Contents

1 Mathematical Preliminaries

1.1 Notational Conventions

The integers, the rational numbers, and the real numbers are denoted $\mathbb{Z}$, $\mathbb{Q}$, and $\mathbb{R}$, respectively. The natural numbers are denoted $\mathbb{N}$ or ω; we usually use the former when thinking of them as an algebraic structure with arithmetic operations $+$ and $\cdot$ and the latter when thinking of them as the set of finite ordinal numbers. We reserve the symbols i, j, k, m, n to denote natural numbers.

We use the symbols $\Longrightarrow$, $\rightarrow$ for implication (if-then) and $\Longleftrightarrow$, $\leftrightarrow$ for bidirectional implication (equivalence, if and only if). The single-line versions will denote logical symbols in the systems under study, whereas the double-line versions are metasymbols standing for the English "implies" and "if and only if" in verbal proofs. Some authors use $\supset$ and $\equiv$, but we will reserve these symbols for other purposes. The symbol $\rightarrow$ is also used in the specification of the type of a function, as in $f : A \rightarrow B$, indicating that f has domain A and range B.

The word "iff" is an abbreviation for "if and only if."

We use the notation $\stackrel{\text{def}}{=}$ and $\stackrel{\text{def}}{\Longleftrightarrow}$ to indicate that the object on the left is being defined in terms of the object on the right. When a term is being defined, it is written in *italics*.

We adopt the convention that for any associative binary operation $\cdot$ with left and right identity ι, the empty product is ι. For example, for the operation of addition in $\mathbb{R}$, we take the empty sum $\sum_{a \in \varnothing} a$ to be 0, and we take the empty union $\bigcup_{A \in \varnothing} A$ to be $\varnothing$.

The length of a finite sequence σ of objects is denoted $|\sigma|$. The set of all finite sequences of elements of A is denoted A^*. Elements of A^* are also called *strings*. The unique element of A^* of length 0 is called the *empty string* or *empty sequence* and is denoted ε. The set A^* is called the *asterate* of A, and the set operator * is called the *asterate operator*.

The *reverse* of a string w, denoted w^R, is w written backwards.

1.2 Sets

Sets are denoted $A, B, C, \ldots$, possibly with subscripts. The symbol $\in$ denotes set containment: $x \in A$ means x is an element of A. The symbol $\subseteq$ denotes set inclusion: $A \subseteq B$ means A is a subset of B. We write $x \notin B$ and $A \not\subseteq B$ to indicate that x is not an element of B and A is not a subset of B, respectively.

Strict inclusion is denoted $\subset$. The cardinality of a set A is denoted $\#A$.

The *powerset* of a set A is the set of all subsets of A and is denoted 2^A. The empty set is denoted $\varnothing$. The union and intersection of sets A and B are denoted $A \cup B$ and $A \cap B$, respectively. If $\mathcal{A}$ is a set of sets, then $\bigcup \mathcal{A}$ and $\bigcap \mathcal{A}$ denote the union and the intersection, respectively, of all sets in $\mathcal{A}$ (for the latter, we require $\mathcal{A} \neq \varnothing$). That is,

$$\bigcup \mathcal{A} \stackrel{\text{def}}{=} \{x \mid \exists B \in \mathcal{A}\ x \in B\}$$

$$\bigcap \mathcal{A} \stackrel{\text{def}}{=} \{x \mid \forall B \in \mathcal{A}\ x \in B\}.$$

The *complement* of A in B is the set of all elements of B that are not in A and is denoted $B - A$. If B is understood, then we sometimes write $\sim A$ for $B - A$.

We use the standard set-theoretic notation $\{x \mid \varphi(x)\}$ and $\{x \in A \mid \varphi(x)\}$ for the class of all x satisfying the property φ and the set of all $x \in A$ satisfying φ, respectively.

The *Cartesian product* of sets A and B is the set of ordered pairs

$$A \times B \stackrel{\text{def}}{=} \{(a, b) \mid a \in A \text{ and } b \in B\}.$$

More generally, if A_α is an indexed family of sets, $\alpha \in I$, then the *Cartesian product* of the sets A_α is the set $\prod_{\alpha \in I} A_\alpha$ consisting of all I-tuples whose α^{th} component is in A_α for all $\alpha \in I$.

In particular, if all $A_\alpha = A$, we write A^I for $\prod_{\alpha \in I} A_\alpha$; if in addition I is the finite set $\{0, 1, \ldots, n-1\}$, we write

$$\begin{aligned} A^n &\stackrel{\text{def}}{=} \prod_{i=0}^{n-1} A_i \\ &= \underbrace{A \times \cdots \times A}_{n} \\ &= \{(a_0, \ldots, a_{n-1}) \mid a_i \in A,\ 0 \leq i \leq n-1\}. \end{aligned}$$

The set A^n is called the n^{th} *Cartesian power* of A.

Along with the Cartesian product $\prod_{\alpha \in I} A_\alpha$ come the *projection functions* $\pi_\beta : \prod_{\alpha \in I} A_\alpha \to A_\beta$. The function π_β applied to $x \in \prod_{\alpha \in I} A_\alpha$ gives the β^{th} component of x. For example, the projection function $\pi_0 : \mathbb{N}^3 \to \mathbb{N}$ gives $\pi_0(3, 5, 7) = 3$.

A Note on Foundations

We take Zermelo–Fraenkel set theory with the axiom of choice (ZFC) as our foundational system. In pure ZFC, everything is built out of sets, and sets are

the only objects that can be elements of other sets. We will not give a systematic introduction to ZFC, but just point out a few relevant features.

The axioms of ZFC allow the formation of unions, ordered pairs, powersets, and Cartesian products. Infinite sets are allowed. Representations of all common datatypes and operations on them, such as strings, natural numbers, real numbers, trees, graphs, lists, and so forth, can be defined from these basic set operations. One important feature is the construction of the ordinal numbers and the transfinite induction principle, which we discuss in more detail in Section 1.6 below.

Sets and Classes

In ZFC, there is a distinction between *sets* and *classes*. For any property φ of sets, one can form the class

$$\{X \mid \varphi(X)\} \tag{1.2.1}$$

of all sets satisfying φ, along with the corresponding deduction rule

$$A \in \{X \mid \varphi(X)\} \iff \varphi(A) \tag{1.2.2}$$

for any set A. Any set B is a class $\{X \mid X \in B\}$, but classes need not be sets. Early versions of set theory assumed that any class of the form (1.2.1) was a set; this assumption is called *comprehension*. Unfortunately, this assumption led to inconsistencies such as *Russell's paradox* involving the class of all sets that do not contain themselves:

$$\{X \mid X \notin X\}. \tag{1.2.3}$$

If this were a set, say B, then by (1.2.2), $B \in B$ if and only if $B \notin B$, a contradiction.

Russell's paradox and other similar paradoxes were resolved by weakening Cantor's original version of set theory. Comprehension was replaced by a weaker axiom that states that if A is a set, then so is $\{X \in A \mid \varphi(X)\}$, the class of all elements of A satisfying φ. Classes that are not sets, such as (1.2.3), are called *proper classes*.

