional case (proof of lemma 3.5.2), and we only have to check that $t(TERM_0)$ is a term algebra, and conditions H3, H4, and H5. Since **L** does not contain function symbols, $t(TERM_0)$ is trivially closed under the operations (there are none). Since the constants occurring in the input sequent are put in $t(TERM_0)$, $t(TERM_0)$ is a term algebra. Since $t(TERM_0)$ is initialized with the list of free variables and constants occurring in the input sequent (or y_0 if this list is empty), and since every time a variable y is removed from the head of $AVAIL_0$, $< y, < nil >>$ is added to $TERM_0$, H5 holds. Every time a formula C of type c is expanded, all substitution instances $C(t)$ for all t in $t(TERM_0)$ that have not already been used with C are added to the upper sequent. Hence, H3 is satisfied. Every time a formula D of type d is expanded, y is added to $t(TERM_0)$ and the substitution instance $D(y)$ is added to the upper sequent. Hence, H4 is satisfied, and the claim holds. $\square$

By lemma 5.4.5, some assignment s satisfies S in the structure $\mathbf{H_S}$. This implies that $\Gamma \rightarrow \Delta$ is falsified by $\mathbf{H_S}$ and s. Note that H must be infinite if the tree T is infinite. Otherwise, since we start with a finite sequent, every path would be finite and would end either with an axiom or a finished sequent.

If the sequent $\Gamma \rightarrow \Delta$ is valid the tree T must be finite and closed since otherwise, the above argument shows that $\Gamma \rightarrow \Delta$ is falsifiable. $\square$

As a corollary, we obtain a version of Gödel's completeness theorem for first-order languages without function symbols or equality.

Corollary If a sequent in which no variable occurs both free and bound (over a first-order language without function symbols or equality) is valid then it is G-provable. $\square$

In Section 5.5, we shall modify the *search* procedure to handle function symbols and possibly infinite sequents. Finally in Section 5.6, we will adapt the procedure to deal with languages with equality.

PROBLEMS

5.4.1. Give proof trees for the following formulae:

$$\forall x A \supset A[t/x], \quad A[t/x] \supset \exists x A,$$
$$\text{where } t \text{ is free for } x \text{ in } A.$$

5.4.2. Let x, y be any distinct variables. Let A be any formula, C any formula not containing the variable x free, and let E be any formula such that x is free for y in E. Give proof trees for the following formulae:

$$\begin{array}{cc}
\forall x C \equiv C & \exists x C \equiv C \\
\forall x \forall y A \equiv \forall y \forall x A & \exists x \exists y A \equiv \exists y \exists x A \\
\forall x \forall y E \supset \forall x E[x/y] & \exists x E[x/y] \supset \exists x \exists y E \\
\forall x A \supset \exists x A & \\
\exists x \forall y A \supset \forall y \exists x A &
\end{array}$$

5.4.3. Let A, B be any formulae, and C any formula not containing the variable x free. Give proof trees for the following formulae:

$$\begin{array}{cc}
\neg \exists x A \equiv \forall x \neg A & \neg \forall x A \equiv \exists x \neg A \\
\exists x A \equiv \neg \forall x \neg A & \forall x A \equiv \neg \exists x \neg A \\
\forall x A \land \forall x B \equiv \forall x (A \land B) & \exists x A \lor \exists x B \equiv \exists x (A \lor B) \\
C \land \forall x A \equiv \forall x (C \land A) & C \lor \exists x A \equiv \exists x (C \lor A) \\
C \land \exists x A \equiv \exists x (C \land A) & C \lor \forall x A \equiv \forall x (C \lor A) \\
\exists x (A \land B) \supset \exists x A \land \exists x B & \forall x A \lor \forall x B \supset \forall x (A \lor B)
\end{array}$$

5.4.4. Let A, B be any formulae, and C any formula not containing the variable x free. Give proof trees for the following formulae:

$$\begin{array}{ll}
(C \supset \forall xA) \equiv \forall x(C \supset A) & (C \supset \exists xA) \equiv \exists x(C \supset A) \\
(\forall xA \supset C) \equiv \exists x(A \supset C) & (\exists xA \supset C) \equiv \forall x(A \supset C) \\
(\forall xA \supset \exists xB) \equiv \exists x(A \supset B) & \\
(\exists xA \supset \forall xB) \supset \forall x(A \supset B) &
\end{array}$$

5.4.5. Give a proof tree for the following formula:

$$\neg\forall x\exists y(\neg P(x) \land P(y)).$$

5.4.6. Show that the rules $\forall : right$ and $\exists : left$ are not necessarily sound if y occurs free in the lower sequent.

$*$ **5.4.7.** Let **L** be a first-order language without function symbols and without equality. A first-order formula A is called *simple* if, for every subformula of the form $\forall xB$ or $\exists xB$, B is quantifier free.

Prove that the *search* procedure always terminates for simple formulae. Conclude that there is an algorithm for deciding validity of simple formulae.

5.4.8. Let $\Gamma \to A$ be a finite sequent, let x be any variable occurring free in $\Gamma \to A$, and let $\Gamma[c/x] \to A[c/x]$ be the result of substituting any constant c for x in all formulae in Γ and in A.

Prove that if $\Gamma[c/x] \to A[c/x]$ is provable, then $\Gamma[y/x] \to A[y/x]$ is provable for every variable y not occurring in Γ or in A. Conclude that if x does not occur in Γ, if $\Gamma \to A[c/x]$ is provable, then $\Gamma \to \forall xA$ is provable.

$*$ **5.4.9.** The language of a binary relation (simple directed graphs) consists of one binary predicate symbol R. The axiom for simple directed graphs is:

$$\forall x\forall y(R(x,y) \supset \neg R(y,x)).$$

This expresses that a simple directed graph has no cycles of length ≤ 2. Let T_1 contain the above axiom together with the axiom

$$\forall x\exists yR(x,y)$$

asserting that every node has an outgoing edge.

(a) Find a model for T_1 having three elements.

Let T_2 be T_1 plus the density axiom:

$$\forall x\forall y(R(x,y) \supset \exists z(R(x,z) \land R(z,y))).$$

(b) Find a model for T_2. Find a seven-node graph model of T_2.

5.5 Completeness for Languages with Function Symbols and no Equality

In order to deal with (countably) infinite sequents, we shall use the technique for building a tree used in Section 3.5. First, we need to deal with function symbols.

5.5.1 Organizing the Terms for Languages with Function Symbols and no Equality

Function symbols are handled in the following way. First, we can assume without loss of generality that the set of variables $\mathbf{V}$ is the disjoint union of two countably infinite sets $\{x_0, x_1, x_2, ...\}$ and $\{y_0, y_1, y_2, ...\}$ and that only variables in the first set (the x's) are used to build formulae. In this way, $\{y_0, y_1, y_2, ...\}$ is an infinite supply of new variables. Let $\Gamma \rightarrow \Delta$ be a possibly infinite sequent. As before, using lemma 5.3.4, we can assume that the set of variables occurring free in $\Gamma \rightarrow \Delta$ is disjoint from the set of variables occurring bound in $\Gamma \rightarrow \Delta$.

Let $\mathbf{L}'$ be the reduct of $\mathbf{L}$ consisting of the constant, function and predicate symbols occurring in formulae in $\Gamma \rightarrow \Delta$. If the set of constants and free variables occurring in $\Gamma \rightarrow \Delta$ is nonempty, let

$$TERMS =< u_0, u_1, ..., u_k, ... >$$

be an enumeration of all $\mathbf{L}'$-terms constructed from the variables occurring free in formulae in $\Gamma \rightarrow \Delta$, and the constant and function symbols in $\mathbf{L}'$. Otherwise, let

$$TERMS =< y_0, u_1, ..., u_k, ... >$$

be an enumeration of all the terms constructed from the variable y_0 and the function symbols in $\mathbf{L}'$.

Let

$$TERM_0 =<< head(TERMS), nil >>$$

and let

$$AVAIL_0 = tail(TERMS).$$

For any $i \geq 1$, let $AVAIL_i$ be an enumeration of the set of all $\mathbf{L}'$-terms *actually containing* some occurrence of y_i and constructed from the variables occurring free in formulae in $\Gamma \rightarrow \Delta$, the constant and function symbols occurring in $\Gamma \rightarrow \Delta$, and the variables y_1,...,y_i. We assume that such an enumeration begins with y_i and is of the form

$$AVAIL_i \ = < y_i, u_{i,1}, ..., u_{i,j}, .. > .$$

EXAMPLE 5.5.1

Assume that $\mathbf{L}'$ contains a constant symbol a, and the function symbols f of rank 2, and g of rank 1. Then, a possible enumeration of $TERMS$ is the following:

$$TERMS =< a, g(a), f(a,a), g(g(a)), g(f(a,a)), f(g(a),a),$$
$$f(a, g(a)), f(g(a), g(a)), ... > .$$

Hence, $TERM_0 =<< a, nil >>$, and

$$AVAIL_0 =< g(a), f(a,a), g(g(a)), g(f(a,a)),$$
$$f(g(a),a), f(a, g(a)), f(g(a), g(a)), ... > .$$

For $i = 1$, a possible enumeration of $AVAIL_1$ is:

$$AVAIL_1 =< y_1, g(y_1), f(a, y_1), f(y_1, a), f(y_1, y_1),$$
$$g(g(y_1)), g(f(a, y_1)), g(f(y_1, a)), g(f(y_1, y_1)), ... > .$$

Each time a rule $\forall : right$ or $\exists : left$ is applied, we use as the variable y the first y_i of y_1,y_2,... not yet activated, append y_i to $t(TERM_0)$, and delete y_i from the list $AVAIL_i$. We use a counter $NUMACT$ to record the number of variables y_1,y_2,... thus far activated. Initially $NUMACT = 0$, and every time a new y_i is activated $NUMACT$ is incremented by 1.

In a $\forall : left$ or a $\exists : right$ step, all the terms in $t(TERM_0)$ thus far activated are available as terms t. Furthermore, at the end of every round, the head of every available list $AVAIL_i$ thus far activated (with $0 \leq i \leq NUMACT$) is appended to $t(TERM_0)$ and deleted from $AVAIL_i$. Thus, along any path in the tree that does not close, once any term in an $AVAIL_i$ list is activated, every term in the list is eventually activated.

Note that if the sets **FS** and **CS** are effective (recursive), recursive functions enumerating the lists $t(TERM_0)$ and $AVAIL_i$ $(i \geq 0)$ can be written (these lists are in fact recursive).

The definition of a *finished leaf* is the following: A leaf of the tree is finished if either

(1) It is an axiom, or

(2) L and R (as defined in subsection 3.5.2) are empty, and the sequent only contains atoms or formulae $\forall xA$ (or $\exists xA$) belonging to all lists $FORM_0(i)$, for all $< u_i, FORM_0(i) >$ in $TERM_0$.

5.5.2 The Search Procedure for Languages with Function Symbols and no Equality

The *search* procedure (and its subprograms) are revised as follows.

Definition 5.5.1 The *search* procedure.

Procedure Search

```
procedure $search(\Gamma_0 \rightarrow \Delta_0 : sequent;$ var $T : tree);$
  begin
    $L := tail(\Gamma_0); \Gamma := head(\Gamma_0);$
    $R := tail(\Delta_0); \Delta := head(\Delta_0);$
    let $T$ be the one-node tree labeled with $\Gamma \rightarrow \Delta$;
    Let $TERM_0$ and $AVAIL_i$, $0 \leq i$,
    be initialized as explained in section 5.5.1.
    $NUMACT := 0;$
    while not all leaves of $T$ are finished do
      $TERM_1 := TERM_0; T_0 := T;$
      for each leaf node of $T_0$
      (in lexicographic order of tree addresses) do
        if not $finished(node)$ then
          $expand(node, T)$
        endif
      endfor;
      $TERM_0 := TERM_1; L := tail(L); R := tail(R);$
      for $i := 0$ to $NUMACT$ do
        $TERM_0 := append(TERM_0, < head(AVAIL_i), nil >);$
        $AVAIL_i := tail(AVAIL_i)$
      endfor
    endwhile;
    if all leaves are closed
    then
      write ('$T$ is a proof of $\Gamma_0 \rightarrow \Delta_0$')
    else
      write ('$\Gamma_0 \rightarrow \Delta_0$ is falsifiable')
    endif
  end
```

The Procedures *expand*, *grow-left*, and *grow-right* appear on the next two pages.

```
procedure expand(node : tree-address; var T : tree);
  begin
    let $A_1, ..., A_m \rightarrow B_1, ..., B_n$ be the label of node;
    let S be the one-node tree labeled with
      $A_1, ..., A_m \rightarrow B_1, ..., B_n$;
      for $i := 1$ to $m$ do
        if nonatomic($A_i$) then
          grow-left($A_i$, S)
        endif
      endfor;
      for $i := 1$ to $n$ do
        if nonatomic($B_i$) then
          grow-right($B_i$, S)
        endif
      endfor;
      for each leaf u of S do
        let $\Gamma \rightarrow \Delta$ be the label of u;
        $\Gamma' := \Gamma, head(L)$;
        $\Delta' := \Delta, head(R)$;
        create a new node u1 labeled with $\Gamma' \rightarrow \Delta'$
      endfor; $T := dosubstitution(T, node, S)$
  end

procedure grow-left(A : formula; var S : tree);
  begin
    case A of
    $B \wedge C$, $B \vee C$,
    $B \supset C$, $\neg B$   : extend every nonaxiom leaf of S using the
                     left rule corresponding to the main
                     propositional connective;
    $\forall xB$         : for every term $u_k \in t(TERM_0)$
                     such that A is not in $FORM_0(k)$ do
                       extend every nonaxiom leaf of S by applying
                       the $\forall : left$ rule using the term $u_k$
                       as one of the terms of the rule. In $TERM_1$
                       let $FORM_1(k) := append(FORM_1(k), A)$
                     endfor;
    $\exists xB$         : $NUMACT := NUMACT + 1$;
                     extend every nonaxiom leaf of S by applying
                     the $\exists : left$ rule using $y = head(AVAIL_{NUMACT})$
                     as the new variable;
                     $TERM_1 := append(TERM_1, < y, nil >)$;
                     $AVAIL_{NUMACT} := tail(AVAIL_{NUMACT})$
    endcase
  end
```

```
procedure grow-right(A : formula; var S : tree);
  begin
    case A of
    $B \land C$, $B \lor C$,
    $B \supset C$, $\neg B$ : extend every nonaxiom leaf of S using the
                 right rule corresponding to the main
                 propositional connective;
    $\exists xB$      : for every term $u_k \in t(TERM_0)$
                 such that A is not in $FORM_0(k)$ do
                   extend every nonaxiom leaf of S by applying
                   the $\exists$ : right rule using the term $u_k$
                   as one of the terms of the rule. In $TERM_1$
                   let $FORM_1(k) := append(FORM_1(k), A)$
                 endfor;
    $\forall xB$      : $NUMACT := NUMACT + 1$;
                 extend every nonaxiom leaf of S by applying
                 the $\forall$ : right rule using $y = head(AVAIL_{NUMACT})$
                 as the new variable;
                 $TERM_1 := append(TERM_1, < y, nil >)$;
                 $AVAIL_{NUMACT} := tail(AVAIL_{NUMACT})$
    endcase
  end
```

Let us give an example illustrating this new version of the *search* procedure.

