# Chapter 1

# Introduction

Logic is concerned mainly with two concepts: *truth* and *provability*. These concepts have been investigated extensively for centuries, by philosophers, linguists, and mathematicians. The purpose of this book is by no means to give a general account of such studies. Instead, the purpose of this book is to focus on a mathematically well defined logical system known as *first-order logic* (and, to some extent, *many-sorted logic*), and prove some basic properties of this system. In particular, we will focus on algorithmic methods for proving theorems (often referred to as *automatic theorem proving*).

Every logical system consists of a *language* used to write statements also called *propositions* or *formulae*. Normally, when one writes a formula, one has some intended *interpretation* of this formula in mind. For example, a formula may assert a true property about the natural numbers, or some property that must be true in a data base. This implies that a formula has a well-defined *meaning* or *semantics*. But how do we define this meaning precisely? In logic, we usually define the meaning of a formula as its *truth value*. A formula can be either true (or valid) or false.

Defining rigorously the notion of truth is actually not as obvious as it appears. We shall present a concept of truth due to Tarski. Roughly speaking, a formula is true if it is satisfied in all possible interpretations. So far, we have used the intuitive meaning of such words as *truth*, *interpretation*, etc. One of the objectives of this book is to define these terms rigorously, for the language of first-order logic (and many-sorted first-order logic). The branch of logic in which abstract structures and the properties true in these structures are studied is known as *model theory*.

Once the concept of truth has been defined rigorously, the next question is to investigate whether it is possible to find methods for deciding in a finite number of steps whether a formula is true (or valid). This is a very difficult task. In fact, by a theorem due to Church, there is no such general method for first-order logic.

However, there is another familiar method for testing whether a formula is true: to give a *proof* of this formula.

Of course, to be of any value, a proof system should be *sound*, which means that every provable formula is true.

We will also define rigorously the notion of proof, and proof system for first-order logic (and many-sorted first-order logic). The branch of logic concerned with the study of proof is known as *proof theory*.

Now, if we have a sound proof system, we know that every provable formula is true. Is the proof system strong enough that it is also possible to prove every true formula (of first-order logic)?

A major theorem of Gödel shows that there are logical proof systems in which every true formula is provable. This is referred to as the *completeness* of the proof system.

To summarize the situation, if one is interested in algorithmic methods for testing whether a formula of first-order logic is valid, there are two logical results of central importance: one positive (Gödel's completeness theorem), the other one negative (Church's undecidability of validity). Roughly speaking, Gödel's completeness theorem asserts that there are logical calculi in which every true formula is provable, and Church's theorem asserts that there is no decision procedure (procedure which always terminates) for deciding whether a formula is true (valid). Hence, any algorithmic procedure for testing whether a formula is true (or equivalently, by Gödel's completeness theorem, provable in a complete system) must run forever when given certain non-true formulae as input.

This book focuses on Gödel's positive result and its applications to automatic theorem proving. We have attempted to present a coherent approach to automatic theorem proving, following a main thread: Gentzen-like sequent calculi. The restriction to the positive result was dictated mostly by the lack of space. Indeed, it should be stressed that Church's negative result is also important, as well as other fundamental negative results due to Gödel. However, the omission of such topics should not be a severe inconvenience to the reader, since there are many texts covering such material (see the notes at the end of Chapter 5).

In spite of the theoretical limitation imposed by Church's result, the goal of automatic theorem proving (for short, atp) is to find *efficient* algorithmic methods for finding proofs of those formulae that are true.

A fairly intuitive method for finding such algorithms is the completeness proof for Gentzen-like sequent calculi. This approach yields a complete procedure (the *search* procedure) for proving valid formulae of first-order logic.

However, the *search* procedure usually requires an enormous amount of space and time and it is not practical. Hence, we will try improve it or find more efficient proof procedures.

For this, we will analyze the structure of proofs carefully. Fundamental results of Gentzen and Herbrand show that if a formula is provable, then it has a proof having a certain form, called a *normal form*.

The existence of such normal forms can be exploited to reduce the size of the search space that needs to be explored in trying to find a proof. Indeed, it is sufficient to look for proofs in normal form.

The existence of normal forms is also fundamental because it reduces the problem of finding a proof of a first-order formula to the problem of finding a proof of a simpler type of formula, called a proposition. Propositions are much simpler than first-order formulae. Indeed, there are algorithms for deciding truth. One of the methods based on this reduction technique is the *resolution method*, which will be investigated in Chapters 4 and 8.

Besides looking for general methods applying to the class of all true (first-order) formulae, it is interesting to consider subclasses for which simpler or more efficient proof procedures exist. Indeed, for certain subclasses there may be decision procedures. This is the case for propositions, and for quantifier-free formulae. Such cases are investigated in Chapters 3 and 10 respectively.

Unfortunately, even in cases in which algorithms exist, another difficulty emerges. A decision procedure may take too much time and space to be practical. For example, even testing whether a proposition is true may be very costly. This will be discussed in Chapter 3.

Automatic theorem proving techniques can be used by computer scientists to axiomatize structures and prove properties of programs working on these structures. Another recent and important role that logic plays in computer science, is its use as a *programming language* and as a *model of computation*. For example, in the programming language PROLOG, programs are specified by sets of assertions. In such a programming language, a computation is in fact a proof, and the output of a program is extracted from the proof. Promoters of such languages claim that since such programs are essentially logical formulae, establishing their correctness is trivial. This is not quite so, because the concept of correctness is relative, and the semantics of a PROLOG program needs to be expressed in a language other than PROLOG. However, using logic as a vehicle for programming is a very interesting idea and should be a selling point for skeptics. This use of logic will be investigated in Chapter 9.

# Chapter 2

# Mathematical Preliminaries

This chapter is devoted to mathematical preliminaries. This fairly lengthy chapter has been included in order to make this book as self-contained as possible. Readers with a firm mathematical background may skim or even skip this chapter entirely. Classroom experience shows that anyone who is not acquainted with the material included in Section 2.3 should probably spend some time reading Sections 2.1 to 2.3. In any case, this chapter can be used as a library of useful facts and may be consulted whenever necessary.

Since trees, inductive definitions and the definition of functions by recursion play an important role in logic, they will be defined carefully. First, we review some basic concepts and establish the terminology and notation used in this book. It is assumed that the reader is familiar with the basic properties of sets. For more details, the reader may consult Enderton, 1972; Enderton, 1977; Lewis and Papadimitriou, 1981; or Suppes, 1972.

## 2.1 Relations, Functions, Partial Orders, Induction

First, we review the concepts of Cartesian product, tuple and relation.

### 2.1.1 Relations

Given two sets $A$ and $B$ (possibly empty), their *Cartesian product* denoted by $A \times B$ is the set of ordered pairs

$$\{< a, b > \mid a \in A,\ b \in B\}.$$

Given any finite number of sets $A_1$,...,$A_n$, the *Cartesian product* $A_1 \times ... \times A_n$ is the set of ordered $n$-tuples

$$\{< a_1, ..., a_n > \mid a_i \in A_i, 1 \leq i \leq n\}$$

(An ordered $n$-tuple $< a_1, ..., a_n >$ is also denoted by $(a_1, ..., a_n)$.)

A *binary relation between $A$ and $B$* is any subset $R$ (possibly empty) of $A \times B$.

Given a relation $R$ between $A$ and $B$, the set

$$\{x \in A \mid \exists y \in B \ < x, y > \in R\},$$

is called the *domain* of $R$ and denoted by $dom(R)$. The set

$$\{y \in B \mid \exists x \in A \ < x, y > \in R\}$$

is called the *range* of $R$ and is denoted by $range(R)$.

When $A = B$, a relation $R$ beween $A$ and $A$ is also called a relation *on* (or *over*) $A$. We will also use the notation $xRy$ as an alternate to $(x, y) \in R$.

### 2.1.2 Partial Functions, Total Functions

A relation $R$ between two sets $A$ and $B$ is *functional* iff, for all $x \in A$, and $y, z \in B$, $(x, y) \in R$ and $(x, z) \in R$ implies that $y = z$.

A *partial function* is a triple $f =< A, G, B >$, where $A$ and $B$ are arbitrary sets (possibly empty) and $G$ is a functional relation (possibly empty) between $A$ and $B$, called the *graph* of $f$.

Hence, a partial function is a functional relation such that every argument has at most one image under $f$. The graph of a function $f$ is denoted as $graph(f)$. When no confusion can arise, a function $f$ and its graph are usually identified.

A partial function $f =< A, G, B >$ is often denoted as $f : A \rightarrow B$. For every element $x$ in the domain of a partial function $f$, the unique element $y$ in the range of $f$ such that $(x, y) \in graph(f)$ is denoted by $f(x)$. A partial function $f : A \rightarrow B$ is a *total function* iff $dom(f) = A$. It is customary to call a total function simply a function.