1.3 Relations

A *relation* is a subset of a Cartesian product. For example, the relation "is a daughter of" is a subset of the product {female humans} × {humans}. Relations are denoted $P, Q, R, \ldots$, possibly with subscripts. If A is a set, then a *relation on* A is a subset R of A^n for some n. The number n is called the *arity* of R. The relation

R on A is called *nullary*, *unary* (or *monadic*), *binary* (or *dyadic*), *ternary*, or *n-ary* if its arity is $0, 1, 2, 3$, or n, respectively. A unary relation on A is just a subset of A. The *empty relation* $\varnothing$ is the relation containing no tuples. It can be considered as a relation of any desired arity; all other relations have a unique arity.

If R is an n-ary relation, we sometimes write $R(a_1, \ldots, a_n)$ to indicate that the tuple $(a_1, \ldots, a_n)$ is in R. For binary relations, we may write $a\ R\ b$ instead of $R(a, b)$, as dictated by custom; this is particularly common for binary relations such as $\leq$, $\subseteq$, and $=$.

Henceforth, we will assume that the use of any expression of the form $R(a_1, \ldots, a_n)$ carries with it the implicit assumption that R is of arity n. We do this to avoid having to write "... where R is n-ary."

A relation R is said to *refine* or *be a refinement of* another relation S if $R \subseteq S$, considered as sets of tuples.

Binary Relations

A binary relation R on U is said to be

- *reflexive* if $(a, a) \in R$ for all $a \in U$;
- *irreflexive* if $(a, a) \notin R$ for all $a \in U$;
- *symmetric* if $(a, b) \in R$ whenever $(b, a) \in R$;
- *antisymmetric* if $a = b$ whenever both $(a, b) \in R$ and $(b, a) \in R$;
- *transitive* if $(a, c) \in R$ whenever both $(a, b) \in R$ and $(b, c) \in R$;
- *well-founded* if every nonempty subset $X \subseteq U$ has an R-minimal element; that is, an element $b \in X$ such that for no $a \in X$ is it the case that $a\ R\ b$.

A binary relation R on U is called

- a *preorder* or *quasiorder* if it is reflexive and transitive;
- a *partial order* if it is reflexive, antisymmetric, and transitive;
- a *strict partial order* if it is irreflexive and transitive;
- a *total order* or *linear order* if it is a partial order and for all $a, b \in U$ either $a\ R\ b$ or $b\ R\ a$;
- a *well order* if it is a well-founded total order; equivalently, if it is a partial order and every subset of U has a *unique* R-least element;
- an *equivalence relation* if it is reflexive, symmetric, and transitive.

A partial order is *dense* if there is an element strictly between any two distinct comparable elements; that is, if $a\ R\ c$ and $a \neq c$, then there exists an element b

such that $b \neq a$, $b \neq c$, $a\ R\ b$, and $b\ R\ c$.

The *identity relation* ι on a set U is the binary relation

$$\iota \stackrel{\text{def}}{=} \{(s,s) \mid s \in U\}.$$

Note that a relation is reflexive iff ι refines it.

The *universal relation* on U of arity n is U^n, the set of all n-tuples of elements of U.

An important operation on binary relations is *relational composition* $\circ$. If P and Q are binary relations on U, their *composition* is the binary relation

$$P \circ Q \stackrel{\text{def}}{=} \{(u,w) \mid \exists v \in U\ (u,v) \in P \text{ and } (v,w) \in Q\}.$$

The identity relation ι is a left and right identity for the operation $\circ$; in other words, for any R, $\iota \circ R = R \circ \iota = R$. A binary relation R is transitive iff $R \circ R \subseteq R$.

One can generalize $\circ$ to the case where P is an m-ary relation on U and Q is an n-ary relation on U. In this case we define $P \circ Q$ to be the $(m+n-2)$-ary relation

$$P \circ Q \stackrel{\text{def}}{=} \{(\overline{u},\overline{w}) \mid \exists v \in U\ (\overline{u},v) \in P,\ (v,\overline{w}) \in Q\}.$$

In Dynamic Logic, we will find this extended notion of relational composition useful primarily in the case where the left argument is binary and the right argument is unary. In this case,

$$P \circ Q = \{s \mid \exists t \in U\ (s,t) \in P \text{ and } t \in Q\}.$$

We abbreviate the n-fold composition of a binary relation R by R^n. Formally,

$$\begin{aligned} R^0 &\stackrel{\text{def}}{=} \iota, \\ R^{n+1} &\stackrel{\text{def}}{=} R \circ R^n. \end{aligned}$$

This notation is the same as the notation for Cartesian powers described in Section 1.2, but both notations are standard, so we will rely on context to distinguish them. One can show by induction that for all $m, n \geq 0$, $R^{m+n} = R^m \circ R^n$.

The *converse* operation $^-$ on a binary relation R reverses its direction:

$$R^- \stackrel{\text{def}}{=} \{(t,s) \mid (s,t) \in R\}.$$

Note that $R^{--} = R$. A binary relation R is symmetric iff $R^- \subseteq R$; equivalently, if $R^- = R$.

Two important operations on binary relations are

$$R^* \stackrel{\text{def}}{=} \bigcup_{n \geq 0} R^n$$
$$R^+ \stackrel{\text{def}}{=} \bigcup_{n \geq 1} R^n.$$

The relations R^+ and R^* are called the *transitive closure* and the *reflexive transitive closure* of R, respectively. The reason for this terminology is that R^+ is the smallest (in the sense of set inclusion $\subseteq$) transitive relation containing R, and R^* is the smallest reflexive and transitive relation containing R (Exercise 1.13).

We will develop some basic properties of these constructs in the exercises. Here is a useful lemma that can be used to simplify several of the arguments involving the * operation.

LEMMA 1.1: Relational composition distributes over arbitrary unions. That is, for any binary relation P and any indexed family of binary relations Q_α,

$$P \circ (\bigcup_\alpha Q_\alpha) = \bigcup_\alpha (P \circ Q_\alpha),$$
$$(\bigcup_\alpha Q_\alpha) \circ P = \bigcup_\alpha (Q_\alpha \circ P).$$

Proof For the first equation,

$$\begin{aligned}
(u,v) \in P \circ (\bigcup_\alpha Q_\alpha) &\iff \exists w\ (u,w) \in P \text{ and } (w,v) \in \bigcup_\alpha Q_\alpha \\
&\iff \exists w\ \exists \alpha\ (u,w) \in P \text{ and } (w,v) \in Q_\alpha \\
&\iff \exists \alpha\ \exists w\ (u,w) \in P \text{ and } (w,v) \in Q_\alpha \\
&\iff \exists \alpha\ (u,v) \in P \circ Q_\alpha \\
&\iff (u,v) \in \bigcup_\alpha (P \circ Q_\alpha).
\end{aligned}$$

The proof of the second equation is similar. ■

Equivalence Relations

Recall from Section 1.3 that a binary relation on a set U is an *equivalence relation* if it is reflexive, symmetric, and transitive. Given an equivalence relation $\equiv$ on U,

the $\equiv$-*equivalence class* of $a \in U$ is the set

$$\{b \in U \mid b \equiv a\},$$

typically denoted $[a]$. By reflexivity, $a \in [a]$; and if $a, b \in U$, then

$$a \equiv b \iff [a] = [b]$$

(Exercise 1.10).