EXAMPLE 5.5.2

Let

$$A = (\forall xP(x) \land \exists yQ(y)) \supset (P(f(v)) \land \exists zQ(z)),$$

where P and Q are unary predicate symbols, and f is a unary function symbol. The variable v is free in A. Initially,

$$TERM_0 =<< v, nil >>,$$
$$AVAIL_0 =< f(v), f(f(v)), ..., f^n(v), ... >,$$

and for $i \geq 1$,

$$AVAIL_i =< y_i, f(y_i), ..., f^n(y_i), ... > .$$

After the first round, we have the following tree:

$$\frac{\forall xP(x) \land \exists yQ(y) \to P(f(v)) \land \exists zQ(z)}{\to \forall xP(x) \land \exists yQ(y) \supset P(f(v)) \land \exists zQ(z)}$$

At the end of this round,

$$TERM_0 =<< v, nil >, < f(v), nil >>,$$

$$AVAIL_0 =< f^2(v), ..., f^n(v), ... >,$$

and for $i \geq 1$, $AVAIL_i$ is unchanged. After the second round, we have the following tree:

$$\dfrac{\dfrac{\dfrac{\forall xP(x), \exists yQ(y) \to P(f(v)) \qquad \forall xP(x), \exists yQ(y) \to \exists zQ(z)}{\forall xP(x), \exists yQ(y) \to P(f(v)) \land \exists zQ(z)}}{\forall xP(x) \land \exists yQ(y) \to P(f(v)) \land \exists zQ(z)}}{\to \forall xP(x) \land \exists yQ(y) \supset P(f(v)) \land \exists zQ(z)}$$

At the end of this round,

$$TERM_0 =<< v, nil >, < f(v), nil >, < f(f(v)), nil >>,$$

$$AVAIL_0 =< f^3(v), ..., f^n(v), ... >,$$

and for $i \geq 1$, $AVAIL_i$ is unchanged. After the third round, we have the following tree:

$$\dfrac{\dfrac{\dfrac{\dfrac{T_1}{\forall xP(x), \exists yQ(y) \to P(f(v))} \qquad \dfrac{T_2}{\forall xP(x), \exists yQ(y) \to \exists zQ(z)}}{\forall xP(x), \exists yQ(y) \to P(f(v)) \land \exists zQ(z)}}{\forall xP(x) \land \exists yQ(y) \to P(f(v)) \land \exists zQ(z)}}{\to \forall xP(x) \land \exists yQ(y) \supset P(f(v)) \land \exists zQ(z)}$$

where the tree T_1 is the proof tree

$$\dfrac{P(v), P(f(v)), P(f(f(v))), \forall xP(x), \exists yQ(y) \to P(f(v))}{\forall xP(x), \exists yQ(y) \to P(f(v))}$$

and the tree T_2 is the tree

$$\dfrac{\dfrac{\dfrac{\Gamma, \forall xP(x), Q(y_1) \to Q(v), Q(f(v)), Q(f(f(v))), \exists zQ(z)}{P(v), P(f(v)), P(f(f(v))), \forall xP(x), Q(y_1) \to \exists zQ(z)}}{P(v), P(f(v)), P(f(f(v))), \forall xP(x), \exists yQ(y) \to \exists zQ(z)}}{\forall xP(x), \exists yQ(y) \to \exists zQ(z)}$$

where

$$\Gamma = P(v), P(f(v)), P(f(f(v))).$$

At the end of this round,

$$TERM_0 =<< v, < \forall x P(x), \exists z Q(z) >>, < f(v), < \forall x P(x), \exists z Q(z) >>, \\ < f(f(v)), < \forall x P(x), \exists z Q(z) >>, < y_1, nil >, < f^3(v), nil >, \\ < f(y_1), nil >>,$$

$$AVAIL_0 =< f^4(v), ..., f^n(v), ... >,$$

$$AVAIL_1 =< f^2(y_1), ..., f^n(y_1), ... >,$$

and for $i > 1$, $AVAIL_i$ is unchanged. At the end of the fourth round, T_2 expands into the following proof tree:

$$\cfrac{\cfrac{\cfrac{\cfrac{\Gamma', Q(y_1) \rightarrow Q(v), Q(f(v)), Q(f(f(v))), Q(y_1), Q(f^3(v)), Q(f(y_1)), \exists z Q(z)}{\Gamma, \forall x P(x), Q(y_1) \rightarrow Q(v), Q(f(v)), Q(f(f(v))), \exists z Q(z)}}{P(v), P(f(v)), P(f(f(v))), \forall x P(x), Q(y_1) \rightarrow \exists z Q(z)}}{P(v), P(f(v)), P(f(f(v))), \forall x P(x), \exists y Q(y) \rightarrow \exists z Q(z)}}{\forall x P(x), \exists y Q(y) \rightarrow \exists z Q(z)}$$

where

$$\Gamma' = P(v), P(f(v)), P(f(f(v))), Q(y_1), Q(f^3(v)), Q(f(y_1)), \forall x P(x).$$

EXAMPLE 5.5.3

Let

$$A = (\exists x P(x) \land Q(a)) \supset \forall y P(f(y)).$$

Initially, $TERM_0 =<< a, nil >>$, and

$$AVAIL_0 =< f(a), f^2(a), ..., f^n(a), ... > .$$

At the end of the first round, the following tree is obtained:

$$\cfrac{\exists x P(x) \land Q(a) \rightarrow \forall y P(f(y))}{\rightarrow \exists x P(x) \land Q(a) \supset \forall y P(f(y))}$$

At the end of the round,

$$TERM_0 =<< a, nil >, < f(a), nil >>,$$

and

$$AVAIL_0 =< f^2(a), ..., f^n(a), ... > .$$

At the end of the second round, the following tree is obtained:

$$\cfrac{\cfrac{\exists xP(x), Q(a) \rightarrow P(f(y_1))}{\exists xP(x) \land Q(a) \rightarrow \forall yP(f(y))}}{\rightarrow \exists xP(x) \land Q(a) \supset \forall yP(f(y))}$$

At the end of this round,

$$\begin{gathered}TERM_0 =<< a, nil >, < f(a), nil >, < y_1, nil >, < f(y_1), nil >,\\ < f^2(a), nil >>,\\ AVAIL_0 =< f^3(a), ..., f^n(a), ... >, \text{ and}\\ AVAIL_1 =< f^2(y_1), ..., f^n(y_1), ... > .\end{gathered}$$

At the end of the third round, the following tree is obtained:

$$\cfrac{\cfrac{\cfrac{P(y_2), Q(a) \rightarrow P(f(y_1))}{\exists xP(x), Q(a) \rightarrow P(f(y_1))}}{\exists xP(x) \land Q(a) \rightarrow \forall yP(f(y))}}{\rightarrow \exists xP(x) \land Q(a) \supset \forall yP(f(y))}$$

This tree is a finished nonclosed tree, since all formulae in the top sequent are atomic. A counter example is given by the structure having

$$\begin{gathered}H =< a, f(a), ..., f^n(a), ..., y_1, f(y_1), ..., f^n(y_1), ...,\\ y_2, f(y_2), ..., f^n(y_2), ... >\end{gathered}$$

as its domain, and by interpreting P as taking the value **T** for y_2, **F** for $f(y_1)$, and Q taking the value **T** for a.

5.5.3 Completeness of the System G (Languages Without Equality)

We can now prove the following fundamental theorem.

Theorem 5.5.1 Let $\Gamma_0 \rightarrow \Delta_0$ be an input sequent in which no free variable occurs bound. (i) If $\Gamma_0 \rightarrow \Delta_0$ is valid, then the procedure *search* halts with a closed tree T, from which a proof tree for a finite subsequent $C_1, ..., C_m \rightarrow D_1, ..., D_n$ of $\Gamma_0 \rightarrow \Delta_0$ can be constructed.

(ii) If $\Gamma_0 \rightarrow \Delta_0$ is falsifiable, then there is a Hintikka set S such that either

(a) Procedure *search* halts with a finite counter-example tree T and, if we let

$$H_S = t(TERM_0) \cup \{AVAIL_i \mid 0 \leq i \leq NUMACT\},$$

$\Gamma_0 \rightarrow \Delta_0$ is falsifiable in a structure with countable domain H_S; or

(b) Procedure *search* generates an infinite tree T and, if we let $H_S = t(TERM_0)$, then $\Gamma_0 \rightarrow \Delta_0$ is falsifiable in a structure with countable domain H_S.

Proof: The proof combines techniques from the proof of theorem 5.4.1 and the proof of theorem 3.5.1. The difference with the proof of theorem 5.4.1 is that a closed tree T is not exactly a proof tree, and that it is necessary to modify the tree T in order to obtain a proof tree.

Assume that *search* produces a closed tree, and let $C_1, ..., C_m$ be the initial subsequence of formulae in Γ which were deleted from Γ to obtain L, and $D_1, ..., D_n$ the initial subsequence of formulae in Δ which were deleted from Δ to obtain R. A proof tree for a the finite sequent $C_1, ..., C_m \rightarrow D_1, ..., D_n$ can easily be obtained from T, using the following technique:

First, starting from the root and proceeding bottom-up, for each node $\Gamma, head(L) \rightarrow \Delta, head(R)$ at depth k created at the end of a call to procedure *expand*, add $head(L)$ after the rightmost formula in the premise of every sequent at depth less than k, and add $head(R)$ after the rightmost formula in the conclusion of every sequent at depth less than k, obtaining the tree T'. Then, a proof tree T'' for $C_1, ..., C_m \rightarrow D_1, ..., D_n$ 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''.

If T is not a closed tree, along a finite or infinite path we obtain a set S of signed formulae as in the proof of theorem 5.4.1. We show the following claim:

Claim: The set H_S of terms defined in clause (ii) of theorem 5.5.1 is a term algebra (over the reduct $\mathbf{L}'$), and S is a Hintikka set with respect to H_S.

To prove the claim, as in theorem 5.4.1, we only need to prove that H_S is a term algebra, and that H3, H4, and H5 hold. We can assume that $\mathbf{L}'$ contains function symbols since the other case has been covered by theorem 5.4.1. Since $\mathbf{L}'$ contains function symbols, all the sets $AVAIL_i$ $(i \geq 0)$ are countably infinite.

First, observe that every variable free in S is either a variable free in the input sequent or one of the activated variables. Since

(i) $t(TERM_0)$ and $AVAIL_0$ are initialized in such a way that the variables free in the input sequent belong to the union of $t(TERM_0)$ and $AVAIL_0$,

(ii) Whenever a new variable y_i is removed from the head of the list $AVAIL_i$ it is added to $t(TERM_0)$ and,

(iii) At the end of every round the head of every activated list (of the form $AVAIL_i$, for $0 \leq i \leq NUMACT$) is removed from that list and added to $t(TERM_0)$, it follows from (i)-(iii) that all variables free in S are in $t(TERM_0) \cup \{AVAIL_i \mid 0 \leq i \leq NUMACT\}$ if T is finite or in $t(TERM_0)$ if T is infinite, and condition H5 holds.

Observe that if T is finite, from (i), (ii), and (iii) it also follows that $t(TERM_0) \cup \{AVAIL_i \mid 0 \leq i \leq NUMACT\}$ contains the set of all terms built up from the variables free in the input sequent, the variables activated during applications of $\exists : left$ or $\forall : right$ rules, and the constant and function symbols occurring in the input sequent. Hence, H_S is closed under the function symbols in $\mathbf{L}'$, and it is a term algebra.

If T is infinite, all the terms in all the activated lists $AVAIL_i$ are eventually transferred to $t(TERM_0)$, and conditions (i) to (iii) also imply that $t(TERM_0)$ contains the set of all terms built up from the variables free in the input sequent, the variables activated during applications of $\exists : left$ or $\forall : right$ rules, and the constant and function symbols occurring in the input sequent. Hence, $H_S = t(TERM_0)$ is a term algebra.

If T is infinite, condition H5 also follows from (i) to (iii).

Every time a formula C of type c is expanded, all substitution instances $C(t)$ for all t in $t(TERM_0)$ which have not already been used with C are added to the upper sequent. Hence, H3 is satisfied. Every time a formula D of type d is expanded, the new variable y_i removed from $AVAIL_i$ is added to $t(TERM_0)$ and the substitution instance $D(y_i)$ is added to the upper sequent. Hence, H4 is satisfied, and the claim holds.

Since S is a Hintikka set, by lemma 5.4.5, some assignment s satisfies S in a structure $\mathbf{H_S}$ with domain H_S, and so, $\Gamma_0 \rightarrow \Delta_0$ is falsified by $\mathbf{H_S}$ and s. $\square$

Corollary (A version of Gödel's extended completeness theorem for G) A sequent in which no variable occurs both free and bound (even infinite) is valid iff it is G-provable. $\square$

As a second corollary, we obtain the following useful result.

Corollary There is an algorithm for deciding whether a finite sequent consisting of quantifier-free formulae is valid.

Proof: Observe that for quantifier-free formulae, the *search* procedure never uses the quantifier rules. Hence, it behaves exactly as in the propositional case, and the result follows from theorem 3.4.1. $\square$

Unfortunately, this second corollary does not hold for all formulae. Indeed, it can be shown that there is no algorithm for deciding whether any given first-order formula is valid. This is known as *Church's theorem*. A proof of Church's theorem can be found in Enderton, 1972, or Kleene, 1952. A particularly concise and elegant proof due to Floyd is also given in Manna, 1974.

The completeness theorem only provides a semidecision procedure, in the sense that if a formula is valid, this can be demonstrated in a finite number of steps, but if it is falsifiable, the procedure may run forever.

Even though the *search* procedure provides a rather natural proof procedure which is theoretically complete, in practice it is extremely inefficient in terms of the number of steps and the amount of memory needed.