### 2.1.3 Composition of Relations and Functions

Given two binary relations $R$ between $A$ and $B$, and $S$ between $B$ and $C$, their *composition* denoted by $R \circ S$ is a relation between $A$ and $C$ defined by the following set of ordered pairs:

$$\{(a, c) \mid \exists b \in B, (a, b) \in R \text{ and } (b, c) \in S\}.$$

Given a set $A$, the *identity relation* of $A$ is denoted by $I_A$ and is the relation $\{(x,x) \mid x \in A\}$. Note that $I_A$ is also a total function.

Given a relation $R$ between $A$ and $B$, its *converse* is the relation between $B$ and $A$ denoted by $R^{-1}$ defined by the set

$$\{(b,a) \in B \times A \mid (a,b) \in R\}.$$

Given two partial or total functions $f : A \to B$ and $g : B \to C$, with $f = < A, G_1, B >$ and $g = < B, G_2, C >$, their composition denoted by $f \circ g$ (or $f.g$, or $fg$), is the partial or total function defined by $< A, G_1 \circ G_2, C >$. Notice that according to our notation, $f \circ g(x) = g(f(x))$, that is, $f$ is applied first. Note also that composition is associative.

### 2.1.4 Injections, Surjections, Bijections

A function $f : A \to B$ is *injective* (or *one to one*) iff, for all $x, y \in A$, $f(x) = f(y)$ implies that $x = y$.

A function $f : A \to B$ is *surjective* (or *onto*) iff, for all $y \in B$, there is some $x \in A$ such that $f(x) = y$. Equivalently, the range of $f$ is the set $B$.

A function is *bijective* iff it is both injective and surjective.

It can be shown that a function $f$ is surjective if and only if there exists a function $g : B \to A$ such that $g \circ f = I_B$. If there exists a function $g : B \to A$ such that $f \circ g = I_A$, then $f : A \to B$ is injective. If $f : A \to B$ is injective and $A \neq \emptyset$, then there exists a function $g : B \to A$ such that $f \circ g = I_A$. As a consequence, it can be shown that a function $f : A \to B$ is bijective if and only if there is a unique function $f^{-1}$ called its *inverse* such that $f \circ f^{-1} = I_A$ and $f^{-1} \circ f = I_B$.

### 2.1.5 Direct Image, Inverse Image

Given a (partial) function $f : A \to B$, for every subset $X$ of $A$, the *direct image* (or for short, *image*) of $X$ under $f$ is the set

$$\{y \in B \mid \exists x \in X,\ f(x) = y\}$$

and is denoted by $f(X)$. For every subset $Y$ of $B$, the *inverse image* of $Y$ under $f$ is the set

$$\{x \in A \mid \exists y \in Y,\ f(x) = y\}$$

and is denoted by $f^{-1}(Y)$.

*Warning*: The function $f$ may not have an inverse. Hence, $f^{-1}(Y)$ should not be confused with $f^{-1}(y)$ for $y \in B$, which is only defined when $f$ is a bijection.

### 2.1.6 Sequences

Given two sets $I$ and $X$, an *I-indexed sequence* (or *sequence*) is any function $A : I \to X$, usually denoted by $(A_i)_{i \in I}$. The set $I$ is called the *index set*. If $X$ is a set of sets, $(A_i)_{i \in I}$ is called a *family* of sets.

### 2.1.7 Natural Numbers and Countability

The set of *natural numbers* (or nonnegative integers) is denoted by $\mathbf{N}$ and is the set $\{0, 1, 2, 3, ...\}$. A set $A$ is *countable* (or *denumerable*) iff either $A = \emptyset$ or there is a surjection $h : \mathbf{N} \to A$ from $\mathbf{N}$ onto $A$, *countably infinite* iff there is a bijection $h : \mathbf{N} \to A$. Otherwise, $A$ is said to be *uncountable*. The *cardinality* of a countably infinite set is denoted by $\omega$. The set of positive integers is denoted by $\mathbf{N}_+$. For every positive integer $n \in \mathbf{N}_+$, the set $\{1, ..., n\}$ is denoted as $[n]$, and $[0]$ denotes the empty set. A set $A$ is *finite* iff there is a bijection $h : [n] \to A$ for some natural number $n \in \mathbf{N}$. The natural number $n$ is called the *cardinality* of the set $A$, which is also denoted by $|A|$. When $I$ is the set $\mathbf{N}$ of natural numbers, a sequence $(A_i)_{i \in I}$ is called a *countable sequence*, and when $I$ is some set $[n]$ with $n \in \mathbf{N}$, $(A_i)_{i \in I}$ is a *finite sequence*.

### 2.1.8 Equivalence Relations

A binary relation $R \subset A \times A$ is *reflexive* iff for all $x \in A$, $(x, x) \in R$.

The relation $R$ is *symmetric* iff for all $x, y \in A$, $(x, y) \in R$ implies that $(y, x) \in R$.

The relation $R$ is *transitive* iff for all $x, y, z \in A$, $(x, y) \in R$ and $(y, z) \in R$ implies that $(x, z) \in R$.

The relation $R$ is an *equivalence relation* if it is reflexive, symmetric and transitive. Given an equivalence relation $R$ on a set $A$, for every $x \in A$, the set $\{y \in A \mid (x, y) \in R\}$ is the *equivalence class of x modulo R* and is denoted by $[x]_R$, or $\overline{x}_R$, or simply $[x]$ or $\overline{x}$. The set of equivalence classes modulo $R$ is the *quotient of A by R* and is denoted by $A/R$. The set $A/R$ is also called a *partition* of $A$, since any two distinct equivalence classes are nonempty and disjoint, and their union is $A$ itself. The surjective function $h_R : A \to A/R$ such that $h_R(x) = [x]_R$ is called the *canonical function* associated with $R$.

Given any relation $R$ on a set $A$, we define the *powers* of $R$ as follows: For every integer $n \geq 0$,

$$R^0 = I_A, \ R^1 = R, \text{ and } R^{n+1} = R^n \circ R.$$

The union

$$R^+ = \bigcup_{n \geq 1} R^n$$

called the *transitive closure* of $R$ is the smallest transitive relation on $A$ containing $R$, and

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

is called the *reflexive and transitive closure* of $R$ and is the smallest reflexive and transitive relation on $A$ containing $R$. It is obvious that $R^+ = R \circ R^* = R^* \circ R$, and that $R^* = I_A \cup R^+$. Thus, it can also be shown that for any relation $R$ on a set $A$, $(R \cup R^{-1})^*$ is the least equivalence relation containing $R$.

### 2.1.9 Partial and Total Orders

A relation $R$ on a set $A$ is *antisymmetric* iff for all $x, y \in A$, $(x, y) \in R$ and $(y, x) \in R$ implies that $x = y$.

A relation $R$ on a set $A$ is a *partial order* iff $R$ is reflexive, transitive and antisymmetric.

Given a partial order $R$ on a set $A$, the pair $< A, R >$ is called a *partially ordered set* (or *poset*). A partial order is often denoted by the symbol $\leq$.

Given a partial order $\leq$ on a set $A$, given any subset $X$ of $A$, $X$ is a *chain* iff for all $x, y \in X$, either $x \leq y$, or $y \leq x$.

A partial order $\leq$ on a set $A$ is a *total order* (or a *linear* order) iff $A$ is a chain.

Given a partial order $\leq$ on a set $A$, given any subset $X$ of $A$, an element $b \in A$ is a *lower bound* of $X$ iff for all $x \in X$, $b \leq x$. An element $m \in A$ is an *upper bound* of $X$ iff for all $x \in X$, $x \leq m$. Note that $b$ or $m$ may or may not belong to $X$. It can be easily shown that a lower bound (resp. upper bound) of $X$ in $X$ is unique. Hence the following definition is legitimate.

An element $b \in X$ is *the least element of* $X$ iff for all $x \in X$, $b \leq x$. An element $m \in X$ is *the greatest element of* $X$ iff for all $x \in X$, $x \leq m$. In view of the above remark, least and greatest elements are unique (when they exist).

Given a subset $X$ of $A$, an element $b \in X$ is *minimal in* $X$ iff for all $x \in X$, $x \leq b$ implies that $x = b$. An element $m \in X$ is *maximal in* $X$ if for all $x \in X$, $m \leq x$ implies that $m = x$. Contrary to least and greatest elements, minimal or maximal elements are not necessarily unique.

An element $m \in A$ is *the least upper bound* of a subset $X$, iff the set of upper bounds of $X$ is nonempty, and $m$ is the least element of this set. An element $b \in A$ is *the greatest lower bound* of $X$ if the set of lower bounds of $X$ is nonempty, and $b$ is the greatest element of this set.