A *partition* of a set U is a collection of pairwise disjoint subsets of U whose union is U. That is, it is an indexed collection $A_\alpha \subseteq U$ such that $\bigcup_\alpha A_\alpha = U$ and $A_\alpha \cap A_\beta = \varnothing$ for all $\alpha \neq \beta$.

There is a natural one-to-one correspondence between equivalence relations and partitions. The equivalence classes of an equivalence relation on U form a partition of U; conversely, any partition of U gives rise to an equivalence relation by declaring two elements to be equivalent if they are in the same set of the partition.

Recall that a binary relation $\equiv_1$ on U *refines* another binary relation $\equiv_2$ on U if for any $a, b \in U$, if $a \equiv_1 b$ then $a \equiv_2 b$. For equivalence relations, this is the same as saying that every equivalence class of $\equiv_1$ is included in an equivalence class of $\equiv_2$; equivalently, every $\equiv_2$-class is a union of $\equiv_1$-classes. Any family $\mathcal{E}$ of equivalence relations has a *coarsest common refinement*, which is the $\subseteq$-greatest relation refining all the relations in $\mathcal{E}$. This is just $\bigcap \mathcal{E}$, thinking of elements of $\mathcal{E}$ as sets of ordered pairs. Each equivalence class of the coarsest common refinement is an intersection of equivalence classes of relations in $\mathcal{E}$.

Functions

Functions are denoted $f, g, h, \ldots$, possibly with subscripts. Functions, like relations, are formally sets of ordered pairs. More precisely, a function f is a binary relation such that no two distinct elements have the same first component; that is, for all a there is at most one b such that $(a, b) \in f$. In this case we write $f(a) = b$. The a is called the *argument* and the b is called the *value*.

The *domain* of f is the set $\{a \mid \exists b\ (a, b) \in f\}$ and is denoted $\operatorname{dom} f$. The function f is said to be *defined* on a if $a \in \operatorname{dom} f$. The *range* of f is any set containing $\{b \mid \exists a\ (a, b) \in f\} = \{f(a) \mid a \in \operatorname{dom} f\}$. We use the phrase "the range" loosely; the range is not unique. We write $f : A \to B$ to denote that f is a function with domain A and range B. The set of all functions $f : A \to B$ is denoted $A \to B$ or B^A.

A function can be specified anonymously with the symbol $\mapsto$. For example, the function $x \mapsto 2x$ on the integers is the function $\mathbb{Z} \to \mathbb{Z}$ that doubles its argument.

Formally, it is the set of ordered pairs $\{(x, 2x) \mid x \in \mathbb{Z}\}$. The symbol $\restriction$ is used to restrict a function to a smaller domain. For example, if $f : \mathbb{R} \to \mathbb{R}$, then $f \restriction \mathbb{Z} : \mathbb{Z} \to \mathbb{R}$ is the function that agrees with f on the integers but is otherwise undefined.

If $C \subseteq A$ and $f : A \to B$, then $f(C)$ denotes the set

$$f(C) \stackrel{\text{def}}{=} \{f(a) \mid a \in C\} \subseteq B.$$

This is called the *image* of C under f. The *image* of f is the set $f(A)$.

The *composition* of two functions f and g is just the relational composition $f \circ g$ as defined in Section 1.3. If $f : A \to B$ and $g : B \to C$, then $f \circ g : A \to C$, and

$$(f \circ g)(a) = g(f(a)).^1$$

A function $f : A \to B$ is *one-to-one* or *injective* if $f(a) \neq f(b)$ whenever $a, b \in A$ and $a \neq b$. A function $f : A \to B$ is *onto* or *surjective* if for all $b \in B$ there exists $a \in A$ such that $f(a) = b$. A function is *bijective* if it is both injective and surjective. A bijective function $f : A \to B$ has an *inverse* $f^{-1} : B \to A$ in the sense that $f \circ f^{-1}$ is the identity function on A and $f^{-1} \circ f$ is the identity function on B. Thinking of f as a binary relation, f^{-1} is just the converse f^- as defined in Section 1.3. Two sets are said to be in *one-to-one correspondence* if there exists a bijection between them.

We will find the following *function-patching* operator useful. If $f : A \to B$ is any function, $a \in A$, and $b \in B$, then we denote by $f[a/b] : A \to B$ the function defined by

$$f[a/b](x) \stackrel{\text{def}}{=} \begin{cases} b, & \text{if } x = a \\ f(x), & \text{otherwise.} \end{cases}$$

In other words, $f[a/b]$ is the function that agrees with f everywhere except possibly a, on which it takes the value b.

Partial Orders

Recall from Section 1.3 that a binary relation $\leq$ on a set A is a *preorder* (or *quasiorder*) if it is reflexive and transitive, and it is a *partial order* if in addition it is antisymmetric. Any preorder $\leq$ has a natural associated equivalence relation

$$a \equiv b \iff a \leq b \text{ and } b \leq a.$$

1 Unfortunately, the definition $(f \circ g)(a) = f(g(a))$ is fairly standard, but this conflicts with other standard usage, namely the definition of $\circ$ for binary relations and the definition of functions as sets of (argument,value) pairs.

The order $\leq$ is well-defined on $\equiv$-equivalence classes; that is, if $a \leq b$, $a \equiv a'$, and $b \equiv b'$, then $a' \leq b'$. It therefore makes sense to define $[a] \leq [b]$ if $a \leq b$. The resulting order on $\equiv$-classes is a partial order.

A *strict partial order* is a binary relation $<$ that is irreflexive and transitive. Any strict partial order $<$ has an associated partial order $\leq$ defined by $a \leq b$ if $a < b$ or $a = b$. Any preorder $\leq$ has an associated strict partial order defined by $a < b$ if $a \leq b$ but $b \not\leq a$. For partial orders $\leq$, these two operations are inverses. For example, the strict partial order associated with the set inclusion relation $\subseteq$ is the proper inclusion relation $\subset$.

A function $f : (A, \leq) \to (A', \leq)$ between two partially ordered sets is *monotone* if it preserves order: if $a \leq b$ then $f(a) \leq f(b)$.

Let $\leq$ be a partial order on A and let $B \subseteq A$. An element $x \in A$ is an *upper bound* of B if $y \leq x$ for all $y \in B$. The element x itself need not be an element of B. If in addition $x \leq z$ for every upper bound z of B, then x is called the *least upper bound*, *supremum*, or *join* of B, and is denoted $\sup_{y \in B} y$ or $\sup B$. The supremum of a set need not exist, but if it does, then it is unique.

A function $f : (A, \leq) \to (A', \leq)$ between two partially ordered sets is *continuous* if it preserves all existing suprema: whenever $B \subseteq A$ and $\sup B$ exists, then $\sup_{x \in B} f(x)$ exists and is equal to $f(\sup B)$.

Any partial order extends to a total order. In fact, any partial order is the intersection of all its total extensions (Exercise 1.14).

Recall that a preorder $\leq$ is *well-founded* if every subset has a $\leq$-minimal element. An *antichain* is a set of pairwise $\leq$-incomparable elements.

Proposition 1.2: Let $\leq$ be a preorder on a set A. The following four conditions are equivalent:

(i) The relation $\leq$ is well-founded and has no infinite antichains.

(ii) For any infinite sequence $x_0, x_1, x_2, \ldots$, there exist i, j such that $i < j$ and $x_i \leq x_j$.