This is illustated by the very simple formulae of example 5.5.2 and example 5.5.3 for which it is already laborious to apply the *search* procedure. The main difficulty is the proper choice of terms in applications of $\forall : left$ and $\exists : right$ rules. In particular, note that the ordering of the terms in the lists $t(TERM_0)$ and $AVAIL_i$ can have a drastic influence on the length of proofs.

After having tried the *search* procedure on several examples, the following fact emerges: It is highly desirable to perform all the quantifier rules first, in order to work as soon as possible on quantifier-free formulae. Indeed, by the second corollary to the completeness theorem, proving quantifier-free formulae is a purely mechanical process.

It will be shown in the next chapter that provided that the formulae in the input sequent have a certain form, if such a sequent is provable, then it has a proof in which all the quantifier inferences are performed below all the propositional inferences. This fact will be formally established by Gentzen's sharpened Hauptsatz, proved in Chapter 7. Hence, the process of finding a proof can be viewed as a two-step procedure:

(1) In the first step, one attempts to "guess" the right terms used in $\forall : left$ and $\exists : right$ inferences;

(2) In the second step, one checks that the quantifier-free formula obtained in step 1 is a tautology.

A rigorous justification of this method will be given by Herbrand's theorem, proved in Chapter 7. The resolution method for first-order logic presented in Chapter 8 can be viewed as an improvement of the above method.

For the time being, we consider some applications of theorem 5.5.1. The following theorems are easily obtained as consequences of the main theorem.

5.5.4 Löwenheim-Skolem, Compactness, and Model Existence Theorems for Languages Without Equality

The following result known as Löwenheim-Skolem's theorem is often used in model theory.

Theorem 5.5.2 (Löwenheim-Skolem) If a set of formulae Γ is satisfiable in some structure $\mathbf{M}$, then it is satisfiable in a structure whose domain is at most countably infinite.

Proof: It is clear that $\Gamma \rightarrow$ is falsifiable in $\mathbf{M}$. By theorem 5.5.1, the *search* procedure yields a tree from which a Hintikka set S can be obtained. But the domain H of $\mathbf{H_S}$ is at most countably infinite since it consists of terms built from countable sets. Hence, $\mathbf{H_S}$ is a countable structure in which Γ is satisfiable. $\square$

The other results obtained as consequences of theorem 5.5.1 are counterparts of theorem 3.5.3, theorem 3.5.4, and lemma 3.5.4. The proofs are similar and use structures instead of valuations.

Theorem 5.5.3 (Compactness theorem) For any (possibly countably infinite) set Γ of formulae, if every nonempty finite subset of Γ is satisfiable then Γ is satisfiable.

Proof: Similar to that of theorem 3.5.3. $\square$

Recall that a set Γ of formulae is *consistent* if there exists some formula A such that $C_1, ..., C_m \rightarrow A$ is *not* G-provable for any $C_1, ..., C_m$ in Γ.

Theorem 5.5.4 (Model existence theorem) If a set Γ of formulae is consistent then it is satisfiable.

Proof: Similar to that of theorem 3.5.4. $\square$

Note that the *search* procedure will actually yield a structure $\mathbf{H_S}$ for each Hintikka set S arising along each nonclosed path in the tree T, in which S and Γ are satisfiable.

Lemma 5.5.1 (Consistency lemma) If a set Γ of formulae is satisfiable then it is consistent.

Proof: Similar to that of lemma 3.5.4. $\square$

5.5.5 Maximal Consistent Sets

The concept of a maximal consistent set introduced in definition 3.5.9 is generalized directly to sets of first-order formulae:

A consistent set Γ of first-order formulae (without equality) 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. Lemma 3.5.5 can be easily generalized to the first-order case.

Lemma 5.5.2 Given a first-order language (without equality), every consistent set Γ is a subset of some maximal consistent set Δ.

Proof: Almost identical to that of lemma 3.5.5, but using structures instead of valuations. $\square$

It is possible as indicated in Section 3.5 for propositional logic to prove lemma 5.5.2 directly, and use lemma 5.5.2 to prove the model existence theorem (theorem 5.5.4). From the model existence theorem, the extended Gödel completeness theorem can be shown (corollary to theorem 5.5.1). Using this approach, it is necessary to use a device due to Henkin known as *adding witnesses*. Roughly speaking, this is necessary to show that a Henkin maximal consistent set is a Hintikka set with respect to a certain term algebra. The addition of witnesses is necessary to ensure condition H4 of Hintikka sets. Such an approach is explored in the problems and for details, we refer the reader to Enderton, 1972; Chang and Keisler, 1973; or Van Dalen, 1980.

PROBLEMS

5.5.1. Using the *search* procedure, find counter examples for the formulae:

$$\exists xA \land \exists xB \supset \exists x(A \land B)$$
$$\forall y\exists xA \supset \exists x\forall yA$$
$$\exists xA \supset \forall xA$$

5.5.2. Using the *search* procedure, prove that the following formula is valid:

$$\neg\exists y\forall x(S(y,x) \equiv \neg S(x,x))$$

5.5.3. Using the *search* procedure, prove that the following formulae are valid:

$$\forall xA \supset A[t/x],$$
$$A[t/x] \supset \exists xA,$$
$$\text{where } t \text{ is free for } x \text{ in } A.$$

5.5.4. This problem is a generalization of the Hilbert system H of problem 3.4.9 to first-order logic. For simplicity, we first treat the case of first-order languages without equality. The *Hilbert system* H for first-order logic (without equality) over the language using the connectives, $\land, \lor, \supset, \neg, \forall$ and $\exists$ is defined as follows:

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

$$A \supset (B \supset A)$$
$$(A \supset B) \supset ((A \supset (B \supset C)) \supset (A \supset C))$$

$$A \supset (B \supset (A \wedge B))$$
$$A \supset (A \vee B), \qquad B \supset (A \vee B)$$
$$(A \supset B) \supset ((A \supset \neg B) \supset \neg A)$$
$$(A \wedge B) \supset A, \qquad (A \wedge B) \supset B$$
$$(A \supset C) \supset ((B \supset C) \supset ((A \vee B) \supset C))$$
$$\neg\neg A \supset A$$
$$\forall x A \supset A[t/x]$$
$$A[t/x] \supset \exists x A$$

where in the last two axioms, t is free for x in A.

There are three inference rules:

(i) The rule *modus ponens* given by:

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

(ii) The *generalization rules* given by:

$$\frac{(B \supset A)}{(B \supset \forall x A)}\ \forall : rule \qquad \frac{(A \supset B)}{(\exists x A \supset B)}\ \exists : rule$$

where x does not occur free in B.

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

(i) Every one-node tree labeled with an axiom B or a formula 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 $\{A_1, ..., A_m\}$:

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

(iii) If T_1 is a deduction tree of $(B \supset A)$ from $\{A_1, ..., A_m\}$ (or a deduction tree of $(A \supset B)$ from $\{A_1, ..., A_m\}$), and x does not occur free in any of the formulae in $\{A_1, ..., A_m\}$, then the following trees are deduction trees of $(B \supset \forall x A)$ from $\{A_1, ..., A_m\}$ (or $(\exists x A \supset B)$ from $\{A_1, ..., A_m\}$).

$$\dfrac{\dfrac{T_1}{(B \supset A)}}{(B \supset \forall x A)} \qquad\qquad \dfrac{\dfrac{T_1}{(A \supset B)}}{(\exists x A \supset B)}$$

A *proof tree* is a deduction tree whose leaves are labeled with axioms. Given a set $\{A_1, ..., A_m\}$ of formulae and a formula B, we use the notation $A_1, ..., A_m \vdash B$ to denote that there is a deduction tree 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 $\vdash B$.

Prove that the generalization rules are sound rules, in the sense that if the premise is valid, then the conclusion is valid. Prove that the system H is sound; that is, every provable formula is valid.

5.5.5. (i) Show that if $A_1, ..., A_m, A \vdash B$ is a deduction in H not using the generalization rules, then $A_1, ..., A_m \vdash (A \supset B)$.

(ii) Check that the following is a deduction of C from $A \supset (B \supset C)$ and $A \land B$:

$$\dfrac{\dfrac{A \land B \qquad A \land B \supset B}{B} \qquad \dfrac{\dfrac{A \land B \qquad A \land B \supset A}{A} \qquad A \supset (B \supset C)}{B \supset C}}{C}$$

Conclude that $A \supset (B \supset C) \vdash (A \land B) \supset C$.

(iii) Check that the following is a deduction of C from $(A \land B) \supset C$, A, and B:

$$\dfrac{\dfrac{B \qquad \dfrac{A \qquad A \supset (B \supset (A \land B))}{B \supset (A \land B)}}{A \land B} \qquad (A \land B) \supset C}{C}$$

Conclude that $(A \land B) \supset C \vdash A \supset (B \supset C)$.

$*$ **5.5.6.** In this problem, we are also considering the proof system H of problem 5.5.4. The *deduction theorem* states that, for arbitrary formulae 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. In the case of the generalization rules, use problem 5.5.5.

5.5.7. Prove the following:

$$\text{If } A_1, ..., A_m \vdash B_i \text{ for every } i,\ 1 \leq i \leq m \text{ and}$$
$$B_1, ..., B_m \vdash C,\ \text{then } A_1, ..., A_m \vdash C.$$

Hint: Use the deduction theorem.

5.5.8. Prove that for any set Γ of formulae, and any two formulae B and C,

$$\text{if } \vdash C \quad \text{then}$$
$$\Gamma, C \vdash B \quad \text{iff} \quad \Gamma \vdash B.$$

$*$ **5.5.9.** In this problem, we are still considering the proof system H of problem 5.5.4. Prove that the following meta rules hold about deductions in the system H:

For all formulae A, B, C and finite sequence Γ of formulae (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)
$\forall$	If $\Gamma \vdash A$ then $\Gamma \vdash \forall x A$	$\forall x A \vdash A[t/x]$
$\exists$	$A[t/x] \vdash \exists x A$	If $\Gamma, A \vdash C$ then $\Gamma, \exists x A \vdash C$

where t is free for x in A, x does not occur free in Γ, and x does not occur free in C.

Hint: Use problem 5.5.7, problem 5.5.8, and the deduction theorem.

$*$ **5.5.10.** 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 formula, there is a proof in the system H.

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

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

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

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

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

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

if $\vdash A_1, ..., A_m \to B_1, ..., B_n$ 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 5.5.9.

5.5.11. Recall that the cut rule is the 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.

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 structure **A**, if **A** 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.

5.5.12. (i) Prove solely in terms of proofs in $G + \{cut\}$ that a set Γ of formulae is inconsistent if and only if there is a formula A such that both $\Gamma \to A$ and $\Gamma \to \neg A$ are provable in $G + \{cut\}$.

(ii) Prove solely in terms of proofs in $G + \{cut\}$ that, $\Gamma \to A$ is not provable in $G + \{cut\}$ if and only if $\Gamma \cup \{\neg A\}$ is consistent.

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 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.)

5.5.13. A set Γ of sentences is said to be *complete* if, for every sentence A, either

$$\vdash \Gamma \rightarrow A \quad \text{or}$$
$$\vdash \Gamma \rightarrow \neg A,$$

but not both. Prove that for any set Γ of *sentences*, the following are equivalent:

(i) The set $\{A \mid \ \vdash \Gamma \rightarrow A \text{ in G}\}$ is a maximal consistent set.

(ii) Γ is complete.

(iii) Any two models of Γ are elementary equivalent.

(See problem 5.3.25 for the definition of *elementary equivalence*.)

5.5.14. Let Γ be a consistent set (of **L**-formulae). Let $A_1, A_2, ..., A_n$,... be an enumeration of *all* **L**-formulae. 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.

∗ **5.5.15.** Prove that the results of problem 3.5.16 hold for first-order logic if we change the word *proposition* to *sentence*.

∗ **5.5.16.** Prove that the results of problem 3.5.17 hold for first-order logic if we change the word *proposition* to *sentence* and work in $G + \{cut\}$.

∗ **5.5.17.** In this problem and the next four, Henkin's version of the completeness theorem is worked out. The approach is to prove the model existence theorem, and derive the completeness theorem as a consequence. To prove that every consistent set S is satisfiable, first problem 5.5.14 is used to extend S to a maximal consistent set. However, such a maximal consistent set is not necessarily a Hintikka set, because condition H4 may not be satisfied. To overcome this problem, we use Henkin's method, which consists of adding to S formulae of the form $\exists x B \supset B[c/x]$, where c is a new constant called a *witness* of $\exists x B$. However, we have to iterate this process to obtain the desired property.

Technically, we proceed as follows. Let $\mathbf{L}$ be a first-order language (without equality). Let $\mathbf{L}^*$ be the extension of $\mathbf{L}$ obtained by adding the set of new constants

$$\{c_D \mid D \text{ is any sentence of the form } \exists xB\}.$$

The constant c_D is called a *witness* of D. Let S be any set of $\mathbf{L}$-sentences, and let

$$S^* = S \cup \{\exists xB \supset B[c_D/x] \mid \exists xB \in FORM_{\mathbf{L}},$$
$$c_D \text{ is a witness of the sentence } D = \exists xB\}.$$

(Note that $\exists xB$ is any *arbitrary* existential sentence which need not belong to S).

(a) Prove that for every $\mathbf{L}$-sentence A, if $S^* \to A$ is provable in $G + \{cut\}$, then $S \to A$ is provable in $G + \{cut\}$.

Hint: Let Γ be a finite subset of S^* such that $\Gamma \to A$ is provable (in $G + \{cut\}$). Assume that Γ contains some sentence of the form $\exists xB \supset B[c_D/x]$, where $D = \exists xB$. Let $\Delta = \Gamma - \{\exists xB \supset B[c_D/x]\}$. Note that c_D does not occur in Δ, $\exists xB$, or A. Using the result of problem 5.4.8, show that $\Delta, \exists xB \supset B[y/x] \to A$ is provable (in $G + \{cut\}$), where y is a new variable not occurring in $\Gamma \to A$. Next, show that $\to \exists x(\exists xB \supset B)$ is provable (in G). Then, show (using a cut) that $\Delta \to A$ is provable (in $G + \{cut\}$). Conclude by induction on the number of sentences of the form $\exists xB \supset B[c_D/x]$ in Γ.