Although the following fundamental result known as Zorn's lemma will not be used in the main text, it will be used in some of the problems. Hence, this result is stated without proof. For details and the proof, the reader is referred to Suppes, 1972; Levy, 1979; or Kuratowski and Mostowski, 1976.

**Theorem 2.1.1** (Zorn's lemma) Given a partially ordered set $< A, \leq >$, if every (nonempty) chain in $A$ has an upper bound, then $A$ has some maximal element.

### 2.1.10 Well-Founded Sets and Complete Induction

A very general induction principle holds for the class of partially ordered sets having a well-founded ordering. Given a partial order $\leq$ on a set $A$, the *strict order* $<$ associated with $\leq$ is defined as follows:

$x < y$ if and only if $x \leq y$ and $x \neq y$.

A partially ordered set $< A, \leq >$ is *well-founded* iff it has no infinite decreasing sequence $(x_i)_{i \in \mathbf{N}}$, that is, sequence such that $x_{i+1} < x_i$ for all $i \geq 0$.

The following property of well-founded sets is fundamental.

**Lemma 2.1.1** Given a partially ordered set $< A, \leq >$, $< A, \leq >$ is a well-founded set if and only if every nonempty subset of $A$ has a minimal element.

*Proof*: First, assume that $< A, \leq >$ is well-founded. We proceed by contradiction. Let $X$ be any nonempty subset of $A$, and assume that $X$ does not have a minimal element. This means that for any $x \in X$, there is some $y \in X$ such that $y < x$, since otherwise there would be some minimal $x \in X$. Since $X$ is nonempty, there is some $x_0$ in $X$. By the above remark, there is some $x_1 \in X$ such that $x_1 < x_0$. By repeating this argument (using induction on $\mathbf{N}$), an infinite decreasing sequence $(x_i)$ can be defined in $X$, contradicting the fact that $A$ is well-founded. Hence, $X$ must have some minimal element.

Conversely, assume that every nonempty subset has a minimal element. If an infinite decreasing sequence $(x_i)$ exists in $A$, $(x_i)$ has some minimal element $x_k$. But this contradicts the fact that $x_{k+1} < x_k$. $\square$

The principle of *complete induction* (or *structural induction*) is now defined. Let $(A, \leq)$ be a well-founded poset, and let $P$ be a property of the set $A$, that is, a function $P : A \to \{\mathbf{false}, \mathbf{true}\}$. We say that $P(x)$ *holds* if $P(x) = \mathbf{true}$.

**Principle of Complete Induction**

To prove that a property $P$ holds for all $z \in A$, it suffices to show that, for every $x \in A$,

$(*)$ if $x$ is minimal, or $P(y)$ holds for all $y < x$,

$(**)$ then $P(x)$ holds.

The statement $(*)$ is called the *induction hypothesis*, and the implication for all $x$, $(*)$ implies $(**)$

is called the *induction step*. Formally, the induction principle can be stated as:

$$(CI) \qquad (\forall x \in A)[(\forall y \in A)(y < x \supset P(y)) \supset P(x)] \supset (\forall z \in A)P(z)$$

Note that if $x$ is minimal, then there is no $y \in A$ such that $y < x$, and $(\forall y \in A)(y < x \supset P(y))$ is **true**. Hence, $P(x)$ has to be shown to be **true** for every minimal element $x$. These cases are called the *base cases*. Complete induction is not valid for arbitrary posets (see the problems) but holds for well-founded sets as shown in the following lemma.

**Lemma 2.1.2** The principle of complete induction holds for every well-founded set.

*Proof*: We proceed by contradiction. Assume that $(CI)$ is false. Then,

$$(1) \qquad (\forall x \in A)[(\forall y \in A)(y < x \supset P(y)) \supset P(x)]$$

is **true** and

$$(2) \qquad (\forall z \in A)P(z)$$

is false, that is,

$$(\exists z \in A)(P(z) = \mathbf{false})$$

is **true**.

Hence, the subset $X$ of $A$ defined by

$$X = \{x \in A \mid P(x) = \mathbf{false}\}$$

is nonempty. Since $A$ is well founded, by lemma 2.1.1, $X$ has some minimal element $b$. Since (1) is **true** for all $x \in A$, letting $x = b$,

$$(3) \qquad [(\forall y \in A)(y < b \supset P(y)) \supset P(b)]$$

is **true**. If $b$ is also minimal in $A$, there is no $y \in A$ such that $y < b$ and so,

$$(\forall y \in A)(y < b \supset P(y))$$

holds trivially and (3) implies that $P(b) = \mathbf{true}$, which contradicts the fact that $b \in X$. Otherwise, for every $y \in A$ such that $y < b$, $P(y) = \mathbf{true}$, since otherwise $y$ would belong to $X$ and $b$ would not be minimal. But then,

$$(\forall y \in A)(y < b \;\supset\; P(y))$$

also holds and (3) implies that $P(b) = \mathbf{true}$, contradicting the fact that $b \in X$. Hence, complete induction is valid for well-founded sets. $\square$

As an illustration of well-founded sets, we define the *lexicographic ordering*. Given a partially ordered set $(A, \leq)$, the *lexicographic ordering* $<<$ on $A \times A$ induced by $\leq$ is defined a follows: For all $x, y, x', y' \in A$,

$$\begin{aligned}(x, y) << (x', y') &\text{ if and only if either}\\ x = x' &\text{ and } y = y', \quad \text{or}\\ x &< x' \quad \text{or}\\ x = x' &\text{ and } y < y'.\end{aligned}$$

We leave as an exercise the check that $<<$ is indeed a partial order on $A \times A$. The following lemma will be useful.

**Lemma 2.1.3** If $< A, \leq >$ is a well-founded partially ordered set, the lexicographic ordering $<<$ on $A \times A$ is also well founded.

*Proof*: We proceed by contradiction. Assume that there is an infinite decreasing sequence $(< x_i, y_i >)_{i \in \mathbf{N}}$ in $A \times A$. Then, either,

(1) There is an infinite number of distinct $x_i$, or

(2) There is only a finite number of distinct $x_i$.

In case (1), the subsequence consisting of these distinct elements forms a decreasing sequence in $A$, contradicting the fact that $\leq$ is well founded. In case (2), there is some $k$ such that for all $i \geq k$, $x_i = x_{i+1}$. By definition of $<<$, the sequence $(y_i)_{i \geq k}$ is a decreasing sequence in $A$, contradicting the fact that $\leq$ is well founded. Hence, $<<$ is well founded on $A \times A$. $\square$

As an illustration of the principle of complete induction, consider the following example in which it is shown that a function defined recursively is a total function.

**EXAMPLE 2.1.1**

(Ackermann's function) The following function $A : \mathbf{N} \times \mathbf{N} \to \mathbf{N}$ known as *Ackermann's function* is well known in recursive function theory for its extraordinary rate of growth. It is defined recursively as follows:

$$\begin{aligned}A(x, y) = if\ x = 0\ then\ y + 1&\\ else\ if\ y = 0\ then\ A(x-1, 1)&\\ else\ A(x-1, A(x, y-1))&\end{aligned}$$

It is actually not obvious that such a recursive definition defines a partial function, but this can be shown. The reader is referred to Machtey and Young, 1978; or Rogers, 1967, for more details.

We wish to prove that $A$ is a total function. We proceed by complete induction over the lexicographic ordering on $\mathbf{N} \times \mathbf{N}$.

The base case is $x = 0$, $y = 0$. In this case, since $A(0, y) = y + 1$, $A(0, 0)$ is defined and equal to 1.

The induction hypothesis is that for any $(m, n)$, $A(m', n')$ is defined for all $(m', n') << (m, n)$, with $(m, n) \neq (m', n')$.

For the induction step, we have three cases:

(1) If $m = 0$, since $A(0, y) = y + 1$, $A(0, n)$ is defined and equal to $n + 1$.

(2) If $m \neq 0$ and $n = 0$, since $(m - 1, 1) << (m, 0)$ and $(m - 1, 1) \neq (m, 0)$, by the induction hypothesis, $A(m - 1, 1)$ is defined, and so $A(m, 0)$ is defined since it is equal to $A(m - 1, 1)$.