(iii) Any infinite sequence has an infinite nondecreasing subsequence; that is, for any sequence $x_0, x_1, x_2, \ldots$, there exist $i_0 < i_1 < i_2 < \cdots$ such that $x_{i_0} \leq x_{i_1} \leq x_{i_2} \leq \cdots$.

(iv) Any set $X \subseteq A$ has a finite base $X_0 \subseteq X$; that is, a finite subset X_0 such that for all $y \in X$ there exists $x \in X_0$ such that $x \leq y$.

In addition, if $\leq$ is a partial order, then the four conditions above are equivalent to the fifth condition:

(v) Any total extension of $\leq$ is a well order.

Proof Exercise 1.16. ■

A preorder or partial order is called a *well quasiorder* or *well partial order*, respectively, if it satisfies any of the four equivalent conditions of Proposition 1.2.

Well-Foundedness and Induction

Everyone is familiar with the set $\omega = \{0, 1, 2, \ldots\}$ of *finite ordinals*, also known as the *natural numbers*. An essential mathematical tool is the *induction principle* on this set, which states that if a property is true of zero and is preserved by the successor operation, then it is true of all elements of ω.

There are more general notions of induction that we will find useful. *Transfinite induction* extends induction on ω to higher ordinals. We will discuss transfinite induction later on in Section 1.6. *Structural induction* is used to establish properties of inductively defined objects such as lists, trees, or logical formulas.

All of these types of induction are instances of a more general notion called *induction on a well-founded relation* or just *well-founded induction*. This is in a sense the most general form of induction there is (Exercise 1.15). Recall that a binary relation R on a set X is *well-founded* if every subset of X has an R-minimal element. For such relations, the following induction principle holds:

(WFI) If φ is a property that holds for x whenever it holds for all R-predecessors of x—that is, if φ is true of x whenever it is true of all y such that $y \mathrel{R} x$—then φ is true of all x.

The "basis" of the induction is the case of R-minimal elements; that is, those with no R-predecessors. In this case, the premise "φ is true of all R-predecessors of x" is vacuously true.

Lemma 1.3 (Principle of well-founded induction): If R is a well-founded relation on a set X, then the well-founded induction principle (WFI) is sound.

Proof Suppose φ is a property such that, whenever φ is true of all y such that $y \mathrel{R} x$, then φ is also true of x. Let S be the set of all elements of X satisfying property φ. Then $X - S$ is the set of all elements of X not satisfying property φ. Since R is well-founded, if $X - S$ were nonempty, it would have a minimal element x. But then φ would be true of all R-predecessors of x but not of x itself, contradicting our assumption. Therefore $X - S$ must be empty, and consequently $S = X$. ■

In fact, the converse holds as well: if R is not well-founded, then there is a property on elements of X that violates the principle (WFI) (Exercise 1.15).

1.4 Graphs and Dags

For the purposes of this book, we define a *directed graph* to be a pair $\mathcal{D} = (D, \to_{\mathcal{D}})$, where D is a set and $\to_{\mathcal{D}}$ is a binary relation on D. Elements of D are called *vertices* and elements of $\to_{\mathcal{D}}$ are called *edges*. In graph theory, one often sees more general types of graphs allowing multiple edges or weighted edges, but we will not need them here.

A directed graph is called *acyclic* if for no $d \in D$ is it the case that $d \to_{\mathcal{D}}^{+} d$, where $\to_{\mathcal{D}}^{+}$ is the transitive closure of $\to_{\mathcal{D}}$. A directed acyclic graph is called a *dag*.

1.5 Lattices

An *upper semilattice* is a partially ordered set in which every finite subset has a supremum (join). Every semilattice has a unique smallest element, namely the join of the empty set.

A *lattice* is a partially ordered set in which every finite subset has both a supremum and an *infimum*, or *greatest lower bound*. The infimum of a set is often called its *meet*. Every lattice has a unique largest element, which is the meet of the empty set.

A *complete lattice* is a lattice in which all joins exist; equivalently, in which all meets exist. Each of these assumptions implies the other: the join of a set B is the meet of the set of upper bounds of B, and the meet of B is the join of the set of lower bounds of B. Every set B has at least one upper bound and one lower bound, namely the top and bottom elements of the lattice, respectively.

For example, the powerset 2^A of a set A is a complete lattice under set inclusion $\subseteq$. In this lattice, for any $B \subseteq 2^A$, the supremum of B is $\bigcup B$ and its infimum is $\bigcap B$.

1.6 Transfinite Ordinals

The *induction principle* on the natural numbers $\omega = \{0, 1, 2, \ldots\}$ states that if a property is true of zero and is preserved by the successor operation, then it is true of all elements of ω.

In the study of logics of programs, we often run into higher ordinals, and it is useful to have a *transfinite induction principle* that applies in those situations. Cantor recognized the value of such a principle in his theory of infinite sets. Any modern account of the foundations of mathematics will include a chapter on ordinals and transfinite induction.

Unfortunately, a complete understanding of ordinals and transfinite induction is impossible outside the context of set theory, since many issues impact the very foundations of the subject. A thorough treatment would fill a large part of a basic course in set theory, and is well beyond the scope of this introduction. We provide here only a cursory account of the basic facts, tools and techniques we will need in the sequel. We encourage the student interested in foundations to consult the references at the end of the chapter for a more detailed treatment.

Set-Theoretic Definition of Ordinals

Ordinals are defined as certain sets of sets. The key facts we will need about ordinals, succinctly stated, are:

(i) There are two kinds: *successors* and *limits*.

(ii) They are well-ordered.

(iii) There are a lot of them.

(iv) We can do induction on them.

We will explain each of these statements in more detail below.

A set C of sets is said to be *transitive* if $C \subseteq 2^C$; that is, every element of C is a subset of C. In other words, if $A \in B$ and $B \in C$, then $A \in C$. Formally, an *ordinal* is defined to be a set A such that

- A is transitive
- all elements of A are transitive.

It follows that any element of an ordinal is an ordinal. We use $\alpha, \beta, \gamma, \ldots$ to refer to ordinals. The class of all ordinals is denoted **Ord**. It is not a set, but a proper class.

This rather neat but perhaps obscure definition of ordinals has some far-reaching consequences that are not at all obvious. For ordinals α, β, define $\alpha < \beta$ if $\alpha \in \beta$. Then every ordinal is equal to the set of all smaller ordinals (in the sense of $<$). The relation $<$ is a strict partial order in the sense of Section 1.3.

If α is an ordinal, then so is $\alpha \cup \{\alpha\}$. The latter is called the *successor* of α and is denoted $\alpha + 1$. Also, if A is any set of ordinals, then $\bigcup A$ is an ordinal and is the

supremum of the ordinals in A under the relation $\leq$.

The smallest few ordinals are

$$\begin{aligned}
0 &\stackrel{\text{def}}{=} \varnothing \\
1 &\stackrel{\text{def}}{=} \{0\} = \{\varnothing\} \\
2 &\stackrel{\text{def}}{=} \{0,1\} = \{\varnothing,\{\varnothing\}\} \\
3 &\stackrel{\text{def}}{=} \{0,1,2\} = \{\varnothing,\{\varnothing\},\{\varnothing,\{\varnothing\}\}\} \\
&\vdots
\end{aligned}$$

The first infinite ordinal is

$$\omega \stackrel{\text{def}}{=} \{0,1,2,3,\ldots\}.$$

An ordinal is called a *successor ordinal* if it is of the form $\alpha+1$ for some ordinal α, otherwise it is called a *limit ordinal*. The smallest limit ordinal is 0 and the next smallest is ω. Of course, $\omega + 1 = \omega \cup \{\omega\}$ is an ordinal, so it does not stop there.