(b) A set S of $\mathbf{L}$-sentences is a *theory* iff it is closed under provability, that is, iff

$$S = \{A \mid \vdash S \to A \text{ in } G + \{cut\}, FV(A) = \emptyset\}.$$

A theory S is a *Henkin theory* iff for any sentence D of the form $\exists xB$ in $FORM_{\mathbf{L}}$ (*not necessarily* in S), there is a constant c in $\mathbf{L}$ such that $\exists xB \supset B[c/x]$ is also in S.

Let S be any set of $\mathbf{L}$-sentences and let T be the theory

$$T = \{A \mid \vdash S \to A \text{ in } G + \{cut\}, FV(A) = \emptyset\}.$$

Define the sequence of languages $\mathbf{L}_n$ and the sequence of theories T_n as follows:

$$\mathbf{L}_0 = \mathbf{L};\ \mathbf{L}_{n+1} = (\mathbf{L}_n)^*;\ \text{and}$$
$$T_0 = T;\ T_{n+1} = \{A \mid \vdash (T_n)^* \to A \text{ in } G + \{cut\}, FV(A) = \emptyset\}.$$

Let

$$\mathbf{L}^H = \bigcup_{n\geq 0} \mathbf{L}_n,$$

and

$$T^H = \bigcup_{n\geq 0} T_n.$$

Prove that T^H is a Henkin theory over the language $\mathbf{L}^H$.

(c) Prove that for every $\mathbf{L}$-sentence C,

$$\begin{aligned} &\text{if } \vdash T^H \rightarrow C \text{ in } G + \{cut\},\\ &\text{then } \vdash T \rightarrow C \text{ in } G + \{cut\}. \end{aligned}$$

We say that T^H is *conservative* over T.

(d) Prove that T^H is consistent iff T is.

$*$ **5.5.18.** Let T be a consistent set of $\mathbf{L}$-sentences.

(a) Show that there exists a maximal consistent extension T' over $\mathbf{L}^H$ of T which is also a Henkin theory.

Hint: Let $S = \{A \mid\ \vdash T \rightarrow A, FV(A) = \emptyset\}$. Show that any maximally consistent extension of S^H is also a Henkin theory which is maximally consistent.

(b) The definition of formulae of type a, b, c, d for unsigned formulae and of their *immediate descendants* given in problem 3.5.7 is extended to the first-order case.

Formulae of types a, b, c, d and their *immediate descendants* are defined 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

Type-c formulae

C	C_1	C_2
$\forall xY$	$Y[t/x]$	$Y[t/x]$
$\neg\exists xY$	$\neg Y[t/x]$	$\neg Y[t/x]$

where t is free for x in Y

Type-d formulae

D	D_1	D_2
$\exists xY$	$Y[t/x]$	$Y[t/x]$
$\neg\forall xY$	$\neg Y[t/x]$	$\neg Y[t/x]$

where t is free for x in Y

The definition of a Hintikka set is also adapted in the obvious way to unsigned formulae. Also, in this problem and some of the following problems, given any set S of formulae and any formulae A_1,...,A_n, the set $S \cup \{A_1, ...A_n\}$ will also be denoted by $\{S, A_1, ..., A_n\}$.

Prove that a maximal consistent Henkin set T' of sentences is a Hintikka set with respect to the term algebra H consisting of all closed terms built up from the function and constant symbols in $\mathbf{L}^H$.

(c) Prove that every consistent set T of $\mathbf{L}$-sentences is satisfiable.

(d) Prove that a formula A with free variables $\{x_1, ..., x_n\}$ is satisfiable iff $A[c_1/x_1, ..., c_n/x_n]$ is satisfiable, where c_1,...,c_n are new constants. Using this fact, prove that every consistent set T of $\mathbf{L}$-formulae is satisfiable.

$*$ **5.5.19.** Recall that from problem 5.5.12, in $G + \{cut\}$, $\Gamma \rightarrow A$ is not provable if and only if $\Gamma \cup \{\neg A\}$ is consistent. Use the above fact and problem 5.5.18 to give an alternate proof of the extended completeness theorem for $G + \{cut\}$ ($\models \Gamma \rightarrow A$ iff $\vdash \Gamma \rightarrow A$ in $G + \{cut\}$).

$**$ **5.5.20.** Prove that the results of problem 5.5.17 hold for languages of any cardinality. Using Zorn's lemma, prove that the results of problem 5.5.18 hold for languages of any cardinality. Thus, prove that the extended completeness theorem holds for languages of any cardinality.

$*$ **5.5.21.** For a language $\mathbf{L}$ of cardinality α, show that the cardinality of the term algebra arising in problem 5.5.18 is at most α. Conclude that any consistent set of $\mathbf{L}$-sentences has a model of cardinality at most α.

$*$ **5.5.22.** Let $\mathbf{L}$ be a countable first-order language without equality. A property P of sets of formulae is an *analytic consistency property* iff the following hold:

(i) P is of finite character (see problem 3.5.12).

(ii) For every set S of formulae for which P is true, the following conditions hold:

A_0: S contains no atomic formula and its negation.

A_1: For every formula A of type a in S, P holds for $\{S, A_1\}$ and $\{S, A_2\}$.

A_2: For every formula B of type b in S, either P holds for $\{S, B_1\}$ or P holds for $\{S, B_2\}$.

A_3: For every formula C of type c in S, P holds for $\{S, C(t)\}$ for every term t.

A_4: For every formula D of type d in S, for some constant c not occurring in S, P holds for $\{S, D(c)\}$.

Prove that if P is an analytic consistency property and P holds for S, then S is satisfiable.

Hint: The *search* procedure can be adapted to work with sets of formulae rather than sequents, by identifying each sequent $\Gamma \to \Delta$ with the set of formulae $\Gamma \cup \{\neg B \mid B \in \Delta\}$, and using the rules corresponding to the definition of descendants given in problem 5.5.18. Using the *search* procedure applied to the set S (that is, to the sequent $S \to$), show that at any stage of the construction of a deduction tree, there is a path such that P holds for the union of S and the union of the formulae along that path. Also, the constant c plays the same role as a new variable.

$*$ **5.5.23.** Let **L** be a countable first-order language without equality. A set S of formulae is *truth-functionally inconsistent* iff S is the result of substituting first-order formulae for the propositional letters in an unsatisfiable set of propositions (see definition 5.3.11). We say that a formula A is *truth-functionally valid* iff it is obtained by substitution of first-order formulae into a tautology (see definition 5.3.11). We say that a finite set $S = \{A_1, ..., A_m\}$ *truth-functionally implies* B iff the formula $(A_1 \wedge ... \wedge A_m) \supset B$ is truth-functionally valid. A property P of sets of formulae is a *synthetic consistency property* iff the following conditions hold:

(i) P is of finite character (see problem 3.5.12).

(ii) For every set S of formulae, the following conditions hold:

B_0: If S is truth-functionally inconsistent, then P does not hold for S;

B_3: If P holds for S then for every formula C of type c in S, P holds for $\{S, C(t)\}$ for every term t.

B_4: For every formula D of type d in S, for some constant c not occurring in S, P holds for $\{S, D(c)\}$.

B_5: For every formula X, if P does not hold for $\{S, X\}$ or $\{S, \neg X\}$, then P does not hold for S. Equivalently, if P holds for S, then for every formula X, either P holds for $\{S, X\}$ or P holds for $\{S, \neg X\}$

(a) Prove that if P is a synthetic consistency property then the following condition holds:

B_6: If P holds for S and a finite subset of S truth-functionally implies X, then P holds for $\{S, X\}$.

(b) Prove that every synthetic consistency property is an analytic consistency property.

(c) Prove that consistency within the Hilbert system H of problem 5.5.4 is a synthetic consistency property.

∗ **5.5.24.** A set S of formulae is *Henkin-complete* if for every formula $D = \exists x B$ of type d, there is some constant c such that $B(c)$ is also in S.

Prove that if P is a synthetic consistency property and S is a set of formulae that is both Henkin-complete and a maximal set for which P holds, then S is satisfiable.

Hint: Show that S is a Hintikka set for the term algebra consisting of all terms built up from function, constant symbols, and variables occuring free in S.

∗ **5.5.25.** Prove that if P is a synthetic consistency property, then every set S of formulae for which P holds can be extended to a Henkin-complete set which is a maximal set for which P holds.

Hint: Use the idea of problem 5.5.17.

Use the above property to prove the completeness of the Hilbert system H.

∗ **5.5.26.** Let **L** be a countable first-order language without equality. The method of this problem that is due to Henkin (reported in Smullyan, 1968, Chapter 10, page 96) yields one of the most elegant proofs of the model existence theorem.

Using this method, a set that is both Henkin-complete and maximal consistent is obtained from a *single* construction.

Let $A_1, ..., A_n, ...$ be an enumeration of all **L**-formulae. Let P be a synthetic consistency property, and let S be a set of formulae for which P holds. Construct the sequence S_n as follows:

$$S_0 = S;$$

$$S_{n+1} = \begin{cases} S_n \cup \{A_{n+1}\} & \text{If } P \text{ satisfies } S_n \cup \{A_{n+1}\}; \\ S_n \cup \{\neg A_{n+1}\} & \text{otherwise;} \end{cases}$$

In addition, if A_{n+1} is a formula $\exists x D$ of type d, then add $D(c)$ to S_{n+1} for some new constant c not in S_n. (It may be necessary to add countably many new constants to **L**.)

(a) Prove that $S' = \bigcup_{n\geq 0} S_n$ is Henkin-complete, and a maximal set such that P holds for S'.

(b) Let P be an analytic consistency property. Given a set S of formulae, let $Des(S)$ be the set of *immediate descendants* of formulae in S as defined in problem 5.5.18, and define S^n by induction as follows:

$$S^0 = S;$$
$$S^{n+1} = Des(S^n).$$
$$\text{Let } S^* = \bigcup_{n\geq 0} S^n.$$

S^* is called the *set of descendants* of S. Use the construction of (a) to prove that every set S of formulae for which P holds can be extended to a Henkin-complete subset S' of S^* which is a maximal subset of S^* for which P holds.

Hint: Consider an enumeration of S^* and extend S to a subset of S^*, which is Henkin-complete and maximal with respect to P.

Why does such a set generally fail to be the set of formulae satisfied by some structure?

(c) Prove that if M is a subset of S^* which is Henkin-complete and is a maximal subset of S^* for which P holds, then M is a Hintikka set with respect to the term algebra H consisting of all terms built up from function symbols, constants, and variables occurring free in M.

(d) Use (c) to prove that every set S of formulae for which P holds is satisfiable.

5.6 A Gentzen System for First-Order Languages With Equality

Let **L** be a first-order language with equality. The purpose of this section is to generalize the results of Section 5.5 to first-order languages with equality. First, we need to generalize the concept of a Hintikka set and lemma 5.4.5 to deal with languages with equality.

5.6.1 Hintikka Sets for Languages With Equality

The only modification to the definition of a signed formula (definition 5.4.3) is that we allow atomic formulae of the form $(s \doteq t)$ as the formula A in TA or FA.

Definition 5.6.1 A *Hintikka S set with respect to a term algebra H* (over the reduct $\mathbf{L}_S$ *with equality*) is a set of signed $\mathbf{L}$-formulae such that the following conditions hold for all signed formulae A,B,C,D of type a,b,c,d:

H0: No atomic formula and its conjugate are both in S (TA or FA is atomic iff A is).

H1: If a type-a formula A is in S, then both A_1 and A_2 are in S.

H2: If a type-b formula B is in S, then either B_1 is in S or B_2 is in S.

H3: If a type-c formula C is in S, then for every $t \in H$, $C(t)$ is in S (we require that t is free for x in C for every $t \in H$).

H4: If a type-d formula D is in S, then for at least one term $t \in H$, $D(t)$ is in S (we require that t is free for x in D for every $t \in H$).

H5: Every variable x occurring free in some formula of S is in H.

H6(i): For every term $t \in H$,

$$T(t \doteq t) \in S.$$

(ii): For every n-ary function symbol f in $\mathbf{L}_S$ and any terms $s_1, ..., s_n$, $t_1, ..., t_n \in H$,

$$T((s_1 \doteq t_1) \wedge ... \wedge (s_n \doteq t_n) \supset (fs_1...s_n \doteq ft_1...t_n)) \in S.$$

(iii): For every n-ary predicate symbol P in $\mathbf{L}_S$ (including $\doteq$) and any terms $s_1, ..., s_n, t_1, ..., t_n \in H$

$$T(((s_1 \doteq t_1) \wedge ... \wedge (s_n \doteq t_n) \wedge Ps_1...s_n) \supset Pt_1...t_n) \in S.$$

As in lemma 5.4.5, we can assume without loss of generality that the set of variables occurring free in formulae in S is disjoint from the set of variables occurring bound in formulae in S.

The alert reader will have noticed that conditions H6(i) to H6(iii) do not state explicitly that $\doteq$ is an equivalence relation. However, these conditions will be used in the proof of lemma 5.6.1 to show that a certain relation on terms is a congruence as defined in Subsection 2.4.6. The generalization of lemma 5.4.5 is as follows.

Lemma 5.6.1 Every Hintikka set S (over a language with equality) with respect to a term algebra H is satisfiable in a structure $\mathbf{H_S}$ whose domain H_S is a quotient of H.

Proof: In order to define the $\mathbf{L}_S$-structure $\mathbf{H_S}$, we define the relation $\cong$ on the set of all terms in H as follows:

$$s \cong t \quad \text{if and only if} \quad T(s \doteq t) \in S.$$

First, we prove that $\cong$ is an equivalence relation.

(1) By condition H6(i), $\cong$ is reflexive.

(2) Assume that $T(s \doteq t) \in S$. By H6(iii),

$$T((s \doteq t) \land (s \doteq s) \land Pss \supset Pts) \in S$$

for any binary predicate symbol P, and in particular when P is $\doteq$. By H2, either $T(t \doteq s) \in S$ or $F((s \doteq t) \land (s \doteq s) \land (s \doteq s)) \in S$. In the second case, using H2 again either $F(s \doteq t) \in S$, or $F(s \doteq s) \in S$. In each case we reach a contradiction by H0 and H6(i). Hence, we must have $T(t \doteq s) \in S$, establishing that $\cong$ is symmetric.

(3) To prove transitivity we proceed as follows. Assume that $T(r \doteq s)$ and $T(s \doteq t)$ are in S. By (2) above, $T(s \doteq r) \in S$. By H6(iii), we have

$$T((s \doteq r) \land (s \doteq t) \land (s \doteq s) \supset (r \doteq t)) \in S.$$

(With P being identical to $\doteq$.) A reasoning similar to that used in (2) shows that $T(r \doteq t) \in S$, establishing transitivity.

Hence, $\cong$ is an equivalence relation on H.