(3) If $m \neq 0$ and $n \neq 0$, since $(m, n - 1) << (m, n)$ and $(m, n - 1) \neq (m, n)$, by the induction hypothesis, $A(m, n - 1)$ is defined. Since $(m-1, y) << (m, z)$ and $(m-1, y) \neq (m, z)$ no matter what $y$ and $z$ are, $(m - 1, A(m, n - 1)) << (m, n)$ and $(m - 1, A(m, n - 1)) \neq (m, n)$, and by the induction hypothesis, $A(m - 1, A(m, n - 1))$ is defined. But this is precisely $A(m, n)$, and so $A(m, n)$ is defined. This concludes the induction step. Hence, $A(x, y)$ is defined for all $x, y \geq 0$. $\square$

### 2.1.11 Restrictions and Extensions

We define a partial ordering $\subseteq$ on partial functions as follows: $f \subseteq g$ if and only if $graph(f)$ is a subset of $graph(g)$. We say that $g$ is an *extension* of $f$ and that $f$ is a *restriction* of $g$. The following lemma will be needed later.

**Lemma 2.1.4** Let $(f_n)_{n \geq 0}$ be a sequence of partial functions $f_n : A \to B$ such that $f_n \subseteq f_{n+1}$ for all $n \geq 0$. Then, $g = (A, \bigcup\, graph(f_n), B)$ is a partial function. Furthermore, $g$ is the least upper bound of the sequence $(f_n)$.

*Proof*: First, we show that $G = \bigcup graph(f_n)$ is functional. Note that for every $(x, y) \in G$, there is some $n$ such that $(x, y) \in graph(f_n)$. If $(x, y) \in G$ and $(x, z) \in G$, then there is some $m$ such that $(x, y) \in graph(f_m)$ and some $n$ such that $(x, z) \in graph(f_n)$. Letting $k = max(m, n)$, since $(f_n)$ is a chain, we have $(x, y) \in graph(f_k)$ and $(x, z) \in graph(f_k)$. But since $graph(f_k)$ is functional, we must have $y = z$. Next, the fact that each relation $graph(f_n)$ is contained in $G$ is obvious since $G = \bigcup graph(f_n)$. If $h$ is any partial function such that $graph(f_n)$ is a subset of $graph(h)$ for all $n \geq 0$, by definition of a union, $G = \bigcup graph(f_n)$ is a subset of $h$. Hence, $g$ is indeed the least upper bound of the chain $(f_n)$. $\square$

### 2.1.12 Strings

Given any set $A$ (even infinite), a *string* over $A$ is any finite sequence $u : [n] \to A$, where $n$ is a natural number. It is customary to call the set $A$ an *alphabet*.

Given a string $u : [n] \to A$, the natural number $n$ is called the *length* of $u$ and is denoted by $|u|$. For $n = 0$, we have the string corresponding to the unique function from the empty set to $A$, called the *null string* (or *empty string*), and denoted by $e_A$, or for simplicity by $e$ when the set $A$ is understood. Given any set $A$ (even infinite), the set of all strings over $A$ is denoted by $A^*$. If $u : [n] \to A$ is a string and $n > 0$, for every $i \in [n]$, $u(i)$ is some element of $A$ also denoted by $u_i$, and the string $u$ is also denoted by $u_1 ... u_n$.

Strings can be *concatenated* as follows. Given any two strings $u : [m] \to A$ and $v : [n] \to A$, $(m, n \geq 0)$, their *concatenation* denoted by $u.v$ or $uv$ is the string $w : [m + n] \to A$ such that:

$$w(i) = \begin{cases} u(i) & \text{if } 1 \leq i \leq m; \\ v(i - m) & \text{if } m + 1 \leq i \leq m + n. \end{cases}$$

One verifies immediately that for every string $u$, $u.e = e.u = u$. In other words, viewing concatenation as an algebraic operation on the set $A^*$ of all strings, $e$ is an identity element. It is also obvious that concatenation is associative, but not commutative in general.

Given a string $u$, a string $v$ is a *prefix* (or *head*) of $u$ if there is a string $w$ such that $u = vw$. A string $v$ is a *suffix* (or *tail*) of $u$ if there is a string $w$ such that $u = wv$. A string $v$ is a *substring* of $u$ if there are strings $x$ and $y$ such that $u = xvy$. A prefix $v$ (suffix, substring) of a string $u$ is *proper* if $v \neq u$.

## 2.2 Tree Domains and Trees

In order to define finite or infinite trees, we use the concept of a *tree domain* due to Gorn (Gorn, 1965).

### 2.2.1 Tree Domains

A *tree domain* $D$ is a nonempty subset of strings in $\mathbf{N}_+^*$ satisfying the conditions:

(1) For each $u \in D$, every prefix of $u$ is also in $D$.

(2) For each $u \in D$, for every $i \in \mathbf{N}_+$, if $ui \in D$ then, for every $j$, $1 \leq j \leq i$, $uj$ is also in $D$.

**EXAMPLE 2.2.1**

The tree domain

$$D = \{e, 1, 2, 11, 21, 22, 221, 222, 2211\}$$

is represented as follows:

$$\begin{tikzcd}
 & & e \arrow[dl] \arrow[dr] & & & \\
 & 1 \arrow[dl] & & 2 \arrow[dl] \arrow[dr] & & \\
11 & & 21 & & 22 \arrow[dl] \arrow[dr] & \\
 & & & 221 \arrow[d] & & 222 \\
 & & & 2211 & &
\end{tikzcd}$$

### 2.2.2 Trees

Given a set $\Sigma$ of labels, a $\Sigma$-tree (for short, a *tree*) is a total function $t : D \to \Sigma$, where $D$ is a tree domain.

The domain of a tree $t$ is denoted by $dom(t)$. Every string $u$ in $dom(t)$ is called a *tree address* or a *node*.

**EXAMPLE 2.2.2**

Let $\Sigma = \{f, g, h, a, b\}$. The tree $t : D \to \Sigma$, where $D$ is the tree domain of example 2.2.1 and $t$ is the function whose graph is

$$\{(e, f), (1, h), (2, g), (11, a), (21, a), (22, f), (221, h), (222, b), (2211, a)\}$$

is represented as follows:

$$\begin{tikzcd}
 & & f \arrow[dl] \arrow[dr] & & & \\
 & h \arrow[dl] & & g \arrow[dl] \arrow[dr] & & \\
a & & a & & f \arrow[dl] \arrow[dr] & \\
 & & & h \arrow[d] & & b \\
 & & & a & &
\end{tikzcd}$$

The *outdegree* (sometimes called *ramification*) $d(u)$ of a node $u$ is the cardinality of the set $\{i \mid ui \in dom(t)\}$. Note that the outdegree of a node can be infinite. Most of the trees that we shall consider will be *finite-branching*, that is, for every node $u$, $d(u)$ will be an integer, and hence finite. A node of outdegree 0 is called a *leaf*. The node whose address is $e$ is called the *root* of the tree. A tree is *finite* if its domain $dom(t)$ is finite. Given a node $u$ in $dom(t)$, every node of the form $ui$ in $dom(t)$ with $i \in \mathbf{N}_+$ is called a *son* (or *immediate successor*) of $u$.

Tree addresses are totally ordered *lexicographically* as follows: $u \leq v$ if either $u$ is a prefix of $v$ or, there exist strings $x, y, z \in \mathbf{N}_+^*$ and $i, j \in \mathbf{N}_+$, with $i < j$, such that $u = xiy$ and $v = xjz$. In the first case, we say that $u$ is an *ancestor* (or *predecessor*) of $v$ (or $u$ *dominates* $v$) and in the second case, that $u$ is *to the left* of $v$. If $y = e$ and $z = e$, we say that $xi$ is a *left brother* (or *left sibling*) of $xj$, $(i < j)$. Two tree addresses $u$ and $v$ are *independent* if $u$ is not a prefix of $v$ and $v$ is not a prefix of $u$.

### 2.2.3 Paths

A *finite path* with *source* $u$ and *target* $v$ is a finite sequence of nodes $u_0$,$u_1$,...,$u_n$ such that $u_0 = u$, $u_n = v$, and for all $j$, $1 \leq j \leq n$, $u_j = u_{j-1}i_j$ for some $i_j \in \mathbf{N}_+$. The *length of a path* $u_0, u_1, ..., u_n$ is $n$ $(n \geq 0)$. When $n = 0$, we have the *null path* from $u$ to $u$ (of length 0). A *branch* (or *chain*) is a path from the root to a leaf. An *infinite path* with *source* $u$ is an infinite sequence of nodes $u_0$,$u_1$,...,$u_n$,..., such that $u_0 = u$ and, for all $j \geq 1$, $u_j = u_{j-1}i_j$ for some $i_j \in \mathbf{N}_+$.

Given a finite tree $t$, the *height* of a node $u$ in $dom(t)$ is equal to $max(\{length(p) \mid p \text{ is a path from } u \text{ to a leaf}\})$. The *depth* of a finite tree is the height of its root (the length of a longest path from the root to a leaf).