It follows from the axioms of ZFC that the relation $<$ on ordinals is a linear order. That is, if α and β are any two ordinals, then either $\alpha < \beta$, $\alpha = \beta$, or $\beta < \alpha$. This is most easily proved by induction on the well-founded relation

$$(\alpha,\beta) \leq (\alpha',\beta') \iff \alpha \leq \alpha' \text{ and } \beta \leq \beta'.$$

The class of ordinals is well-founded in the sense that any nonempty set of ordinals has a least element.

Since the ordinals form a proper class, there can be no one-to-one function $\mathbf{Ord} \to A$ into a set A. This is what is meant in clause (iii) above. In practice, this fact will come up when we construct functions $f : \mathbf{Ord} \to A$ from **Ord** into a set A by induction. Such an f, regarded as a collection of ordered pairs, is necessarily a class and not a set. We will always be able to conclude that there exist distinct ordinals α, β with $f(\alpha) = f(\beta)$.

Transfinite Induction

The *transfinite induction principle* is a method of establishing that a particular property is true of all ordinals. It states that in order to prove that the property is true of all ordinals, it suffices to show that the property is true of an arbitrary ordinal α whenever it is true of all ordinals $\beta < \alpha$. Proofs by transfinite induction typically contain two cases, one for successor ordinals and one for limit ordinals. The basis of the induction is often a special case of the case for limit ordinals, since $0 = \varnothing$ is a limit ordinal; here the premise that the property holds of all ordinals

$\beta < \alpha$ is vacuously true.

The validity of this principle follows ultimately from the well-foundedness of the set containment relation $\in$. This is an axiom of ZFC called the *axiom of regularity*.

We will see many examples of definitions and proofs by transfinite induction in subsequent sections.

Zorn's Lemma and the Axiom of Choice

Related to the ordinals and transfinite induction are the *axiom of choice* and *Zorn's lemma*.

The *axiom of choice* is an axiom of ZFC. It states that for every set A of nonempty sets, there exists a function f with domain A that picks an element out of each set in A; that is, for every $B \in A$, $f(B) \in B$. Equivalently, any Cartesian product of nonempty sets is nonempty.

Zorn's lemma states that every set of sets closed under unions of chains contains a $\subseteq$-maximal element. Here a *chain* is a family of sets linearly ordered by the inclusion relation $\subseteq$, and to say that a set C of sets is closed under unions of chains means that if $B \subseteq C$ and B is a chain, then $\bigcup B \in C$. An element $B \in C$ is $\subseteq$-*maximal* if it is not properly included in any $B' \in C$.

The *well ordering principle*, also known as *Zermelo's theorem*, states that every set is in one-to-one correspondence with some ordinal. A set is *countably infinite* if it is in one-to-one correspondence with ω. A set is *countable* if it is finite or countably infinite.

The axiom of choice, Zorn's lemma, and the well ordering principle are equivalent to one another and *independent* of ZF set theory (ZFC without the axiom of choice) in the sense that if ZF is consistent, then neither they nor their negations can be proven from the axioms of ZF.

In subsequent sections, we will use the axiom of choice, Zorn's lemma, and the transfinite induction principle freely.

1.7 Set Operators

A *set operator* is a function that maps sets to sets. Set operators arise everywhere in mathematics, and we will see many applications in subsequent sections. Here we introduce some special properties of set operators such as monotonicity and closure and discuss some of their consequences. We culminate with a general theorem due to Knaster and Tarski concerning inductive definitions.

Let U be a fixed set. Recall that 2^U denotes the *powerset* of U, or the set of

subsets of U:

$$2^U \stackrel{\text{def}}{=} \{A \mid A \subseteq U\}.$$

A *set operator* on U is a function $\tau : 2^U \to 2^U$.

Monotone, Continuous, and Finitary Operators

A set operator τ is said to be *monotone* if it preserves set inclusion:

$$A \subseteq B \implies \tau(A) \subseteq \tau(B).$$

A *chain of sets* in U is a family of subsets of U totally ordered by the inclusion relation $\subseteq$; that is, for every A and B in the chain, either $A \subseteq B$ or $B \subseteq A$. A set operator τ is said to be *(chain-)continuous* if for every chain of sets $\mathcal{C}$,

$$\tau(\bigcup \mathcal{C}) = \bigcup_{A \in \mathcal{C}} \tau(A).$$

A set operator τ is said to be *finitary* if its action on a set A depends only on finite subsets of A in the following sense:

$$\tau(A) = \bigcup_{\substack{B \subseteq A \\ B \text{ finite}}} \tau(B).$$

A set operator is finitary iff it is continuous, and every continuous operator is monotone, but not vice versa (Exercise 1.17). In many applications, the appropriate operators are finitary.

EXAMPLE 1.4: For a binary relation R on a set V, let

$$\begin{aligned} \tau(R) &= \{(a,c) \mid \exists b\ (a,b),\ (b,c) \in R\} \\ &= R \circ R. \end{aligned}$$

The function τ is a set operator on V^2; that is,

$$\tau : 2^{V^2} \to 2^{V^2}.$$

The operator τ is finitary, because $\tau(R)$ is determined by the action of τ on two-element subsets of R.

Prefixpoints and Fixpoints

A *prefixpoint* of a set operator τ is a set A such that $\tau(A) \subseteq A$. A *fixpoint* of τ is a set A such that $\tau(A) = A$. We say that a set A is *closed* under the operator τ if A is a prefixpoint of τ. Every set operator on U has at least one prefixpoint, namely U. Monotone set operators also have fixpoints, as we shall see.

EXAMPLE 1.5: By definition, a binary relation R on a set V is *transitive* if $(a, c) \in R$ whenever $(a, b) \in R$ and $(b, c) \in R$. Equivalently, R is transitive iff it is closed under the finitary set operator τ defined in Example 1.4.

LEMMA 1.6: The intersection of any set of prefixpoints of a monotone set operator τ is a prefixpoint of τ.

Proof Let $\mathcal{C}$ be any set of prefixpoints of τ. We wish to show that $\bigcap \mathcal{C}$ is a prefixpoint of τ. For any $A \in \mathcal{C}$, $\bigcap \mathcal{C} \subseteq A$, therefore

$$\begin{array}{lcll} \tau(\bigcap \mathcal{C}) & \subseteq & \tau(A) & \text{monotonicity of } \tau \\ & \subseteq & A & \text{since } A \text{ is a prefixpoint.} \end{array}$$

Since $A \in \mathcal{C}$ was arbitrary, $\tau(\bigcap \mathcal{C}) \subseteq \bigcap \mathcal{C}$. ■

It follows from Lemma 1.6 and the characterization of complete lattices of Section 1.5 that the set of prefixpoints of a monotone operator τ forms a complete lattice under the inclusion ordering $\subseteq$. In this lattice, the meet of any set of prefixpoints $\mathcal{C}$ is the set $\bigcap \mathcal{C}$, and the join of any set of prefixpoints $\mathcal{C}$ is the set

$$\bigcap \{A \subseteq U \mid \bigcup \mathcal{C} \subseteq A,\ A \text{ is a prefixpoint of } \tau\}.$$

Note that the join of $\mathcal{C}$ is *not* $\bigcup \mathcal{C}$ in general; this is not necessarily a prefixpoint (Exercise 1.23).