We now define the structure $\mathbf{H_S}$ as follows. The domain H_S of this structure is the quotient of the set H modulo the equivalence relation $\cong$, that is, the set of all equivalence classes of terms in H modulo $\cong$ (see Subsection 2.4.7). Given a term $t \in H$, we let $\bar{t}$ denote its equivalence class. The interpretation function is defined as follows:

Every constant c in $\mathbf{L}_S$ is interpreted as the equivalence class $\bar{c}$;

Every function symbol f of rank n in $\mathbf{L}_S$ is interpreted as the function such that, for any equivalence classes $\overline{t_1}$,...,$\overline{t_n}$,

$$f_{\mathbf{H_S}}(\overline{t_1}, ..., \overline{t_n}) = \overline{ft_1...t_n};$$

For every predicate symbol P of rank n in $\mathbf{L}_S$, for any equivalence classes $\overline{t_1}$,...,$\overline{t_n}$,

$$P_{\mathbf{H_S}}(\overline{t_1}, ..., \overline{t_n}) = \begin{cases} \mathbf{F} & \text{if } FPt_1...t_n \in S, \\ \mathbf{T} & \text{if } TPt_1...t_n \in S, \\ & \text{or neither } TPt_1...t_n \text{ nor } FPt_1...t_n \text{ is in } S. \end{cases}$$

To define a quotient structure as in Subsection 2.4.7, we need to show that the equivalence relation $\cong$ is a congruence (as defined in Subsection 2.4.6) so that the functions $f_{\mathbf{H_S}}$ and the predicates $P_{\mathbf{H_S}}$ are well defined; that is, that their definition is independent of the particular choice of representatives $t_1, ..., t_n$ in the equivalence classes of terms (in H). This is shown using condition H6(ii) for functions and condition H6(iii) for predicates. More specifically, the following claim is proved:

Claim 1: For any terms $s_1, ..., s_n, t_1, ..., t_n \in H$, if $\overline{s_i} = \overline{t_i}$ for $i = 1, .., n$, then:

(i) For each n-ary function symbol f,

$$\overline{fs_1...s_n} = \overline{ft_1...t_n}.$$

(ii) For each n-ary predicate symbol P,

$$\begin{array}{llll} \text{if} \quad TPs_1...s_n \in S & \text{then} \quad TPt_1...t_n \in S, & \text{and} \\ \text{if} \quad FPs_1...s_n \in S & \text{then} \quad FPt_1...t_n \in S. \end{array}$$

Proof of claim 1:

To prove (i), assume that $T(s_i \doteq t_i) \in S$, for all i, $1 \leq i \leq n$. Since by H6(ii),

$$T((s_1 \doteq t_1) \wedge ... \wedge (s_n \doteq t_n) \supset (fs_1...s_n \doteq ft_1...t_n)) \in S,$$

by condition H2 of a Hintikka set, either

$$T(fs_1...s_n \doteq ft_1...t_n) \in S,$$

or

$$F((s_1 \doteq t_1) \wedge ... \wedge (s_n \doteq t_n)) \in S.$$

But if $F((s_1 \doteq t_1) \wedge ... \wedge (s_n \doteq t_n)) \in S$, by H2, some $F(s_i \doteq t_i) \in S$, contradicting H0. Hence, $T(fs_1...s_n \doteq ft_1...t_n)$ must be in S, which is just the conclusion of (i) by the definition of the relation $\cong$.

Next, we prove (ii). If $TPs_1...s_n \in S$, since by H6(iii),

$$T((s_1 \doteq t_1) \wedge ... \wedge (s_n \doteq t_n) \wedge Ps_1...s_n \supset Pt_1...t_n) \in S,$$

by condition H2, either

$$TPt_1...t_n \in S,$$

or

$$F((s_1 \doteq t_1) \wedge ... \wedge (s_n \doteq t_n) \wedge Ps_1...s_n) \in S.$$

By condition H2, either

$$FPs_1...s_n \in S,$$

or

$$F((s_1 \doteq t_1) \land ... \land (s_n \doteq t_n)) \in S.$$

In the first case, H0 is violated, and in the second case, since we have assumed that $T(s_i \doteq t_i) \in S$ for all i, $1 \leq i \leq n$, H0 is also violated as in the proof of (i). Hence, $TPt_1...t_n$ must be in S.

If $FPs_1...s_n \in S$, since by H6(iii),

$$T((t_1 \doteq s_1) \land ... \land (t_n \doteq s_n) \land Pt_1...t_n \supset Ps_1...s_n) \in S,$$

either

$$TPs_1...s_n \in S,$$

or

$$F((t_1 \doteq s_1) \land ... \land (t_n \doteq s_n) \land Pt_1...t_n) \in S.$$

In the first case, H0 is violated. In the second case, either

$$FPt_1...t_n \in S,$$

or

$$F((t_1 \doteq s_1) \land ... \land (t_n \doteq s_n)) \in S.$$

However, if

$$F((t_1 \doteq s_1) \land ... \land (t_n \doteq s_n)) \in S,$$

since we have assumed that $Ts_i \doteq t_i \in S$ for all i, $1 \leq i \leq n$, and we have shown in (2) that whenever $Ts_i \doteq t_i \in S$, then $Tt_i \doteq s_i$ is also in S, H0 is also contradicted. Hence, $FPt_1...t_n$ must be in S. This concludes the proof of claim 1. $\square$

Claim (1)(i) shows that $\cong$ is a congruence with respect to function symbols. To show that $\cong$ is a congruence with respect to predicate symbols, we show that:

If $s_i \cong t_i$ for all i, $1 \leq i \leq n$, then

$$\begin{array}{lll} TPs_1...s_n \in S & \text{iff} & TPt_1...t_n \in S \quad \text{and} \\ FPs_1...s_n \in S & \text{iff} & FPt_1...t_n \in S. \end{array}$$

Proof: If $TPs_1...s_n \in S$, by claim 1(ii), $TPt_1...t_n \in S$. If $TPs_1...s_n \notin S$, then either

$$FPs_1...s_n \in S,$$

or

$$FPs_1...s_n \notin S.$$

If $FPs_1...s_n \in S$, by claim 1(ii), $FPt_1...t_n \in S$, and by H0, $TPt_1...t_n \notin S$. If $FPs_1...s_n \notin S$, then if $TPt_1...t_n$ was in S, then by claim 1(ii), $TPs_1...s_n$

would be in S, contrary to the assumption. The proof that $FPs_1...s_n \in S$ iff $FPt_1...t_n \in S$ is similar. This concludes the proof. $\square$

Having shown that the definition of the structure $\mathbf{H_S}$ is proper, let s be any assignment such that $s(x) = \overline{x}$, for each variable $x \in H$. It remains to show that $\mathbf{H_S}$ and s satisfy S. For this, we need the following claim which is proved using the induction principle for terms:

Claim 2: For every term t (in H),

$$t_{\mathbf{H_S}}[s] = \overline{t}.$$

If t is a variable x, then $x_{\mathbf{H_S}}[s] = s(x) = \overline{x}$, and the claim holds.

If t is a constant c, then

$$c_{\mathbf{H_S}}[s] = c_{\mathbf{H_S}} = \overline{c}$$

by the definition of the interpretation function.

If t is a term $ft_1...t_k$, then

$$(ft_1...t_k)_{\mathbf{H_S}}[s] = f_{\mathbf{H_S}}((t_1)_{\mathbf{H_S}}[s], ..., (t_k)_{\mathbf{H_S}}[s]).$$

By the induction hypothesis, for all i, $1 \leq i \leq k$,

$$(t_i)_{\mathbf{H_S}}[s] = \overline{t}_i.$$

Hence,

$$\begin{aligned}(ft_1...t_k)_{\mathbf{H_S}}[s] &= f_{\mathbf{H_S}}((t_1)_{\mathbf{H_S}}[s], ..., (t_k)_{\mathbf{H_S}}[s]) \\ &= f_{\mathbf{H_S}}(\overline{t_1}, ..., \overline{t_k}).\end{aligned}$$

But by the definition of $f_{\mathbf{H_S}}$,

$$f_{\mathbf{H_S}}(\overline{t_1}, ..., \overline{t_k}) = \overline{ft_1...t_k} = \overline{t}.$$

This concludes the induction and the proof of claim 2. $\square$

Claim 2 is now used to show that for any atomic formula $TPt_1...t_n \in S$, $Pt_1...t_n[s]$ is satisfied in $\mathbf{H_S}$ (and that for any $FPt_1...t_n \in S$, $Pt_1...t_n[s]$ is not satisfied in $\mathbf{H_S}$). Indeed, if $TPt_1...t_n \in S$, by H5 and since H is a term algebra, $t_1, ..., t_n \in H$. Then,

$$\begin{aligned}(Pt_1...t_n)_{\mathbf{H_S}}[s] &= P_{\mathbf{H_S}}((t_1)_{\mathbf{H_S}}[s], ..., (t_n)_{\mathbf{H_S}}[s]) \\ &= \text{(by Claim 2) } P_{\mathbf{H_S}}(\overline{t_1}, ..., \overline{t_n}) = \mathbf{T}.\end{aligned}$$

Hence,

$$\mathbf{H_S} \models (Pt_1...t_n)[s].$$

A similar proof applies when $FPt_1...t_n \in S$. Also, if $T(t_1 \doteq t_2) \in S$, by definition of $\cong$ we have $t_1 \cong t_2$, which is equivalent to $\overline{t_1} = \overline{t_2}$, that is, $(t_1)_{\mathbf{H_S}}[s] = (t_2)_{\mathbf{H_S}}[s]$. The rest of the argument proceeds using the induction principle for formulae. The propositional connectives are handled as before.

Let us give the proof for a signed formula of type c. If a formula C of type c is in S, then by H3, $C(t)$ is in S for every term $t \in H$. By the induction hypothesis, we have

$$\mathbf{H_S} \models C(t)[s].$$

By lemma 5.4.1, for any formula A, any term t free for x in A, any structure $\mathbf{M}$ and any assignment v, we have

$$(A[t/x])_{\mathbf{M}}[v] = (A[\mathbf{a}/x])_{\mathbf{M}}[v],$$

where $a = t_{\mathbf{M}}[v]$.

Since $t_{\mathbf{H_S}}[s] = \overline{t}$, we have

$$(C[t/x])_{\mathbf{H_S}}[s] = (C[\overline{\mathbf{t}}/x])_{\mathbf{H_S}}[s].$$

Hence,

$$\mathbf{H_S} \models C(t)[s] \quad \text{iff} \quad \mathbf{H_S} \models C(\overline{\mathbf{t}})[s],$$

and since $\overline{t} \in \mathbf{H_S}$, by lemma 5.4.4, this shows that

$$\mathbf{H_S} \models C[s].$$

As in lemma 5.4.5, $\mathbf{H_S}$ is expanded to an $\mathbf{L}$-structure in which S is satisfiable. This concludes the proof that every signed formula in S is satisfied in $\mathbf{H_S}$ by the assignment s. □

5.6.2 The Gentzen System $G_=$ (Languages With Equality)

Let $G_=$ be the Gentzen system obtained from the Gentzen system G (defined in definition 5.4.1) by adding the following rules.

Definition 5.6.2 (Equality rules for $G_=$) Let Γ, Δ, Λ denote arbitrary sequences of formulae (possibly empty) and let t,s_1,...,s_n, t_1,...,t_n denote arbitrary $\mathbf{L}$-terms.

For each term t, we have the following rule:

$$\frac{\Gamma, t \doteq t \rightarrow \Delta}{\Gamma \rightarrow \Delta} \quad \textit{reflexivity}$$

For each n-ary function symbol f and any terms $s_1, ..., s_n$, $t_1, ..., t_n$, we have the following rule:

$$\frac{\Gamma, (s_1 \doteq t_1) \land ... \land (s_n \doteq t_n) \supset (fs_1...s_n \doteq ft_1...t_n) \rightarrow \Delta}{\Gamma \rightarrow \Delta}$$

For each n-ary predicate symbol P (including $\doteq$) and any terms $s_1, ..., s_n$, $t_1, ..., t_n$, we have the following rule:

$$\frac{\Gamma, ((s_1 \doteq t_1) \land ... \land (s_n \doteq t_n) \land Ps_1...s_n) \supset Pt_1...t_n \rightarrow \Delta}{\Gamma \rightarrow \Delta}$$

Note that the reason these formulae are added after the rightmost formula of the premise of a sequent is that in the proof of theorem 5.6.1, this simplifies the reconstruction of a legal proof tree from a closed tree. In fact, as in definition 5.4.8, we generalize the above rules to inferences of the form

$$\frac{\Gamma, A_1, ..., A_k \rightarrow \Delta}{\Gamma \rightarrow \Delta}$$

where $A_1, ..., A_k$ are any formulae of the form either

$$t \doteq t,$$
$$(s_1 \doteq t_1) \land ... \land (s_n \doteq t_n) \supset (fs_1...s_n \doteq ft_1...t_n), \quad \text{or}$$
$$((s_1 \doteq t_1) \land ... \land (s_n \doteq t_n) \land Ps_1...s_n) \supset Pt_1...t_n.$$

EXAMPLE 5.6.1

Let

$$A = \forall x(x \doteq f(y) \supset P(x)) \supset P(f(y)),$$

where f is a unary function symbol, and P is a unary predicate symbol.