### 2.2.4 Subtrees

Given a tree $t$ and a node $u$ in $dom(t)$, the *subtree rooted at* $u$ (also called *scope*) is the tree $t/u$ whose domain is the set $\{v \mid uv \in dom(t)\}$ and such that $t/u(v) = t(uv)$ for all $v$ in $dom(t/u)$.

Another important operation is the operation of tree replacement (or tree substitution).

### 2.2.5 Tree Replacement

Given two trees $t_1$ and $t_2$ and a tree address $u$ in $t_1$, the *result of replacing* $t_2$ at $u$ in $t_1$, denoted by $t_1[u \leftarrow t_2]$, is the function whose graph is the set of pairs

$$\{(v, t_1(v)) \mid u \text{ is not a prefix of } v\} \cup \{(uv, t_2(v)\}.$$

**EXAMPLE 2.2.3**

Let $t_1$ and $t_2$ be the trees defined by the following diagrams:

**Tree $t_1$**

$$\begin{tikzcd}[column sep=small]
 & & f \arrow[dl] \arrow[dr] & & & \\
 & h \arrow[dl] & & g \arrow[dl] \arrow[dr] & & \\
a & & a & & f \arrow[dl] \arrow[dr] & \\
 & & & h \arrow[d] & & b \\
 & & & a & &
\end{tikzcd}$$

**Tree $t_2$**

$$\begin{tikzcd}[column sep=small]
 & g \arrow[dl] \arrow[dr] & \\
a & & b
\end{tikzcd}$$

The tree $t_1[22 \leftarrow t_2]$ is defined by the following diagram:

$$\begin{tikzcd}[column sep=small]
 & & f \arrow[dl] \arrow[dr] & & & \\
 & h \arrow[dl] & & g \arrow[dl] \arrow[dr] & & \\
a & & a & & g \arrow[dl] \arrow[dr] & \\
 & & & a & & b
\end{tikzcd}$$

### 2.2.6 Ranked Alphabets and $\Sigma$-Trees

In many situations, it is desirable to have a standard set of symbols to name operations taking a specified number of arguments. Such a set is called a *ranked alphabet* (or *simply stratified alphabet*, or *signature*).

A *ranked alphabet* is a set $\Sigma$ together with a *rank function* $r : \Sigma \to \mathbf{N}$. Every symbol $f \in \Sigma$ has a *rank* (or *arity*) $r(f)$ indicating the fixed number of arguments of $f$. Symbols of arity 0 are also called *constants*. For every $n \geq 0$, $\Sigma_n$ denotes the subset of $\Sigma$ consisting of the function symbols of rank $n$.

If the set $\Sigma$ of labels is a ranked alphabet, a $\Sigma$-*tree* is a function $t : dom(t) \to \Sigma$ as before, with the additional requirement that for every node $u$ in $dom(t)$, $d(u) = r(t(u))$. In other words, the outdegree of a node is equal to the rank of its label.

**EXAMPLE 2.2.4**

Let $\Sigma = \{a, b, +, *\}$, where $a, b$ have rank 0, and $+, *$ have rank 2. The following is a $\Sigma$-tree:

$$\begin{tikzcd}
& + \arrow[dl] \arrow[dr] & & \\
a & & * \arrow[dl] \arrow[dr] & \\
& a & & b
\end{tikzcd}$$

The set of all $\Sigma$-trees is denoted by $CT_\Sigma$ and the set of all finite trees by $T_\Sigma$. Every one-node tree labeled with a constant $a$ is also denoted by $a$.

## 2.3 Inductive Definitions

Most objects used in logic or computer science are *defined inductively*. By this we mean that we frequently define a set $S$ of objects as:

The smallest set of objects containing a given set $X$ of atoms, and closed under a given set $F$ of constructors.

The purpose of this section is to define rigorously what the above sentence means.

### 2.3.1 Inductive Closures

Let us begin with an example.

**EXAMPLE 2.3.1**

Let $V = \{x_0, x_1, ...\}$ be a countable set of variables, let $X = V \cup \{0, 1\}$, let $+$ and $*$ two binary function symbols, let "(" denote the left parenthesis and ")" the right parenthesis. We wish to define the set $EXPR$ of arithmetic expressions defined using the variables in $V$, the constants 0,1, and the operators $+$ and $*$. The following definition is often given:

An arithmetic expression $E$ is one of the following expressions:

(1) A variable in $V$, or 0, or 1;

(2) If $E_1$ and $E_2$ are arithmetic expressions, then so are $(E_1 + E_2)$ and $(E_1 * E_2)$;

(3) An expression is an arithmetic expression only if it is obtained by applications of clauses (1) and (2).

In such a definition called an *inductive definition*, clause (1) defines the atoms, clause (2) asserts some closure conditions, and clause (3) is supposed to assert that the set $EXPR$ of arithmetic expressions is the smallest set of expressions containing the atoms and closed under the operations described in (2). However, it is by no means clear that (1),(2),(3) really define a set, and that this set is the smallest set having properties defined by clauses (1) and (2).

The problem with the above definition is that the universe of all possible expressions is not defined, and that the operations defined by (2) are not clearly defined either. This can be remedied as follows. Let $\Sigma$ be the alphabet $V \cup \{0, 1, (, ), +, *\}$, and $A = \Sigma^*$ be the set of all strings over $\Sigma$. The set $A$ is the universe of all possible expressions. Note that $A$ contains a lot of expressions that are not arithmetic expressions, and the purpose of the above inductive definition is to define the subset $EXPR$ of $\Sigma^*$ describing exactly all arithmetic expressions. We define the functions $H_+$ and $H_*$ from $A \times A$ to $A$ as follows: For all strings $u, v \in A$,

$$H_+(u, v) = (u + v)$$

$$H_*(u, v) = (u * v)$$

The string $(u + v)$ is the string obtained by concatenating the symbol "(", the string $u$, the symbol $+$, the string $v$, and the symbol ")", and similarly for the string $(u * v)$. Also, note that $H_+$ and $H_*$ are defined for all strings in $A$, and not just legal arithmetic expressions. For example, if $u = 0$ and $v = *)$, $H_+(u, v) = (0 + *))$, which is not a legal arithmetic expression.

We say that a subset $Y$ of $A$ is *closed under* $H_+$ and $H_*$ if for all $u, v \in Y$, $H_+(u, v) \in Y$ and $H_*(u, v) \in Y$. We are now in a position to give a precise definition of the set $EXPR$ of arithmetic expressions. We define $EXPR$ as the least subset of $A$ containing $X$ and closed under $H_+$ and $H_*$. The only remaining problem is that we have not shown that such a set actually exists. This can be shown in two ways that turn out to be equivalent as we will prove shortly. The first method which might be called a top-down method, is to observe that:

(1) The family $\mathcal{C}$ of all subsets $Y$ of $A$ that contain $X$ and are closed under $H_+$ and $H_*$ is nonempty, since $A$ satisfies these properties;

(2) Given any family of subsets of $A$ containing $X$ and closed under $H_+$ and $H_*$, the intersection of this family also contains $X$ and is closed under $H_+$ and $H_*$.

Hence, the least subset $X^+$ of $A$ containing $X$ and closed under $H_+$ and $H_*$ is the intersection of the family $\mathcal{C}$.

The bottom-up method is to define a sequence $EXPR_i$ of subsets of $A$ by induction as follows:

$$EXPR_0 = V \cup \{0, 1\};$$

$$EXPR_{i+1} = EXPR_i \cup \{H_+(u, v), H_*(u, v) | u, v \in EXPR_i\}, \text{ for } i \geq 0.$$

We let $X_+ = \bigcup EXPR_i$. We shall show below that $X_+ = X^+$ and therefore, $EXPR$ is equal to $X_+$.

Generalizing the method described in example 2.3.1, we give the following general definition.

Let $A$ be a set, $X \subset A$ a subset of $A$, and $F$ a set of functions $f : A^n \rightarrow A$, each having some arity $n > 0$. We say that a set $Y$ is *inductive on* $X$, iff $X$ is a subset of $Y$ and $Y$ is *closed under the functions in* $F$, that is: For every function $f : A^n \rightarrow A$ in $F$, for every $y_1, ..., y_n \in Y$, $f(y_1, ..., y_n)$ is also in $Y$. Clearly, $A$ itself is inductive on $X$. The intersection of all inductive sets on $X$ is also closed under $F$ and it is called the *inductive closure of* $X$ *under* $F$. Let us denote the inductive closure of $X$ by $X^+$.