By Lemma 1.6, for any set A, the meet of all prefixpoints containing A is a prefixpoint of τ containing A and is necessarily the least prefixpoint of τ containing A. That is, if we define

$$\mathcal{C}(A) \stackrel{\text{def}}{=} \{B \subseteq U \mid A \subseteq B \text{ and } \tau(B) \subseteq B\} \tag{1.7.1}$$

$$\tau^\dagger(A) \stackrel{\text{def}}{=} \bigcap \mathcal{C}(A), \tag{1.7.2}$$

then $\tau^\dagger(A)$ is the least prefixpoint of τ containing A with respect to $\subseteq$. Note that the set $\mathcal{C}(A)$ is nonempty, since it contains U at least.

LEMMA 1.7: Any monotone set operator τ has a $\subseteq$-least fixpoint.

Proof We show that $\tau^\dagger(\varnothing)$ is the least fixpoint of τ. By Lemma 1.6, it is the least prefixpoint of τ. If it is a fixpoint, then it is the least one, since every fixpoint is a prefixpoint. But if it were not a fixpoint, then by monotonicity, $\tau(\tau^\dagger(\varnothing))$ would be a smaller prefixpoint, contradicting the fact that $\tau^\dagger(\varnothing)$ is the least. ■

Closure Operators

A set operator σ on U is called a *closure operator* if it satisfies the following three properties:

(i) σ is monotone
(ii) $A \subseteq \sigma(A)$
(iii) $\sigma(\sigma(A)) = \sigma(A)$.

Because of clause (ii), fixpoints and prefixpoints coincide for closure operators. Thus a set is closed with respect to a closure operator σ iff it is a fixpoint of σ. As shown in Lemma 1.6, the set of closed sets of a closure operator forms a complete lattice.

LEMMA 1.8: For any monotone set operator τ, the operator $\tau^\dagger$ defined in (1.7.2) is a closure operator.

Proof The operator $\tau^\dagger$ is monotone, since

$$A \subseteq B \implies \mathcal{C}(B) \subseteq \mathcal{C}(A) \implies \bigcap \mathcal{C}(A) \subseteq \bigcap \mathcal{C}(B),$$

where $\mathcal{C}(A)$ is the set defined in (1.7.1).

Property (ii) of closure operators follows directly from the definition of $\tau^\dagger$. Finally, to show property (iii), since $\tau^\dagger(A)$ is a prefixpoint of τ, it suffices to show that any prefixpoint of τ is a fixpoint of $\tau^\dagger$. But

$$\begin{aligned} \tau(B) \subseteq B &\iff B \in \mathcal{C}(B) \\ &\iff B = \bigcap \mathcal{C}(B) = \tau^\dagger(B). \end{aligned}$$

■

EXAMPLE 1.9: The *transitive closure* of a binary relation R on a set V is the least transitive relation containing R; that is, it is the least relation containing R and closed under the finitary transitivity operator τ of Example 1.4. This is the

relation $\tau^\dagger(R)$. Thus the closure operator $\tau^\dagger$ maps an arbitrary binary relation R to its transitive closure.

EXAMPLE 1.10: The *reflexive transitive closure* of a binary relation R on a set V is the least reflexive and transitive relation containing R; that is, it is the least relation that contains R, is closed under transitivity, and contains the identity relation $\iota = \{(a, a) \mid a \in V\}$. Note that "contains the identity relation" just means closed under the constant-valued monotone set operator $R \mapsto \iota$. Thus the reflexive transitive closure of R is $\sigma^\dagger(R)$, where σ denotes the finitary operator $R \mapsto \tau(R) \cup \iota$.

The Knaster–Tarski Theorem

The *Knaster–Tarski theorem* is a useful theorem that describes how least fixpoints of monotone set operators can be obtained either "from above," as in the proof of Lemma 1.7, or "from below," as a limit of a chain of sets defined by transfinite induction. In general, the Knaster–Tarski Theorem holds for monotone operators on an arbitrary complete lattice, but we will find it most useful for the lattice of subsets of a set U, so we prove it only for that case.

Let U be a set and let τ be a monotone operator on U. Let $\tau^\dagger$ be the associated closure operator defined in (1.7.2). We show how to attain $\tau^\dagger(A)$ starting from A and working up. The idea is to start with A, then apply τ repeatedly, adding new elements until achieving closure. In most applications, the operator τ is continuous, in which case this takes only countably many iterations; but for monotone operators in general, it can take more.

Formally, we construct by transfinite induction a chain of sets $\tau^\alpha(A)$ indexed by ordinals α:

$$\begin{aligned} \tau^{\alpha+1}(A) &\stackrel{\text{def}}{=} A \cup \tau(\tau^\alpha(A)) \\ \tau^\lambda(A) &\stackrel{\text{def}}{=} \bigcup_{\alpha<\lambda} \tau^\alpha(A), \quad \lambda \text{ a limit ordinal} \\ \tau^*(A) &\stackrel{\text{def}}{=} \bigcup_{\alpha\in\mathbf{Ord}} \tau^\alpha(A). \end{aligned}$$

The base case is included in the case for limit ordinals:

$$\tau^0(A) = \varnothing.$$

Intuitively, $\tau^\alpha(A)$ is the set obtained by applying τ to A α times, reincluding A at successor stages.

LEMMA 1.11: If $\alpha \leq \beta$, then $\tau^\alpha(A) \subseteq \tau^\beta(A)$.

Proof We proceed by transfinite induction. For two successor ordinals $\alpha+1$ and $\beta+1$ with $\alpha+1 \leq \beta+1$,

$$\begin{aligned} \tau^{\alpha+1}(A) &= A \cup \tau(\tau^\alpha(A)) \\ &\subseteq A \cup \tau(\tau^\beta(A)) \qquad \text{induction hypothesis and monotonicity} \\ &= \tau^{\beta+1}(A). \end{aligned}$$

If $\alpha \leq \beta$ and α is a limit ordinal,

$$\begin{aligned} \tau^\alpha(A) &= \bigcup_{\gamma<\alpha} \tau^\gamma(A) \\ &\subseteq \tau^\beta(A) \quad \text{induction hypothesis.} \end{aligned}$$

Finally, if $\alpha \leq \beta$ and β is a limit ordinal, the result is immediate from the definition of $\tau^\beta(A)$. ■

Lemma 1.11 says that the $\tau^\alpha(A)$ form a chain of sets. The set $\tau^*(A)$ is the union of this chain over all ordinals α.

Now there must exist an ordinal κ such that $\tau^{\kappa+1}(A) = \tau^\kappa(A)$, because there is no one-to-one function from the class of ordinals to the powerset of U. The least such κ is called the *closure ordinal* of τ. If κ is the closure ordinal of τ, then $\tau^\beta(A) = \tau^\kappa(A)$ for all $\beta > \kappa$, therefore $\tau^*(A) = \tau^\kappa(A)$.