The following tree is a proof tree for A.

$$\dfrac{\dfrac{\dfrac{T_1 \quad P(f(y)), \forall x(x \doteq f(y) \supset P(x)) \rightarrow P(f(y))}{f(y) \doteq f(y) \supset P(f(y)), \forall x(x \doteq f(y) \supset P(x)) \rightarrow P(f(y))}}{\forall x(x \doteq f(y) \supset P(x)) \rightarrow P(f(y))}}{\rightarrow \forall x(x \doteq f(y) \supset P(x)) \supset P(f(y))}$$

where T_1 is the tree

$$\frac{\forall x(x \doteq f(y) \supset P(x)), f(y) \doteq f(y) \rightarrow f(y) \doteq f(y), P(f(y))}{\forall x(x \doteq f(y) \supset P(x)) \rightarrow f(y) \doteq f(y), P(f(y))}$$

EXAMPLE 5.6.2

Let

$$A = P(f(y)) \supset \forall x(x \doteq f(y) \supset P(x)).$$

The following tree is a proof tree for A.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{T_1 \qquad\qquad P(z), P(f(y)), z \doteq f(y) \to P(z)}{P(f(y)), z \doteq f(y), (f(y) \doteq z \land P(f(y))) \supset P(z) \to P(z)}}{P(f(y)), z \doteq f(y) \to P(z)}}{P(f(y)) \to z \doteq f(y) \supset P(z)}}{P(f(y)) \to \forall x(x \doteq f(y) \supset P(x))}}{\to P(f(y)) \supset \forall x(x \doteq f(y) \supset P(x))}$$

where T_1 is the tree

$$\dfrac{T_2 \qquad\qquad P(f(y)), z \doteq f(y) \to P(f(y)), P(z)}{P(f(y)), z \doteq f(y) \to f(y) \doteq z \land P(f(y)), P(z)}$$

T_2 is the tree

$$\dfrac{\dfrac{T_3 \qquad\qquad f(y) \doteq z, P(f(y)), z \doteq f(y) \to f(y) \doteq z, P(z)}{P(f(y)), z \doteq f(y), (z \doteq f(y) \land z \doteq z \land z \doteq z) \supset f(y) \doteq z \to f(y) \doteq z, P(z)}}{P(f(y)), z \doteq f(y) \to f(y) \doteq z, P(z)}$$

and T_3 is the tree

$$\dfrac{S_1 \qquad \dfrac{\dfrac{S_2 \qquad\qquad S_2}{P(f(y)), z \doteq f(y), z \doteq z \to z \doteq z \land z \doteq z, f(y) \doteq z, P(z)}}{P(f(y)), z \doteq f(y) \to z \doteq z \land z \doteq z, f(y) \doteq z, P(z)}}{P(f(y)), z \doteq f(y) \to z \doteq f(y) \land z \doteq z \land z \doteq z, f(y) \doteq z, P(z)}$$

The sequent S_1 is

$$P(f(y)), z \doteq f(y) \to z \doteq f(y), f(y) \doteq z, P(z)$$

and the sequent S_2 is

$$P(f(y)), z \doteq f(y), z \doteq z \to z \doteq z, f(y) \doteq z, P(z).$$

5.6.3 Soundness of the System $G_=$

First, the following lemma is easily shown.

Lemma 5.6.2 For any of the equality rules given in definition 5.6.2 (including their generalization), the premise is falsifiable if and only if the conclusion is falsifiable.

Proof: Straightforward, and left as an exercise. □

Lemma 5.6.3 (Soundness of the System $G_=$) If a sequent is $G_=$-provable then it is valid.

Proof: Use the induction principle for $G_=$-proofs and lemma 5.6.2. □

5.6.4 Search Procedure for Languages With Equality

To prove the completeness of the system $G_=$ we modify the procedure *expand* in the following way. Given a sequent $\Gamma_0 \rightarrow \Delta_0$, let $\mathbf{L}'$ be the reduct of $\mathbf{L}$ consisting of all constant, function, and predicate symbols occurring in formulae in $\Gamma_0 \rightarrow \Delta_0$ (It is also assumed that the set of variables occurring free in $\Gamma_0 \rightarrow \Delta_0$ is disjoint from the set of variables occurring bound in $\Gamma_0 \rightarrow \Delta_0$, which is possible by lemma 5.3.4.)

Let $EQ_{1,0}$ be an enumeration of all $\mathbf{L}'$-formulae of the form

$$(t \doteq t),$$

$EQ_{2,0}$ an enumeration of all $\mathbf{L}'$-formulae of the form

$$(s_1 \doteq t_1) \wedge ... \wedge (s_n \doteq t_n) \supset (fs_1...s_n \doteq ft_1...t_n) \text{ and}$$

$EQ_{3,0}$ an enumeration of all $\mathbf{L}'$-formulae of the form

$$((s_1 \doteq t_1) \wedge ... \wedge (s_n \doteq t_n) \wedge Ps_1...s_n) \supset Pt_1...t_n,$$

where the terms $t, s_1, ..., s_n, t_1, ...t_n$ are built from the constant and function symbols in $\mathbf{L}'$, and the set of variables free in formulae in $\Gamma_0 \rightarrow \Delta_0$. Note that these terms belong to the set $TERMS$, defined in Subsection 5.5.1.

For $i \geq 1$, we define the sets $EQ_{1,i}$, $EQ_{2,i}$ and $EQ_{3,i}$ as above, except that the terms $t, s_1, ..., s_n, t_1, ...t_n$ belong to the lists $AVAIL_i$ (at the start). During a round, at the end of every expansion step, we shall add each formula that is the head of the list $EQ_{j,i}$, where $j = (RC \ mod \ 3) + 1$ (RC is a *round counter* initialized to 0 and incremented at the end of every round) for all $i = 0, ..., NUMACT$, to the antecedent of every leaf sequent. At the end of the round, each such formula is deleted from the head of the list $EQ_{j,i}$. In this way, the set S of signed formulae along any (finite or infinite) *nonclosed* path in the tree will be a Hintikka set.

We also add to the procedure *search*, statements to initialize the lists $EQ_{1,i}$, $EQ_{2,i}$ and $EQ_{3,i}$ as explained above.

Note that if the sets **FS**, **PS** and **CS** are effective (recursive), recursive functions enumerating the lists $EQ_{1,i}$, $EQ_{2,i}$ and $EQ_{3,i}$ can be written (these lists are in fact recursive).

In the new version of the *search* procedure for sequents containing the equality symbol, a leaf of the tree is *finished* iff it is an axiom. Hence, the *search* procedure always builds an infinite tree if the input sequent is not valid. This is necessary to guarantee that conditions H6(i) to (iii) are satisfied.

Definition 5.6.3 Procedure *search* for a language with equality.

Procedure Expand for a Language with Equality

```
procedure expand(node : tree-address; var T : tree);
  begin
    let $A_1, ..., A_m \rightarrow B_1, ..., B_n$ be the label of node;
    let S be the one-node tree labeled with
      $A_1, ..., A_m \rightarrow B_1, ..., B_n$;
      for $i := 1$ to $m$ do
        if nonatomic($A_i$) then
          grow-left($A_i, S$)
        endif
      endfor;
      for $i := 1$ to $n$ do
        if nonatomic($B_i$) then
          grow-right($B_i, S$)
        endif
      endfor;
      for each leaf u of S do
        let $\Gamma \rightarrow \Delta$ be the label of u;
        $\Gamma' := \Gamma, head(L)$;
        $\Delta' := \Delta, head(R)$;
        for $i := 0$ to $NUMACT_0$ do
          $\Gamma' := \Gamma', head(EQ_{j,i})$
        endfor;
        create a new node u1 labeled with $\Gamma' \rightarrow \Delta'$
      endfor;
      $T := dosubstitution(T, node, S)$
  end
```

Procedure Search for a Language with Equality

```
procedure $search(\Gamma_0 \to \Delta_0 : sequent$; var $T : tree)$;
  begin
    $L := tail(\Gamma_0)$; $\Gamma := head(\Gamma_0)$;
    $R := tail(\Delta_0)$; $\Delta := head(\Delta_0)$;
    let $T$ be the one-node tree labeled with $\Gamma \to \Delta$;
    Let $TERM_0$, $EQ_{1,i}$, $EQ_{2,i}$, $EQ_{3,i}$ and $AVAIL_i$, $0 \leq i$,
    be initialized as explained above.
    $NUMACT := 0$; $RC := 0$;
    while not all leaves of $T$ are finished do
      $TERM_1 := TERM_0$; $T_0 := T$; $NUMACT_0 := NUMACT$;
      $j := (RC \ mod \ 3) + 1$;
      for each leaf node of $T_0$
      (in lexicographic order of tree addresses) do
        if not $finished(node)$ then
          $expand(node, T)$
        endif
      endfor;
      $TERM_0 := TERM_1$; $L := tail(L)$; $R := tail(R)$;
      for $i := 0$ to $NUMACT_0$ do
        $EQ_{j,i} := tail(EQ_{j,i})$
      endfor;
      $RC := RC + 1$;
      for $i := 0$ to $NUMACT$ do
        $TERM_0 := append(TERM_0, < head(AVAIL_i), nil >)$;
        $AVAIL_i := tail(AVAIL_i)$
      endfor
    endwhile;
    if all leaves are closed
    then
      write ('$T$ is a proof of $\Gamma_0 \to \Delta_0$')
    else
      write ('$\Gamma_0 \to \Delta_0$ is falsifiable')
    endif
  end
```

5.6.5 Completeness of the System $G_=$

Using lemma 5.6.1, it is easy to generalize theorem 5.5.1 as follows.

Theorem 5.6.1 For every sequent $\Gamma_0 \to \Delta_0$ containing the equality symbol and in which no free variable occurs bound the following holds:

(i) If the input sequent $\Gamma_0 \to \Delta_0$ is valid, then the procedure *search* halts with a closed tree T, from which a proof tree for a finite subsequent $C_1, ..., C_m \to D_1, ..., D_n$ of $\Gamma_0 \to \Delta_0$ can be constructed.

(ii) If the input sequent $\Gamma_0 \to \Delta_0$ is falsifiable, then *search* builds an infinite counter-example tree T and $\Gamma_0 \to \Delta_0$ can be falsified in a finite or a countably infinite structure which is a quotient of $t(TERM_0)$.

Proof: It is similar to that of theorem 5.5.1 but uses lemma 5.6.1 instead of lemma 5.4.5 in order to deal with equality. What is needed is the following claim:

Claim: If S is the set of signed formulae along any nonclosed path in the tree T generated by *search*, the set $H_S = t(TERM_0)$ is a term algebra (over the reduct $\mathbf{L}'$) and S is a Hintikka set for H_S.

The proof of the claim is essentially the same as the proof given for theorem 5.5.1. The only difference is that it is necessary to check that conditions H6(i) to (iii) hold, but this follows immediately since all formulae in the lists $EQ_{j,0}$ and $EQ_{j,i}$ for all activated variables y_i $(i > 0)$ are eventually entered in each infinite path of the tree. $\square$

Corollary (A version of Gödel's extended completeness theorem for $G_=$) Let $\mathbf{L}$ be a first-order language with equality. A sequent in which no variable occurs both free and bound (even infinite) is valid iff it is $G_=$-provable. $\square$

Again, as observed in Section 5.5 in the paragraph immediately after the proof of theorem 5.5.1, although constructive and theoretically complete, the *search* procedure is horribly inefficient. But in the case of a language with equality, things are even worse. Indeed, the management of the equality axioms (the lists $EQ_{j,i}$) adds significantly to the complexity of the bookkeeping. In addition, it is no longer true that the *search* procedure always halts for quantifier-free sequents, as it does for languages without equality. This is because the equality rules allow the introduction of new formulae.

It can be shown that there is an algorithm for deciding the validity of quantifier-free formulae with equality by adapting the congruence closure method of Kozen, 1976, 1977. For a proof, see Chapter 10.

The difficulty is that the *search* procedure no longer preserves the property known as the *subformula property*. For languages without equality, the Gentzen rules are such that given any input sequent $\Gamma \to \Delta$, the formulae occurring in any deduction tree with root $\Gamma \to \Delta$ contain only subformulae of the formulae in $\Gamma \to \Delta$. These formulae are in fact "descendants" of the formulae in $\Gamma \to \Delta$, which are essentially "signed subformulae" of the formulae in $\Gamma \to \Delta$ (for a precise definition, see problem 5.5.18). We also say that such proofs are *analytic*. But the equality rules can introduce "brand new" formulae that may be totally unrelated to the formulae in $\Gamma \to \Delta$. Of course, one could object that this problem arises because the rules of $G_=$ were badly designed, and that this problem may not arise for a better system. However, it is not easy to find a complete proof system for first-order logic with equality, that incorporates a weaker version of the subformula property. It will be shown in the next chapter that there is a Gentzen system LK_e, such that for every sequent $\Gamma \to \Delta$ provable in $G_=$, there is a proof in LK_e in which the formulae occurring in the proof are not too unrelated to the formulae in $\Gamma \to \Delta$. This is a consequence of the cut elimination theorem without essential cuts for LK_e. Nevertheless, theorem proving for languages with equality tends to be harder than theorem proving for languages without equality. We shall come back to this point later, in particular when we discuss the resolution method.

Returning to the completeness theorem, the following classical results apply to first-order languages with equality and are immediate consequences of theorem 5.6.1.

5.6.6 Löwenheim-Skolem, Compactness, and Model Existence Theorems for Languages With Equality

For a language with equality, the structure given by Löwenheim-Skolem theorem may be finite.

Theorem 5.6.2 (Löwenheim-Skolem) Let **L** be a first-order language with equality. If a set of formulae Γ over **L** is satisfiable in some structure **M**, then it is satisfiable in a structure whose domain is at most countably infinite.

Proof: From theorem 5.6.1, Γ is satisfiable in a quotient structure of $\mathbf{H_S}$. Since, $\mathbf{H_S}$ is countable, a quotient of $\mathbf{H_S}$ is countable, but possibly finite. $\square$

Theorem 5.6.3 (Compactness theorem) Let **L** be a first-order language with equality. For any (possibly countably infinite) set Γ of formulae over **L**, if every nonempty finite subset of Γ is satisfiable then Γ is satisfiable. $\square$

Theorem 5.6.4 (Model existence theorem) Let **L** be a first-order language with equality. If a set Γ of formulae over **L** is consistent, then it is satisfiable. $\square$

Note that if Γ is consistent, then by theorem 5.6.4 the sequent $\Gamma \to$ is falsifiable. Hence the *search* procedure run on input $\Gamma \to$ will actually yield a structure $\mathbf{H_S}$ (with domain a quotient of $t(TERM_0)$) for each Hintikka set S arising along each nonclosed path in the tree T, in which S and Γ are satisfiable.

We also have the following lemma.

Lemma 5.6.4 (Consistency lemma) Let **L** be a first-order language with equality. If a set Γ of formulae is satisfiable then it is consistent. $\square$

5.6.7 Maximal Consistent Sets

As in Subsection 5.5.5, a consistent set Γ of first-order formulae (possibly involving equality) is *maximally consistent* (or a *maximal consistent set*) iff, for every consistent set Δ, if $\Gamma \subseteq \Delta$, then $\Gamma = \Delta$. Lemma 5.5.2 can be easily generalized to the first-order languages with equality.

Lemma 5.6.5 Given a first-order language (with or without equality), every consistent set Γ is a subset of some maximal consistent set Δ.

Proof: Almost identical to that of lemma 3.5.5, but using structures instead of valuations. $\square$

5.6.8 Application of the Compactness and Löwenheim-Skolem Theorems: Nonstandard Models of Arithmetic

The compactness and the Löwenhein-Skolem theorems are important tools in model theory. Indeed, they can be used for showing the existence of countable models for certain interesting theories. As an illustration, we show that the theory of example 5.3.2 known as Peano's arithmetic has a countable model nonisomorphic to $\mathbf{N}$, the set of natural numbers. Such a model is called a *nonstandard model*.