If $X$ is nonempty, since every inductive set on $X$ contains $X$ and there is at least one inductive set on $X$ (namely $A$), $X^+$ is nonempty. Note that $X^+$ is the *least inductive set* containing $X$. The above definition is what we might call a top-down definition. Frequently, $X^+$ is called the least set containing $X$ and closed under $F$. There is also a bottom-up and more constructive way of characterizing $X^+$. The sequence of sets $(X_i)_{i \geq 0}$ is defined by induction as follows:

$X_0 = X$ and

$X_{i+1} = X_i \cup \{f(x_1, ..., x_n) \mid f \in F, x_1, ..., x_n \in X_i, n = r(f)\}$.

It is clear that $X_i \subseteq X_{i+1}$ for all $i \geq 0$. Let

$$X_+ = \bigcup_{i \geq 0} X_i.$$

**Lemma 2.3.1** $X^+ = X_+$.

*Proof*: First we show that $X_+$ is inductive on $X$. Since $X_0 = X$, $X_+$ contains $X$. Next, we show that $X_+$ is closed under $F$. For every $f$ in $F$ of arity $n > 0$ and for all $x_1, ..., x_n \in X_+$, by definition of $X_+$ there is some $i$ such that $x_1, ..., x_n$ are all in $X_i$, and since $f(x_1, ..., x_n) \in X_{i+1}$ (by definition), $f(x_1, ..., x_n) \in X_+$. Since $X_+$ is inductive on $X$ and $X^+$ is the least inductive set containing $X$, $X^+$ is a subset of $X_+$.

To prove that $X_+$ is a subset of $X^+$, we prove by induction that $X_i$ is a subset of $X^+$ for every $i \geq 0$. But this is obvious since $X^+$ is closed under $F$. Hence, we have shown that $X^+ = X_+$. $\square$

The following *induction principle* for inductive sets is very useful:

**Induction Principle for Inductive Sets**

If $X_+$ is the inductive closure of $X$ under $F$, for every subset $Y$ of $X_+$, if $Y$ contains $X$ and is closed under $F$, then $Y = X_+$.

**Lemma 2.3.2** The induction principle for inductive sets holds.

*Proof*: By hypothesis, $Y$ is inductive on $X$. By lemma 2.3.1, $X_+ = X^+$ which is the least inductive set containing $X$. Hence, $X_+$ is a subset of $Y$. But $Y$ is contained in $X_+$, so $Y = X_+$. □

As an illustration of the induction principle, we prove that every arithmetic expression in $EXPR$ has the same number of left and right parentheses. Let $Y$ be the subset of $EXPR$ consisting of all expressions having an equal number of left and right parentheses. Note that $Y$ contains $X$ since neither the variables nor 0 nor 1 contain parentheses. $Y$ is closed under $H_+$ and $H_*$ since these function introduce matching parentheses. Hence, by the induction principle, $Y = EXPR$.

### 2.3.2 Freely Generated Sets

One frequently needs to define functions *recursively* over an inductive closure. For example, one may want to define the process of evaluating arithmetic expressions.

**EXAMPLE 2.3.2**

Let $E$ be the arithmetic expression $((x_0 + 1) * x_1)$. Assume that we want to evaluate the value of the expression $E$ for the assignment to the variables given by $x_0 = 2$, $x_1 = 3$. Naturally, one will first compute the value of $(x_0+1)$, which is $(2+1) = 3$, and then the value of $((x_0+1)*x_1)$ which is $(3 * 3) = 9$. Suppose that we now make the problem slightly more complicated. We want a method which, given any assignment $v : V \cup \{0, 1\} \to \mathbf{N}$ of natural numbers to the variables such that $v(0) = 0$ and $v(1) = 1$, allows us to evaluate any expression $E$. The method is to evaluate expressions *recursively*. This means that we define the function $\widehat{v} : EXPR \to \mathbf{N}$ such that:

(0) $\widehat{v}(E) = v(x_i)$, if $E$ is the variable $x_i$; $\widehat{v}(0) = 0$, $\widehat{v}(1) = 1$;

(1) $\widehat{v}(E) = \widehat{v}(E_1) + \widehat{v}(E_2)$, if $E$ is $(E_1 + E_2)$;

(2) $\widehat{v}(E) = \widehat{v}(E_1) * \widehat{v}(E_2)$, if $E$ is $(E_1 * E_2)$.

Note that $\widehat{v}$ is an extension of $v$, and in fact, it can be shown that it is the unique extension of $v$ satisfying (1) and (2). However, it is not obvious that there is a function $\widehat{v}$ satisfying (0),(1),(2), and if such a function exists, it is not clear that it is unique. The existence and uniqueness of the function $\widehat{v}$ is a consequence of special properties of the inductive closure $EXPR$. In fact, given an inductive closure $X_+$ defined by a set $X$ and a set $F$ of functions, it is not always possible to define recursively a function extending a given function $v : X \to B$ (for some set $B$). We refer the reader to the problems of this chapter for a counter example. It turns out that functions are properly defined by recursion on an inductive closure exactly when this inductive closure is freely generated. The set $EXPR$ of expressions happens to be freely generated, and this is the reason functions are well defined by recursion.

To give an intuitive explanation of what freely generated means, observe that the bottom-up definition of $X_+$ suggests that each element of $X_+$ can be represented by a set of trees. Indeed, each atom, that is, each element $x$ of $X$, is represented by the one-node tree labeled with that element, and each element $a = f(x_1, ..., x_n) \in X_{k+1}$ is represented by all trees of the form $f(t_1, ..., t_n)$, where each subtree $t_i$ is any of the trees representing $x_i$. Each element of $X_+$ is usually represented by many different trees.

Roughly speaking, an inductive closure $X_+$ is freely generated by $X$ and $F$ if every element $a$ of $X_+$ is represented by a *unique tree*.

**EXAMPLE 2.3.3**

Let $A = \{a, b, c\}$ and $* : A \times A \to A$ be the function defined by the following multiplication table:

| $*$ | $a$ | $b$ | $c$ |
|---|---|---|---|
| $a$ | $a$ | $b$ | $c$ |
| $b$ | $b$ | $c$ | $a$ |
| $c$ | $c$ | $a$ | $b$ |

Since $c = *(b, b)$ and $a = *(*(b, b), b)$, the inductive closure of $X = \{b\}$ is $A$. The element $a$ is represented by the trees $*(b, c)$, $*(*(b, b), b)$, $*(*(a, b), c)$, $*(*(a, b), *(b, b))$, and in fact by infinitely many trees. As a consequence, $A$ is not freely generated by $X$.

Technically, the definition of free generation is as follows.

Let $A$ be a set, $X$ a subset of $A$, $F$ a set of functions on $A$, and $X_+$ the inductive closure of $X$ under $F$. We say that $X_+$ is *freely generated by $X$ and $F$* if the following conditions hold:

(1) The restriction of every function $f : A^m \to A$ in $F$ to $X_+^m$ is injective.

(2) For every $f : A^m \to A$, $g : A^n \to A$ in $F$, $f(X_+^m)$ is disjoint from $g(X_+^n)$ whenever $f \neq g$.

(3) For every $f : A^m \to A$ in $F$ and every $(x_1, ..., x_m) \in X_+^m$, $f(x_1, ..., x_m) \notin X$.

Let $X_{-1} = \emptyset$. We now show the following lemma.

**Lemma 2.3.3** If $X_+$ is freely generated by $X$ and $F$, then for every $i \geq 0$, $X_{i-1} \neq X_i$ and $f(x_1, ..., x_n) \notin X_i$, for every $f$ in $F$ of arity $n$ and every $(x_1, ..., x_n) \in X_i^n - X_{i-1}^n$.

*Proof*: We proceed by induction on $i \geq 0$. This is obvious for $i = 0$ since $X_{-1} = \emptyset$, $X_0 = X$ and by condition (3). For $i > 0$, we prove by induction on $k$, $0 \leq k \leq i$, that if $(x_1, ..., x_n) \in X_i^n - X_{i-1}^n$, then $f(x_1, ..., x_n) \notin X_k$. For $k = 0$, this follows from condition (3). Now, assume that if $(x_1, ..., x_n) \in X_i^n - X_{i-1}^n$, then $f(x_1, ..., x_n) \notin X_k$, for $0 \leq k \leq i - 1$. If $f(x_1, ..., x_n) \in X_{k+1}$, then

$f(x_1, ..., x_n) \in X_{k+1} - X_k$. By condition (2) and the definition of $X_{k+1}$, there is some $(y_1, ..., y_n)$ in $X_k^n$ such that $f(x_1, ..., x_n) = f(y_1, ..., y_n)$. Since $f$ is injective on $X_+^n$, we have $x_m = y_m$ for $1 \leq m \leq n$. Hence, we have $(x_1, ..., x_n) \in X_k^n$ for $k < i$, contradicting the hypothesis that $(x_1, ..., x_n) \in X_i^n - X_{i-1}^n$. Therefore, $f(x_1, ..., x_n) \notin X_{k+1}$, establishing the induction step on $k$. But this also shows that $X_i \neq X_{i+1}$, concluding the induction step on $i$. $\square$

It should be noted that conditions (1),(2),(3) apply to the *restrictions* of the functions in $F$ to $X_+$. Indeed, there are cases in which the functions in $F$ are not injective on $A$, and $f(A^m) \cap g(A^n) \neq \emptyset$ for distinct functions $f$, $g$, but conditions (1),(2),(3) hold and $X_+$ is freely generated. See problem 3.2.5. Lemma 2.3.3 can be used to formalize the statement that $X_+$ is freely generated by $X$ and $F$ iff every element has a unique tree representation. However, in order to define precisely what *representation by trees* means, it is necessary to show that trees are freely generated, and to define a function from trees to $X_+$ using theorem 2.3.1 proved next. For details of this representation, the reader is referred to the problems.