If τ is continuous, then its closure ordinal is at most ω, but not for monotone operators in general (Exercise 1.18).

THEOREM 1.12 (KNASTER–TARSKI): $\tau^\dagger(A) = \tau^*(A)$.

Proof First we show the forward inclusion. Let κ be the closure ordinal of τ. Since $\tau^\dagger(A)$ is the least prefixpoint of τ containing A, it suffices to show that $\tau^*(A) = \tau^\kappa(A)$ is a prefixpoint of τ. But

$$\begin{aligned} \tau(\tau^\kappa(A)) &\subseteq A \cup \tau(\tau^\kappa(A)) \\ &= \tau^{\kappa+1}(A) \\ &= \tau^\kappa(A). \end{aligned}$$

Conversely, we show by transfinite induction that for all ordinals α, $\tau^\alpha(A) \subseteq$

$\tau^\dagger(A)$, therefore $\tau^*(A) \subseteq \tau^\dagger(A)$. For successor ordinals $\alpha + 1$,

$$\begin{array}{lcll} \tau^{\alpha+1}(A) & = & A \cup \tau(\tau^\alpha(A)) & \\ & \subseteq & A \cup \tau(\tau^\dagger(A)) & \text{induction hypothesis and monotonicity} \\ & \subseteq & \tau^\dagger(A) & \text{definition of } \tau^\dagger. \end{array}$$

For limit ordinals λ, $\tau^\alpha(A) \subseteq \tau^\dagger(A)$ for all $\alpha < \lambda$ by the induction hypothesis; therefore

$$\tau^\lambda(A) \quad = \quad \bigcup_{\alpha<\lambda} \tau^\alpha(A) \quad \subseteq \quad \tau^\dagger(A).$$

■

1.8 Bibliographical Notes

Most of the result of this chapter can be found in any basic text on discrete structures such as Gries and Schneider (1994); Rosen (1995); Graham et al. (1989). A good reference for introductory axiomatic set theory is Halmos (1960).

Exercises

1.1. Prove that relational composition is associative:

$$P \circ (Q \circ R) \quad = \quad (P \circ Q) \circ R.$$

1.2. Prove that ι is an identity and $\varnothing$ an annihilator for relational composition:

$$\begin{array}{rcl} \iota \circ P & = & P \circ \iota \quad = \quad P \\ \varnothing \circ P & = & P \circ \varnothing \quad = \quad \varnothing. \end{array}$$

1.3. Prove that relational composition is monotone in both arguments with respect to the inclusion order $\subseteq$: if $P \subseteq P'$ and $Q \subseteq Q'$, then $P \circ Q \subseteq P' \circ Q'$.

1.4. Prove that relational composition is continuous in both arguments with respect to the inclusion order $\subseteq$: for any indexed families P_α and Q_β of binary relations on a set U,

$$(\bigcup_\alpha P_\alpha) \circ (\bigcup_\beta Q_\beta) \quad = \quad \bigcup_{\alpha,\beta} (P_\alpha \circ Q_\beta).$$

1.5. Prove that the converse operation $^-$ commutes with $\cup$ and with *:

$$\begin{aligned}(\bigcup_\alpha R_\alpha)^- &= \bigcup_\alpha R_\alpha^- \\ (R^*)^- &= (R^-)^*.\end{aligned}$$

1.6. Prove that the converse operation commutes with relational composition, provided the order of the composition factors is reversed:

$$(P \circ Q)^- = Q^- \circ P^-.$$

1.7. Prove the following identities for binary relations:

$$\begin{aligned}P^n \circ P^m &= P^{m+n}, \quad m, n \geq 0 \\ P \circ P^* &= P^* \circ P \\ P &= P^{--} \\ P &\subseteq P \circ P^- \circ P.\end{aligned}$$

1.8. Give an example of a set U and a nonempty binary relation R on U such that for all m, n with $m \neq n$, $R^m \cap R^n = \varnothing$. On the other hand, show that for any binary relation R, if $m \leq n$ then $(\iota \cup R)^m \subseteq (\iota \cup R)^n$, and that

$$R^* = \bigcup_n (\iota \cup R)^n.$$

1.9. Prove that P^* is the least prefixpoint of the monotone set operator

$$X \mapsto \iota \cup (P \circ X).$$

1.10. Let $\equiv$ be an equivalence relation on a set U with equivalence classes $[a]$, $a \in U$. Show that for any $a, b \in U$,

$$a \equiv b \iff [a] = [b].$$

1.11. Let $(A, \leq)$ be a total order with associated strict order $<$. Define an order on A^* by: $a_1, \ldots, a_m \leq b_1, \ldots, b_n$ if either $a_1, \ldots, a_m$ is a prefix of $b_1, \ldots, b_n$, or there exists $i \leq m, n$ such that $a_j = b_j$ for all $j < i$ and $a_i < b_i$. Prove that this is a total order on A^n. This order is called *lexicographic order* on A^n.

1.12. Prove the following properties of binary relations R:

$$\begin{aligned} \iota &\subseteq R^* \\ R &\subseteq R^* \\ R^* \circ R^* &= R^* \\ R^{**} &= R^* \\ \iota \cup (R \circ R^*) &= R^*. \end{aligned}$$

Give purely equational proofs using Lemma 1.1 and the definition $R^* = \bigcup_n R^n$.

1.13. Prove that for any binary relation R, R^+ is the smallest (in the sense of set inclusion $\subseteq$) transitive relation containing R, and R^* is the smallest reflexive and transitive relation containing R.

1.14. Prove that any partial order is equal to the intersection of all its total extensions. That is, for any partial order R on a set X,

$$R = \bigcap \{T \subseteq X \times X \mid R \subseteq T,\ T \text{ is a total order on } X\}.$$

1.15. Let R be a binary relation on a set X. An *infinite descending R-chain* is an infinite sequence $x_0, x_1, x_2, \ldots$ of elements of X such that $x_{i+1}\ R\ x_i$ for all $i \geq 0$. Prove that the following two statements are equivalent:

(i) The relation R is well-founded.

(ii) There are no infinite descending R-chains.

1.16. Prove Proposition 1.2. *Hint.* Prove the four statements in the first part of the theorem in the order (i) $\Longrightarrow$ (ii) $\Longrightarrow$ (iii) $\Longrightarrow$ (iv) $\Longrightarrow$ (i). For (ii) $\Longrightarrow$ (iii), use *Ramsey's theorem*: if we color each element of $\{(i,j) \mid i,j \in \omega,\ i < j\}$ either red or green, then there exists an infinite set $A \subseteq \omega$ such that either all elements of $\{(i,j) \mid i,j \in A,\ i < j\}$ are red or all elements are green. In the language of graph theory, if we color the edges of the complete undirected graph on countably many vertices either red or green, then there exists either an infinite complete red subgraph or an infinite complete green subgraph. You may use Ramsey's theorem without proof.

1.17. (a) Prove that every finitary set operator is continuous and every continuous set operator is monotone. Give an example showing that the latter inclusion is

strict.

(b) Prove that every continuous set operator is finitary.

1.18. Prove that if τ is a continuous set operator, then its closure ordinal is at most ω. Give a counterexample showing that this is not true for monotone operators in general.