EXAMPLE 5.6.3

Let $\mathbf{L}$ be the language of arithmetic defined in example 5.3.1. The following set A_P of formulae is known as the axioms of *Peano's arithmetic*:

$$\forall x \neg(S(x) \doteq 0)$$
$$\forall x \forall y (S(x) \doteq S(y) \supset x \doteq y)$$
$$\forall x (x + 0 \doteq x)$$
$$\forall x \forall y (x + S(y) \doteq S(x + y))$$
$$\forall x (x * 0 \doteq 0)$$
$$\forall x \forall y (x * S(y) \doteq x * y + x)$$

For every formula A with one free variable x,

$$(A(0) \land \forall x (A(x) \supset A(S(x)))) \supset \forall y A(y)$$

Let $\mathbf{L}'$ be the expansion of $\mathbf{L}$ obtained by adding the new constant c. Let Γ be the union of A_P and the following set I of formulae:

$$I = \{\neg(0 \doteq c), \neg(1 \doteq c), ..., \neg(n \doteq c), ...\},$$

where n is an abbreviation for the term $S(S(...(0)))$, with n occurrences of S.

Observe that $\mathbf{N}$ can be made into a model of any finite subset of Γ. Indeed, $\mathbf{N}$ is a model of A_P, and to satisfy any finite subset X of I, it is sufficient to interpret c as any integer strictly larger than the maximum of the finite set $\{n \mid \neg(n \doteq c) \in X\}$. Since every finite subset of Γ is satisfiable, by the compactness theorem, Γ is satisfiable in some model. By the Löwenheim-Skolem theorem, Γ has a countable model, say $\mathbf{M}_0$. Let $\mathbf{M}$ be the reduct of $\mathbf{M}_0$ to the language $\mathbf{L}$. The model $\mathbf{M}$ still contains the element a, which is the interpretation of $c \in \mathbf{L}'$, but since c does not belong to $\mathbf{L}$, the interpretation function of the structure $\mathbf{M}$ does not have c in its domain.

It remains to show that $\mathbf{M}$ and $\mathbf{N}$ are not isomorphic. An *isomorphism* h from $\mathbf{M}$ to $\mathbf{N}$ is a bijection $h : \mathbf{M} \to \mathbf{N}$ such that:

$$\begin{gathered}h(0_{\mathbf{M}}) = 0 \text{ and, for all } x, y \in \mathbf{M},\\ h(S_{\mathbf{M}}(x)) = S(h(x)),\\ h(x +_{\mathbf{M}} y) = x + y \text{ and}\\ h(x *_{\mathbf{M}} y) = x * y.\end{gathered}$$

Assume that there is an isomorphism h between $\mathbf{M}$ and $\mathbf{N}$. Let $k \in \mathbf{N}$ be the natural number $h(a)$ (with $a = c_{\mathbf{M}_0}$ in $\mathbf{M}_0$). Then, for all $n \in \mathbf{N}$,

$$n \neq k.$$

Indeed, since the formula $\neg(n \doteq c)$ is valid in $\mathbf{M}_0$,

$$n_{\mathbf{M}} \neq a \text{ in } \mathbf{M},$$

and since h is injective, $n_{\mathbf{M}} \neq a$ implies $h(n_{\mathbf{M}}) \neq h(a)$, that is, $n \neq k$. But then, we would have $k \neq k$, a contradiction. $\square$

It is not difficult to see that in the model $\mathbf{M}$, $m_{\mathbf{M}} \neq n_{\mathbf{M}}$, whenever $m \neq n$. Hence, the natural numbers are represented in $\mathbf{M}$, and satisfy Peano's axioms. However, there may be other elements in $\mathbf{M}$. In particular, the interpretation a of $c \in \mathbf{L}'$ belongs to $\mathbf{M}$. Intuitively speaking, a is an infinite element of $\mathbf{M}$.

A model of A_P nonisomorphic to $\mathbf{N}$ such as $\mathbf{M}$ is called a *nonstandard model* of Peano's arithmetic.

For a detailed exposition of model theory including applications of the compactness theorem, Löwenheim-Skolem theorem, and other results, the reader is referred to Chang and Keisler, 1973; Monk, 1976; or Bell and Slomson, 1969. The introductory chapter on model theory by J. Keisler in Barwise, 1977, is also highly recommended.

PROBLEMS

5.6.1. Give proof tree for the following formulae:

$$\forall x(x \doteq x)$$
$$\forall x_1 ... \forall x_n \forall y_1 ... \forall y_n ((x_1 \doteq y_1 \land ... \land x_n \doteq y_n) \supset (f(x_1, ..., x_n) \doteq f(y_1, ..., y_n)))$$
$$\forall x_1 ... \forall x_n \forall y_1 ... \forall y_n (((x_1 \doteq y_1 \land ... \land x_n \doteq y_n) \land P(x_1, ..., x_n)) \supset P(y_1, ..., y_n))$$

The above formulae are called the *closed equality axioms*.

5.6.2. Let $S = \Gamma \rightarrow \Delta$ be a sequent, and let S_e be the set of closed equality axioms for all predicate and function symbols occurring in S. Prove that the sequent $\Gamma \rightarrow \Delta$ is provable in $G_=$ iff the sequent $S_e, \Gamma \rightarrow \Delta$ is provable in $G_= + \{cut\}$.

5.6.3. Prove that the following formula is provable:

$$f^3(a) \doteq a \land f^5(a) \doteq a \supset f(a) \doteq a$$

5.6.4. Let $*$ be a binary function symbol and 1 be a constant. Prove that the following sequent is valid:

$$\forall x(*(x, 1) \doteq x), \forall x(*(1, x) \doteq x),$$
$$\forall x \forall y \forall z(*(x, *(y, z)) \doteq *(*(x, y), z)),$$
$$\forall x(*(x, x) \doteq 1) \rightarrow \forall x \forall y(*(x, y) \doteq *(y, x))$$

5.6.5. Give proof trees for the following formulae:

$$\forall x \exists y(x \doteq y)$$
$$A[t/x] \equiv \forall x((x \doteq t) \supset A), \text{ if } x \notin Var(t) \text{ and } t \text{ is free for } x \text{ in } A.$$
$$A[t/x] \equiv \exists x((x \doteq t) \land A), \text{ if } x \notin Var(t) \text{ and } t \text{ is free for } x \text{ in } A.$$

5.6.6. (a) Prove that the following formulae are valid:

$$x \doteq x$$
$$x \doteq y \supset (x \doteq z \supset y \doteq z)$$
$$x \doteq y \supset (P(x_1, ..., x_{i-1}, x, x_{i+1}, ..., x_n) \supset P(x_1, ..., x_{i-1}, y, x_{i+1}, ..., x_n))$$
$$x \doteq y \supset (f(x_1, ..., x_{i-1}, x, x_{i+1}, ..., x_n) \doteq f(x_1, ..., x_{i-1}, y, x_{i+1}, ..., x_n))$$

Consider the extension $H_=$ of the Hilbert system H defined in problem 5.5.4 obtained by adding the above formulae known as the *open equality axioms*.

(b) Prove that

$$\forall x\forall y(x \doteq y \supset y \doteq x)$$

and

$$\forall x\forall y\forall z(x \doteq y \land y \doteq z \supset x \doteq z)$$

are provable in $H_=$.

(c) Given a set S of formulae (over a language with equality), let S_e be the set of universal closures (defined in problem 5.3.10) of equality axioms for all function and predicate symbols occuring in S. Prove that

$$S \vdash A \text{ in } H_= \quad \text{iff} \quad S, S_e \vdash A \text{ in } H.$$

(d) Prove the completeness of the Hilbert system $H_=$.

Hint: Use the result of problem 5.5.10 and, given a model for $S \cup S_e$, construct a model of S in which the predicate symbol $\doteq$ is interpreted as equality, using the quotient construction.

∗ **5.6.7.** In languages with equality, it is possible to eliminate function and constant symbols as shown in this problem.

Let **L** be a language (with or without equality). A *relational version* of **L** is a language **L**′ with equality satisfying the following properties:

(1) **L**′ has no function or constant symbols;

(2) There is an injection $T : \mathbf{FS} \cup \mathbf{CS} \rightarrow \mathbf{PS}'$ called a *translation* such that $r(T(f)) = r(f) + 1$;

(3) $\mathbf{PS}' = \mathbf{PS} \cup T(\mathbf{FS} \cup \mathbf{CS})$, and $T(\mathbf{FS} \cup \mathbf{CS})$ and **PS** are disjoint.

We define the *T-translate* A^T of an **L**-formula A as follows:

(1) If A is of the form $(s \doteq x)$, where s is a term and x is a variable, then

(i) If s is a variable y, then $(y \doteq x)^T = (y \doteq x)$;

(ii) If s is a constant c, then $(c \doteq x)^T = T(c)(x)$, where $T(c)$ is the unary predicate symbol associated with c;

(iii) If s is a term of the form $fs_1...s_n$, then

$$(s \doteq x)^T = \exists y_1...\exists y_n[(s_1 \doteq y_1)^T \land ... \land (s_n \doteq y_n)^T \land T(f)(y_1, ..., y_n, x)],$$

where $y_1, ..., y_n$ are new variables, and $T(f)$ is the $(n+1)$-ary predicate symbol associated with f;

(2) If A is of the form $(s \doteq t)$, where t is not a variable, then

$$(s \doteq t)^T = \exists y((s \doteq y)^T \land (t \doteq y)^T),$$

where y is a new variable.

(3) If A is of the form $Ps_1...s_n$, then

(i) If every $s_1, ..., s_n$ is a variable, then $(Ps_1...s_n)^T = Ps_1...s_n$;

(ii) If some s_i is not a variable, then

$$(Ps_1...s_n)^T = \exists y_1...\exists y_n[Py_1...y_n \land (s_1 \doteq y_1)^T \land ... \land (s_n \doteq y_n)^T],$$

where $y_1, ..., y_n$ are new variables.

(4) If A is not atomic, then

$$\begin{aligned}(\neg B)^T &= \neg B^T,\\ (B \lor C)^T &= B^T \lor C^T,\\ (B \land C)^T &= B^T \land C^T,\\ (B \supset C)^T &= (B^T \supset C^T),\\ (\forall x B)^T &= \forall x B^T,\\ (\exists x B)^T &= \exists x B^T.\end{aligned}$$

For every function or constant symbol f in $\mathbf{L}$, the *existence condition for f* is the following sentence in $\mathbf{L}'$:

$$\forall x_1...\forall x_n \exists y T(f)(x_1, ..., x_n, y);$$

The *uniqueness condition for f* is the following sentence in $\mathbf{L}'$:

$$\forall x_1...\forall x_n \forall y \forall z (T(f)(x_1, ..., x_n, y) \land T(f)(x_1, ..., x_n, z) \supset y \doteq z).$$

The set of *translation conditions* is the set of all existence and uniqueness conditions for all function and constant symbols in $\mathbf{L}$.

Given an $\mathbf{L}$-structure $\mathbf{A}$, the *relational version* $\mathbf{A}'$ of $\mathbf{A}$ has the same domain as $\mathbf{A}$, the same interpretation for the symbols in $\mathbf{PS}$, and interprets each symbol $T(f)$ of rank $n + 1$ as the relation

$$\{(x_1, ..., x_n, y) \in A^{n+1} \mid f_{\mathbf{A}}(x_1, ..., x_n) = y\}.$$

Prove the following properties:

(a) For every formula A of $\mathbf{L}$, if A does not contain function or constant symbols, then $A^T = A$.

(b) Let **A** be an **L**-structure, and **A**′ its relational version. Prove that **A**′ is a model of the translation conditions. Prove that for every **L**-formula A, for every assignment s,

$$\mathbf{A} \models A[s] \quad \text{iff} \quad \mathbf{A}' \models A^T[s].$$

(c) For every **L**′-formula A, let A^* be the formula obtained by replacing every atomic formula $T(f)(x_1, ..., x_n, y)$ by $(f(x_1, ..., x_n) \doteq y)$. Prove that

$$\mathbf{A} \models A^*[s] \quad \text{iff} \quad \mathbf{A}' \models A[s].$$

(d) Prove that if **B** is an **L**′-structure which is a model of the translation conditions, then $\mathbf{B} = \mathbf{A}'$ for some **L**-structure **A**.

(e) Let Γ be a set of **L**-formulae, and A any **L**-formula. Prove that

$$\Gamma \models A \quad \text{iff}$$
$$\{B^T \mid B \in \Gamma\} \cup \{B \mid B \text{ is a translation condition}\} \models A^T.$$

∗ **5.6.8.** Let **L** be a first-order language with equality. A *theory* is a set Γ of **L**-sentences such that for every **L**-sentence A, if $\Gamma \models A$, then $A \in \Gamma$. If **L**′ is an expansion of **L** and Γ' is a theory over **L**′, we say that Γ' is *conservative over* Γ, iff

$$\{A \mid A \text{ is an } \mathbf{L}\text{-sentence in } \Gamma'\} = \Gamma.$$

Let Γ be a theory over **L**, **L**′ an expansion of **L**, and Γ' a theory over **L**′.

(i) If P is an n-ary predicate symbol in **L**′ but not in **L**, a *possible definition of* P over Γ is an **L**-formula A whose set of free variables is a subset of $\{x_1, ..., x_n\}$.

(ii) If f is an n-ary function symbol or a constant in **L**′ but not in **L**, a *possible definition of* f over Γ is an **L**-formula A whose set of free variables is a subset of $\{x_1, ..., x_n, y\}$, such that the following *existence* and *uniqueness* conditions are in Γ:

$$\forall x_1 ... \forall x_n \exists y A,$$
$$\forall x_1 ... \forall x_n \forall y \forall z (A(x_1, ..., x_n, y) \land A(x_1, ..., x_n, z) \supset y \doteq z).$$

(iii) We say that Γ' over **L**′ is a *definitional extension* of Γ over **L** iff for every constant, function, or predicate symbol X in **L**′ but not in **L**, there is a possible definition A_X over Γ such that the condition stated below holds.