In logic, terms, formulae, and proofs are given by inductive definitions. Another important concept is that of a function defined recursively over an inductive set freely generated.

### 2.3.3 Functions Defined Recursively over Freely Generated Inductive Sets

Let $A$ be a nonempty set, $X$ a subset of $A$, $F$ a set of functions on $A$, and $X_+$ the inductive closure of $X$ under $F$. Let $B$ be any nonempty set, and let $G$ be a set of functions over the set $B$, such that there is a function $d : F \to G$ that associates with each function $f$ of arity $n$ in $F$, the function $d(f) : B^n \to B$ in $G$ ($d$ need not be a bijection).

**Theorem 2.3.1** (Unique homomorphic extension theorem) If $X_+$ is freely generated by $X$ and $F$, for every function $h : X \to B$, there is a unique function $\widehat{h} : X_+ \to B$ such that:

(1) For all $x \in X$, $\widehat{h}(x) = h(x)$;

For every function $f$ of arity $n > 0$ in $F$, for every $x_1, ..., x_n \in X_+^n$,

(2) $\widehat{h}(f(x_1, ..., x_n)) = g(\widehat{h}(x_1), ..., \widehat{h}(x_n))$, where $g = d(f)$.

The diagram below illustrates the fact that $\widehat{h}$ extends $h$. The function $\eta$ is the inclusion function of $X$ into $X_+$.

$$\begin{tikzcd} X \arrow[r, "\eta"] \arrow[dr, "h"'] & X_+ \arrow[d, "\widehat{h}"] \\ & B \end{tikzcd}$$

The identities (1) and (2) mean that $\widehat{h}$ is a *homomorphism*, which is often called the *unique homomorphic extension* of $h$. Clause (2) can be described by the following commutative diagram:

$$\begin{array}{ccc} X_+^n & \xrightarrow{f} & X_+ \\ {\scriptstyle \widehat{h}^n}\downarrow & & \downarrow{\scriptstyle \widehat{h}} \\ B^n & \xrightarrow[d(f)]{} & B \end{array}$$

In the above diagram, the function $\widehat{h}^n$ is defined by $\widehat{h}^n(x_1,...,x_n) = (\widehat{h}(x_1),...,\widehat{h}(x_n))$. We say that this diagram is *commutative* if the composition $f \circ \widehat{h}$ is equal to the composition $\widehat{h}^n \circ g$.

*Proof*: We define by induction a sequence of functions $h_i : X_i \to B$ satisfying conditions (1) and (2) restricted to $X_i$. We set $h_0 = h$. Given $h_i$, let $h_{i+1}$ have the graph:

$$\{(f(x_1,...,x_n), g(h_i(x_1),...,h_i(x_n))) \mid (x_1,...,x_n) \in X_i^n - X_{i-1}^n, f \in F\} \cup \\ graph(h_i)$$

(with g=d(f).)

We have to check that this graph is indeed functional. Since $X_+$ is freely generated, by lemma 2.3.3, $f(x_1,...,x_n) \in X_{i+1} - X_i$ whenever $(x_1,...,x_n) \in X_i^n - X_{i-1}^n$, $(i \geq 0)$, and we only have to check functionality for the first part of the union. Since the elements of $G$ are functions, by lemma 2.3.3, the only possibility for having $(x,y) \in graph(h_i)$ and $(x,z) \in graph(h_i)$ for some $x \in X_{i+1} - X_i$, is to have $x = f(x_1,...,x_m) = f'(y_1,...,y_n)$ for some $(x_1,...,x_m) \in X_i^m - X_{i-1}^m$, $(y_1,...,y_n) \in X_i^n - X_{i-1}^n$ and for some constructors $f$ and $f'$ in $F$. Since $f(X_+^m)$ and $f'(X_+^n)$ are disjoint whenever $f \neq f'$, $f(x_1,...,x_m) = f'(y_1,...,y_n)$ implies that $f = f'$ and $m = n$. Since every $f \in F$ is injective on $X_+^n$, we must also have $x_j = y_j$ for every $j$, $1 \leq j \leq n$. But then, $y = z = g(x_1,...,x_n)$, with $g = d(f)$, showing functionality. Using lemma 2.1.4, $\widehat{h} = \bigcup_{i \geq 0} h_i$ is a partial function. Since $dom(\widehat{h}) = \bigcup dom(h_i) = \bigcup X_i = X_+$, $\widehat{h}$ is total on $X_+$. Furthermore, it is clear by definition of the $h_i$ that $\widehat{h}$ satisfies (1) and (2). To prove that $\widehat{h}$ is unique, for any other function $h'$ satisfying (1) and (2), it can be easily shown by induction that $\widehat{h}$ and $h'$ agree on $X_i$ for all $i \geq 0$. This proves the theorem. $\square$

**EXAMPLE 2.3.4**

Going back to example 2.3.2, the set $A$ is $\Sigma^*$, the set $F$ of functions is $\{H_+, H_*\}$, the set $B$ is $\mathbf{N}$, the set $G$ consists of *addition* and *multiplication* on the natural numbers, and the function $d : F \to G$ is given by $d(H_+) = addition$ and $d(H_*) = multiplication$. It can be shown that $EXPR$ is freely generated by $V \cup \{0,1\}$ and $\{H_+, H_*\}$, but this is not obvious. Indeed, one has to prove rigorously conditions (1),(2),(3), for the functions $H_+$ and $H_*$, and this requires some work. A proof can be given by adapting the method used in theorem 3.2.1, and we leave it as an exercise. Since $EXPR$ is freely generated, for any function $v : V \cup \{0,1\} \to \mathbf{N}$ such that $v(0) = 0$ and $v(1) = 1$, by theorem 2.3.1, there is a unique function $\widehat{v}$ extending $v$ which is a homomorphism.

Later on when we define satisfaction in first-order logic, we will need to define the concept of a *structure*, and we will have to reformulate slightly the notion of an inductive closure. This can be done conveniently by introducing the concept of an *algebra*. Since this material is only used in Chapter 5 and Chapter 10, it has been included in an appendix.

## PROBLEMS

**2.1.1.** Show the following properties:

(a) If there exists a function $g : B \to A$ such that $f \circ g = I_A$, then $f : A \to B$ is injective. If $f : A \to B$ is injective and $A \neq \emptyset$, then there exists a function $g : B \to A$ such that $f \circ g = I_A$.

(b) A function $f$ is surjective if and only if there exists a function $g : B \to A$ such that $g \circ f = I_B$.

(c) A function $f : A \to B$ is bijective if and only if there is a function $f^{-1}$ called its *inverse* such that $f \circ f^{-1} = I_A$ and $f^{-1} \circ f = I_B$.

**2.1.2.** Prove that a function $f : A \to B$ is injective if and only if, for all functions $g, h : C \to A$, $g \circ f = h \circ f$ implies that $g = h$. A function $f : A \to B$ is surjective if and only if for all functions $g, h : B \to C$, $f \circ g = f \circ h$ implies that $g = h$.

**2.1.3.** Given a relation $R$ on a set $A$, prove that $R$ is transitive if and only if $R \circ R$ is a subset of $R$.

**2.1.4.** Given two equivalence relations $R$ and $S$ on a set $A$, prove that if $R \circ S = S \circ R$, then $R \circ S$ is the least equivalence relation containing $R$ and $S$.

**2.1.5.** Prove that $R^+ = \bigcup_{n \geq 1} R^n$ is the smallest transitive relation on $A$ containing $R$, and $R^* = \bigcup_{n \geq 0} R^n$ is the smallest reflexive and transitive relation on $A$ containing $R$. Prove that for any relation $R$ on a set $A$, $(R \cup R^{-1})^*$ is the least equivalence relation containing $R$.