For every possible definition A_X, the sentence A'_X is defined as follows:

For an n-ary predicate symbol P, let A'_P be the sentence

$$\forall x_1 ... \forall x_n (P x_1 ... x_n \equiv A_P);$$

For an n-ary function symbol or a constant f, let A'_f be the sentence

$$\forall x_1 ... \forall x_n \forall y ((f x_1 ... x_n \doteq y) \equiv A_f).$$

Then we require that

$$\Gamma' = \{B \mid B \text{ is an } \mathbf{L}'\text{-formula such that,}$$
$$\Gamma \cup \{A'_X \mid X \text{ is a symbol in } \mathbf{L}' \text{ not in } \mathbf{L}\} \models B\}.$$

(a) Prove that if Γ' is a definitional extension of Γ, then for every $\mathbf{L}'$-formula A, there is an $\mathbf{L}$-formula A^T with the same free variables as A, such that

$$\Gamma' \models (A \equiv A^T).$$

Hint: Use the technique of problem 5.6.7.

(b) Prove that Γ' is conservative over Γ.

5.6.9. Let Γ be a theory over a language $\mathbf{L}$. A *set of axioms* Δ for Γ is any set of $\mathbf{L}$-sentences such that

$$\Gamma = \{B \mid \Delta \models B\}.$$

Given an $\mathbf{L}$-formula A with set of free variables $\{x_1, ..., x_n, y\}$, assume that

$$\Gamma \models \forall x_1 ... \forall x_n \exists y A(x_1, ..., x_n, y).$$

Let $\mathbf{L}'$ be the expansion of $\mathbf{L}$ obtained by adding the new n-ary function symbol f, and let Γ' be the theory with set of axioms

$$\Gamma \cup \{\forall x_1 ... \forall x_n A(x_1, ..., x_n, f(x_1, ..., x_n))\}.$$

Prove that Γ' is conservative over Γ.

$*$ **5.6.10.** Let $\mathbf{L}$ be a first-order language with equality. From problem 5.6.1, the closed equality axioms are provable in $G_=$. Recall the concept of a *Henkin theory* from problem 5.5.17.

(a) Prove that a maximally consistent (with respect to $G_=$) Henkin theory T' of sentences is a Hintikka set with respect to the term algebra H consisting of all closed terms built up from the function and constant symbols in $\mathbf{L}^H$.

(b) Prove that every consistent set T of $\mathbf{L}$-sentences is satisfiable.

(c) Prove that a formula A with free variables $\{x_1, ..., x_n\}$ is satisfiable iff $A[c_1/x_1, ..., c_n/x_n]$ is satisfiable, where $c_1, ..., c_n$ are new constants. Using this fact, prove that every consistent set T of $\mathbf{L}$-formulae is satisfiable.

5.6.11. Prove that in $G_= + \{cut\}$,

$$\Gamma \rightarrow A \text{ is not provable in } G_= + \{cut\} \text{ iff } \Gamma \cup \{\neg A\} \text{ is consistent.}$$

Use the above fact and problem 5.6.10 to give an alternate proof of the extended completeness theorem for $G_= + \{cut\}$.

$*$ **5.6.12.** Prove that the results of problem 5.6.10 hold for languages with equality of any cardinality. Using Zorn's lemma, prove that the results of problem 5.6.10 hold for languages of any cardinality. Thus, prove that the extended completeness theorem holds for languages with equality of any cardinality.

$*$ **5.6.13.** For a language $\mathbf{L}$ with equality of cardinality α, show that the cardinality of the term algebra arising in problem 5.6.10 is at most α. Conclude that any consistent set of $\mathbf{L}$-sentences has a model of cardinality at most α.

$*$ **5.6.14.** Let $\mathbf{L}$ be a countable first-order language with equality. A property P of sets of formulae is an *analytic consistency property* if the following hold:

(i) P is of finite character (see problem 3.5.12).

(ii) For every set S of formulae for which P is true, the following conditions hold:

A_0: S contains no atomic formula and its negation.

A_1: For every formula A of type a in S, P holds for $\{S, A_1\}$ and $\{S, A_2\}$.

A_2: For every formula B of type b in S, P holds either for $\{S, B_1\}$ or for $\{S, B_2\}$.

A_3: For every formula C of type c in S, P holds for $\{S, C(t)\}$ for every term t.

A_4: For every formula D of type d in S, for some constant c not in S, P holds for $\{S, D(c)\}$.

A_5 (i): For every term t, if P holds for S then P holds for $\{S, (t \doteq t)\}$.

(ii): For each n-ary function symbol f, for any terms $s_1, ..., s_n$, t_1, ..., t_n, if P holds for S then P holds for

$$\{S, (s_1 \doteq t_1) \land ... \land (s_n \doteq t_n) \supset (fs_1...s_n \doteq ft_1...t_n)\}.$$

(iii): For each n-ary predicate symbol Q (including $\doteq$), for any terms $s_1, ..., s_n$, $t_1, ..., t_n$, if P holds for S then P holds for

$$\{S, (s_1 \doteq t_1) \land ... \land (s_n \doteq t_n) \land Qs_1...s_n \supset Qt_1...t_n\}.$$

Prove that if P is an analytic consistency property and P holds for S, then S is satisfiable.

Hint: See problem 5.5.22.

$*$ **5.6.15.** Let **L** be a countable first-order language with equality. A set S of formulae is *truth-functionally inconsistent* iff S is the result of substituting first-order formulae for the propositional letters in an unsatisfiable set of propositions (see definition 5.3.11). We say that a formula A is *truth-functionally valid* iff it is obtained by substitution of first-order formulae into a tautology (see definition 5.3.11). We say that a finite set $S = \{A_1, ..., A_m\}$ *truth-functionally implies* B iff the formula

$$(A_1 \land ... \land A_m) \supset B$$

is truth-functionally valid. A property P of sets of formulae is a *synthetic consistency property* iff the following conditions hold:

(i) P is of finite character (see problem 3.5.12).

(ii) For every set S of formulae, the following conditions hold:

B_0: If S is truth-functionally inconsistent, then P does not hold for S;

B_3: If P holds for S then for every formula C of type c in S, P holds for $\{S, C(t)\}$ for every term t.

B_4: For every formula D of type d in S, for some constant c not in S, P holds for $\{S, D(c)\}$.

B_5: For every formula X, if P does not hold for $\{S, X\}$ or $\{S, \neg X\}$, then P does not hold for S. Equivalently, if P holds for S, then for every formula X, either P holds for $\{S, X\}$ or P holds for $\{S, \neg X\}$.

B_6 (i): For every term t, if P holds for S then P holds for $\{S, (t \doteq t)\}$.

(ii): For each n-ary function symbol f, for any terms $s_1, ..., s_n$, t_1, ..., t_n, if P holds for S then P holds for

$$\{S, (s_1 \doteq t_1) \land ... \land (s_n \doteq t_n) \supset (fs_1...s_n \doteq ft_1...t_n)\}.$$

(iii): For each n-ary predicate symbol Q (including $\doteq$), for any terms $s_1, ..., s_n$, $t_1, ..., t_n$, if P holds for S then P holds for

$$\{S, (s_1 \doteq t_1) \wedge ... \wedge (s_n \doteq t_n) \wedge Qs_1...s_n \supset Qt_1...t_n\}.$$

(a) Prove that if P is a synthetic consistency property then the following condition holds:

B_7: If P holds for S and a finite subset of S truth-functionally implies X, then P holds for $\{S, X\}$.

(b) Prove that every synthetic consistency property is an analytic consistency property.

(c) Prove that consistency within the Hilbert system $H_=$ of problem 5.6.6 is a synthetic consistency property.

∗ **5.6.16.** A set S of formulae is *Henkin-complete* iff for every formula $D = \exists x B$ of type d, there is some constant c such that $B(c)$ is also in S.

Prove that if P is a synthetic consistency property and S is a set of formulae that is both Henkin-complete and a maximally set for which P holds, then S is satisfiable.

Hint: Show that S is a Hintikka set for the term algebra consisting of all terms built up from function, constant symbols, and variables occurring free in S.

∗ **5.6.17.** Prove that if P is a synthetic consistency property, then every set S of formulae for which P holds can be extended to a Henkin-complete set which is a maximal set for which P holds.

Hint: Use the idea of problem 5.5.17.

Use the above property to prove the completeness of the Hilbert system $H_=$.

∗ **5.6.18.** Let **L** be a countable first-order language with equality. Prove that the results of problem 5.5.26 are still valid.

∗ **5.6.19.** Given a first-order language **L** with equality, for any **L**-structure **M**, for any finite or countably infinite sequence X of elements in M (the domain of the structure **M**), the language $\mathbf{L}_X$ is the expansion of **L** obtained by adding new distinct constants ($\{c_1, ..., c_n\}$ if $X =< a_1, ..., a_n >$, $\{c_1, ..., c_n, c_{n+1}, ...\}$ if X is countably infinite) to the set of constants in **L**. The structure $(\mathbf{M}, X)$ is the expansion of **M** obtained by interpreting each c_i as a_i.

An **L**-structure **M** is *countably saturated* if, for every finite sequence $X =< a_1, ..., a_n >$ of elements in M, for every set $\Gamma(x)$ of formulae with at most one free variable x over the expanded language $\mathbf{L}_X$, if every finite subset of $\Gamma(x)$ is satisfiable in $(\mathbf{M}, X)$ then $\Gamma(x)$ is satisfiable in $(\mathbf{M}, X)$.

(1) Prove that every finite structure $\mathbf{M}$ is countably saturated.

Hint: Show that if the conclusion does not hold, a finite unsatisfiable subset of $\Gamma(x)$ can be found.

(2) Two $\mathbf{L}$-structures $\mathbf{A}$ and $\mathbf{B}$ are *elementary equivalent* if, for any $\mathbf{L}$-sentence D,

$$\mathbf{A} \models D \quad \text{if and only if} \quad \mathbf{B} \models D.$$

(a) Assume that $\mathbf{A}$ and $\mathbf{B}$ are elementary equivalent. Show that for every formula $E(x)$ with at most one free variable x, $E(x)$ is satisfiable in $\mathbf{A}$ if and only if $E(x)$ is satisfiable in $\mathbf{B}$.

(b) Let $X =< a_1, ..., a_n >$ and $Y =< b_1, ..., b_n >$ be two finite sequences of elements in A and B respectively. Assume that $(\mathbf{A}, X)$ and $(\mathbf{B}, Y)$ are elementary equivalent and that $\mathbf{A}$ is countably saturated.

Show that for any set $\Gamma(x)$ of formulae with at most one free variable x over the expansion $\mathbf{L}_Y$ such that every finite subset of $\Gamma(x)$ is satisfiable in $(\mathbf{B}, Y)$, $\Gamma(x)$ is satisfiable in $(\mathbf{A}, X)$. (Note that the languages $\mathbf{L}_X$ and $\mathbf{L}_Y$ are identical. Hence, we will refer to this language as $\mathbf{L}_X$.)

(c) Assume that $\mathbf{A}$ and $\mathbf{B}$ are elementary equivalent, with $\mathbf{A}$ countably saturated. Let $Y =< b_1, ..., b_n, ... >$ be a countable sequence of elements in B.

Prove that there exists a countable sequence $X =< a_1, ..., a_n, ... >$ of elements from A, such that $(\mathbf{A}, X)$ and $(\mathbf{B}, Y)$ are elementary equivalent.

Hint: Proceed in the following way: Define the sequence $X_n =< a_1, ..., a_n >$ by induction so that $(\mathbf{A}, X_n)$ and $(\mathbf{B}, Y_n)$ are elementary equivalent (with $Y_n =< b_1, ..., b_n >$) as follows: Let $\Gamma(x)$ be the set of formulae over $\mathbf{L}_{Y_n}$ satisfied by b_{n+1} in $(\mathbf{B}, Y_n)$.

Show that $\Gamma(x)$ is maximally consistent. Using 2(b), show that $\Gamma(x)$ is satisfied by some a_{n+1} in $(\mathbf{A}, X_n)$ and that it is the set of formulae satisfied by a_{n+1} in $(\mathbf{A}, X_n)$ (recall that $\Gamma(x)$ is maximally consistent). Use these properties to show that $(\mathbf{A}, X_{n+1})$ and $(\mathbf{B}, Y_{n+1})$ are elementary equivalent.

(d) If $(\mathbf{A}, X)$ and $(\mathbf{B}, Y)$ are elementary equivalent as above, show that

$$a_i = a_j \quad \text{if and only if} \quad b_i = b_j \text{ for all } i, j \geq 1.$$

(e) Use (d) to prove that if **A** and **B** are elementary equivalent and A is finite, then **B** is isomorphic to **A**.

$*$ **5.6.20.** Write a computer program implementing the *search* procedure in the case of a first-order language with equality.

Notes and Suggestions for Further Reading

First-order logic is a rich subject that has been studied extensively. We have presented the basic model-theoretic and proof-theoretic concepts, using Gentzen systems because of their algorithmic nature and their conceptual simplicity. This treatment is inspired from Kleene, 1967. For more on Gentzen systems, the reader should consult Robinson, 1979; Takeuti, 1975; Szabo, 1969; or Smullyan, 1968.

We have focused our attention on a constructive proof of the completeness theorem, using Gentzen systems and Hintikka sets. 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 first-order logic. Most texts adopt Hilbert systems. Just to mention a few texts of varying degree of difficulty, Hilbert systems are discussed in Kleene, 1952; Kleene, 1967; Enderton, 1972; Shoenfield, 1967; and Monk, 1976. A more elementary presentation of first-order logic tailored for computer scientists is found in Manna and Waldinger, 1985. Natural deduction systems are discussed in Kleene, 1967; Van Dalen, 1980; Prawitz, 1965; and Szabo, 1969. A variant of Gentzen systems called the tableaux system is discussed at length in Smullyan, 1968. Type Theory (higher-order logic "a la Church"), including the Gödel incompleness theorems, is discussed in Andrews 1986.

An illuminating comparison of various approaches to the completeness theorem, including Henkin's method, can be found in Smullyan, 1968.

Since we have chosen to emphasize results and techniques dealing with the foundations of (automatic) theorem proving, model theory is only touched upon lightly. However, this is a very important branch of logic. The reader is referred to Chang and Keisler, 1973; Bell and Slomson, 1974; and Monk, 1976, for thorough and advanced expositions of model theory. The article on fundamentals of model theory by Keisler, and the article by Eklof about ultraproducts, both in Barwise, 1977, are also recommended. Barwise 1977 also contains many other interesting articles on other branches of logic.

We have not discussed the important incompleteness theorems of Gödel. The reader is referred to Kleene,1952; Kleene, 1967; Enderton, 1972; Shoenfield, 1967; or Monk, 1976. One should also consult the survey article by Smorynski in Barwise, 1977.