**2.1.6.** Show that complete induction is not valid for posets that are not well-founded by giving a counter example.

**2.1.7.** Let $(A, \leq)$ and $(B, \leq')$ be two partially ordered sets. A function $f : A \to B$ is *monotonic* if, for all $a, b \in A$, $a \leq b$ implies $f(a) \leq' f(b)$.

(a) Show that the composition of monotonic functions is monotonic.

(b) Show that if $f$ is monotonic and $m$ is the least element of a subset $S$ of $A$, then $f(m)$ is the least element of $f(S)$.

(c) Give a counter example showing that if $f$ is monotonic and $m$ is the least element of $A$, then $f(m)$ is not necessarily the least element of $B$. Give a counter example showing that if $m$ is a minimal element of $S$, then $f(m)$ is not necessarily a minimal element of $f(S)$, even if $f$ is surjective.

$*$ **2.1.8.** Given a set $A$, a *multiset* over $A$ is an unordered collection of elements of $A$ that may have multiple occurrences of identical elements. Formally, a multiset over $A$ may be defined as a function $M : A \to \mathbf{N}$, where $\mathbf{N}$ denotes the set of nonnegative integers. An element $a$ in $A$ has exactly $n$ occurrences in $M$ iff $M(a) = n$. In particular, $a$ does not belong to $M$ iff $M(a) = 0$. Let $M(A)$ denote the set of *finite* multisets over $A$, that is, the set of functions $M : A \to \mathbf{N}$ such that $M(a) \neq 0$ only for finitely many $a \in A$. Two (finite) multisets $M$ and $M'$ are equal iff every element occurring exactly $n$ times in $A$ also occurs exactly $n$ times in $B$.

Multiset union and multiset difference is defined as follows: Given two multisets $M_1 : A \to \mathbf{N}$ and $M_2 : A \to \mathbf{N}$, their union is the multiset $M : A \to \mathbf{N}$, such that for all $x \in A$, $M(x) = M_1(x) + M_2(x)$. The *union* of $M_1$ and $M_2$ is also denoted as $M_1 \cup M_2$. The *difference* of $M_1$ and $\mathrm{M}_2$ is the multiset $M : A \to \mathbf{N}$ such that for all $x \in A$, $M(x) = M_1(x) - M_2(x)$ if $M_1(x) \geq M_2(x)$, $M(x) = 0$ otherwise. The difference of $M_1$ and $M_2$ is also denoted as $M_1 - M_2$. A multiset $M_1$ is a *submultiset* of a multiset $M_2$ if for all $x \in A$, $M_1(x) \leq M_2(x)$.

If $A$ is partially ordered by $\preceq$, the relation $<<$ on the set $M(A)$ of finite multisets is defined as follows:

$M << M'$ iff $M$ is obtained from $M'$ by removing zero or more elements from $M'$, and replacing each such element $x$ by zero or any finite number of elements from $A$, each strictly less than $x$ (in the ordering $\preceq$).

Formally, $M << M'$ iff $M = M'$, or there exist some finite multisets $X, Y$ with $X$ a nonempty submultiset of $M'$ such that, $M = (M' - X) \cup Y$, and for all $y \in Y$, there is some $x \in X$ such that $y \prec x$.

(a) If $A = \mathbf{N} = \{0, 1, 2, ...\}$ and $\preceq$ is the natural ordering on $\mathbf{N}$, give examples of pairs of multisets related by the relation $<<$.

(b) Prove that $<<$ is a partial ordering.

(c) Assume that $\preceq$ is well founded. To prove that $<<$ is also well founded, we shall proceed by contradiction as follows. Assume that there is an infinite decreasing sequence $M_0 >> M_1 >> ... >> M_n >> M_{n+1} >> ...$, and $M_n \neq M_{n+1}$ for all $n \geq 0$ ($M >> N$ holds iff $N << M$ holds). We build a tree in the following fashion:

Begin with a root node whose immediate descendants are labeled with the elements of $M_0$. Since $M_0 >> M_1$ (and $M_0 \neq M_1$), there exist multisets $X$ and $Y$ with $X$ a nonempty multiset of $M_0$, such that $M_1 = (M_0 - X) \cup Y$, and for every $y \in Y$, there is some $x \in X$ and $y \prec x$. For each $y$ in $Y$, choose some $x$ in $X$ such that $y \prec x$, and add a successor node labeled $y$ to the node corresponding to that $x$. For every remaining $x$ in $X$ (element that is dropped and replaced by no elements at all), add a successor labeled with the special symbol $\perp$. This last step guarantees that at least one new node is added to the tree for every multiset $M_n$ in the sequence. This is necessary in case $Y$ is empty. Repeat the process for $M_1 >> M_2, M_2 >> M_3$, and so on. Let $T$ be the resulting tree.

Note that by construction, the elements on any path form a strictly decreasing sequence in $A$ (we can assume that $\perp$ is less than any element in $A$).

(i) Prove that the tree $T$ is infinite and that each node has a finite number of successors. Then, by König's lemma (if a tree is finite branching and infinite then it contains an infinite path), there must be an infinite path in $T$.

(ii) Prove that there is a path in $T$ corresponding to an infinite decreasing sequence of elements in $A$. Conclude that the partial ordering $<<$ is well founded.

**2.2.1.** Let $t$ be a tree and let $u$ and $v$ be two independent tree addresses in $dom(t)$ (that is, $u$ is not a prefix of $v$ and $v$ is not a prefix of $u$). Prove that for any trees $t_1$ and $t_2$,

$$t[u \leftarrow t_1][v \leftarrow t_2] = t[v \leftarrow t_2][u \leftarrow t_1].$$

**2.3.1.** Let $A = \{a, b, c\}$ and $* : A \times A \to A$ be the function defined by the following table:

| $*$ | $a$ | $b$ | $c$ |
|---|---|---|---|
| $a$ | $a$ | $b$ | $c$ |
| $b$ | $b$ | $c$ | $a$ |
| $c$ | $c$ | $a$ | $b$ |

(a) Show that the inductive closure of $X = \{b\}$ is $A$.

(b) If $\mathbf{N}$ denotes the set of nonnegative integers and $+$ is addition (of integers), show that there is some function $h : X \to \mathbf{N}$ which does not have any homomorphic extension to $A$ (a function $g$ is a homomorphic extension of $h$ if, $g(b) = h(b)$ and $g(x * y) = g(x) + g(y)$, for all $x, y \in A$).

Find an infinite set that is the inductive closure of a finite set, but is not freely generated.

**2.3.2.** Show that if $X_+$ is freely generated by $X$ and $F$, then $X_i^n - X_{i-1}^n \neq (X_i - X_{i-1})^n$. Show that if $X_+$ is not freely generated, then $X_i^n - X_{i-1}^n = (X_i - X_{i-1})^n$ is possible.

$*$ **2.3.3.** Recall from Subsection 2.2.6 that $T_\Sigma$ denotes the set of all finite $\Sigma$-trees over the ranked alphabet $\Sigma$. Every function symbol $f$ of rank $n > 0$ defines the function $\overline{f} : T_\Sigma^n \to T_\Sigma$ as follows: For every $t_1, t_2, ..., t_n \in T_\Sigma$, $\overline{f}(t_1, t_2, ..., t_n)$ is the tree denoted by $f t_1 t_2 ... t_n$ and whose graph is the set of pairs

$$\{(e, f)\} \cup \bigcup_{i=1}^{i=n} \{(iu, t_i(u)) \mid u \in dom(t_i)\}.$$

The tree $f t_1 ... t_n$ is the tree with $f$ at the root and $t_i$ as the subtree at address $i$.

(a) Prove that $T_\Sigma$ is freely generated by the set $\Sigma_0$ of constant symbols in $\Sigma$ and the functions $\overline{f}$ defined above.

*Hint*: See the proof of lemma 2.4.2 in the Appendix.

Let $A$ be a set, $X$ a subset of $A$, $F$ a set of functions on $A$, and $X_+$ the inductive closure of $X$ under $F$. We define the ranked alphabet $\Sigma$ as follows:

$$\Sigma_0 = X,$$
$$\Sigma_n = \{f \mid f \in F \text{ of rank } n\}.$$

(b) Prove that the unique homomorphic extension $h : T_\Sigma \to X_+$ of the inclusion function $J : X \to X_+$ is surjective. We say that a tree $t \in T_\Sigma$ *represents* an element $x \in X_+$ iff $h(t) = x$.

(c) Prove that $X_+$ is freely generated by $X$ and $F$ iff $h$ is a bijection.

$*$ **2.3.4.** Prove that $EXPR$ is freely generated by $V \cup \{0, 1\}$ and $\{H_+, H_*\}$.

*Hint*: Use the proof technique of theorem 3.2.1.