1.19. Show that there is a natural one-to-one correspondence between the sets of functions $A \to (B \to C)$ and $(A \times B) \to C$. *Hint.* Given $f : A \to (B \to C)$, consider the function $\mathbf{curry}(f)$ defined by

$$\mathbf{curry}(f)(x, y) \stackrel{\text{def}}{=} f(x)(y).$$

Applying the operator **curry** is often called "currying." These terms are named after Haskell B. Curry.

1.20. Prove that

$$(i, j) \mapsto \frac{(i + j + 1)(i + j)}{2} + j$$

is a one-to-one and onto function. Conclude that $\#\omega = \#(\omega^2)$.

1.21. Prove that a countable union of countable sets is countable. That is, if each A_i is countable, then $\bigcup_{i=0}^{\infty} A_i$ is countable. (*Hint.* Use Exercise 1.20).

1.22. Show that the map $\tau \mapsto \tau^*$ is a closure operator on the set $2^U \to 2^U$ curried to $2^{2^U \times U}$, and that τ^* is the least closure operator on U containing τ. (See Exercise 1.19 for a definition of *currying*.)

1.23. Show that in the lattice of prefixpoints of a monotone set operator τ constructed in Lemma 1.6, join is not necessarily union. That is, show that if $\mathcal{C}$ is a set of prefixpoints of τ, then $\bigcup \mathcal{C}$ is not necessarily a prefixpoint of τ.

1.24. (Birkhoff (1973)) Let X and Y be two partially ordered sets. A pair of functions $f : X \to Y$ and $g : Y \to X$ is called a *Galois connection* if for all $x \in X$ and $y \in Y$,

$$x \leq g(y) \iff y \leq f(x).$$

(a) Suppose f and g form a Galois connection. Prove that f and g are *anti-*

monotone in the sense that

$$\begin{aligned} x_1 \leq x_2 &\implies f(x_1) \geq f(x_2) \\ y_1 \leq y_2 &\implies g(y_1) \geq g(y_2), \end{aligned}$$

and that for all $x \in X$ and $y \in Y$,

$$\begin{array}{rclcrcl} x & \leq & g(f(x)) & \qquad & y & \leq & f(g(y)) \\ g(y) & = & g(f(g(y))) & & f(x) & = & f(g(f(x))). \end{array}$$

(b) Let U and V be sets, $R \subseteq U \times V$. Define $f : 2^U \to 2^V$ and $g : 2^V \to 2^U$ by:

$$\begin{aligned} f(A) &\stackrel{\text{def}}{=} \{y \in V \mid \text{for all } x \in A,\ x\ R\ y\} \\ g(B) &\stackrel{\text{def}}{=} \{x \in U \mid \text{for all } y \in B,\ x\ R\ y\}. \end{aligned}$$

Prove that f and g form a Galois connection between 2^U and 2^V ordered by set inclusion. Conclude from (a) that $f \circ g$ and $g \circ f$ are closure operators in the sense of Section 1.7.

2 Computability and Complexity

In this chapter we review the basic definitions and results of machine models, computability theory, and complexity theory that we will need in later chapters.

2.1 Machine Models

Deterministic Turing Machines

Our basic model of computation is the *Turing machine*, named after Alan Turing, who invented it in 1936. Turing machines can compute any function normally considered computable; in fact, we normally define *computable* to mean computable by a Turing machine.

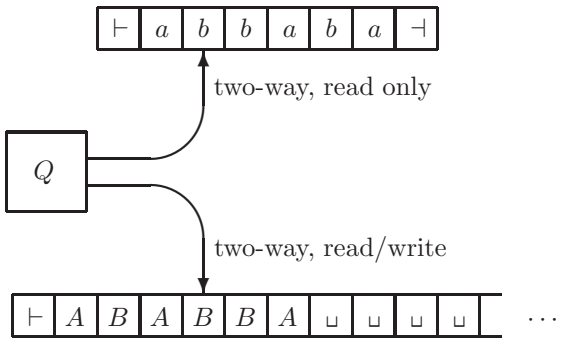
Formally, Turing machines manipulate strings over a finite alphabet. However, there is a natural one-to-one correspondence between strings in $\{0,1\}^*$ and natural numbers $\mathbb{N} = \{0,1,2,\ldots\}$ defined by

$$x \quad \mapsto \quad N(1x) - 1$$

where $N(y)$ is the natural number represented by the string y in binary. It is just as easy to encode other reasonable forms of data (strings over larger alphabets, trees, graphs, dags, etc.) as strings in $\{0,1\}^*$.

We describe below the basic model. There are many apparently more powerful variations (multitape, nondeterministic, two-way infinite tape, two-dimensional tape, ...) that can be simulated by this basic model. There are also many apparently less powerful variations (two-stack machines, two counter machines) that can simulate the basic model. All these models are equivalent in the sense that they compute all the same functions, although not with equal efficiency. One can include suitably abstracted versions of modern programming languages in this list.

Informally, a Turing machine consists of a finite set Q of *states*, an *input tape* consisting of finitely many cells delimited on the left and right by endmarkers $\vdash$ and $\dashv$, a semi-infinite *worktape* delimited on the left by an endmarker $\vdash$ and infinite to the right, and *heads* that can move left and right over the two tapes. The input string is a finite string of symbols from a finite input alphabet Σ and is written on the input tape between the endmarkers, one symbol per cell. The input head is read-only and must stay between the endmarkers. The worktape is initially blank. The worktape head can read and write symbols from a finite worktape alphabet Γ and must stay to the right of the left endmarker, but can move arbitrarily far to the right.



The machine starts in its start state s with its heads scanning the left endmarkers $\vdash$ on both tapes. The worktape is initially blank. In each step it reads the symbols on the tapes under its heads, and depending on those symbols and its current state, writes a new symbol on the worktape cell, moves its heads left or right one cell or leaves them stationary, and enters a new state. The action it takes in each situation is determined by a finite transition function δ. It *accepts* its input by entering a special accept state t and *rejects* by entering a special reject state r. On some inputs it may run infinitely without ever accepting or rejecting.

Formally, a *deterministic Turing machine* is a 10-tuple

$$M \quad = \quad (Q,\ \Sigma,\ \Gamma,\ \sqcup,\ \vdash,\ \dashv,\ \delta,\ s,\ t,\ r)$$

where:

- Q is a finite set of *states*
- Σ is a finite *input alphabet*
- Γ is a finite *worktape alphabet*
- $\sqcup \in \Gamma$ is the *blank symbol*
- $\vdash \in \Gamma - \Sigma$ is the *left endmarker*
- $\dashv \notin \Sigma$ is the *right endmarker*
- $\delta : Q \times (\Sigma \cup \{\vdash, \dashv\}) \times \Gamma \to Q \times \Gamma \times \{-1, 0, 1\}^2$ is the *transition function*
- $s \in Q$ is the *start state*
- $t \in Q$ is the *accept state*
- $r \in Q$ is the *reject state*, $r \neq t$.

The $-1, 0, 1$ in the definition of δ stand for "move left one cell," "remain stationary," and "move right one cell," respectively. Intuitively, $\delta(p, a, b) = (q, c, d, e)$ means,

"When in state p scanning symbol a on the input tape and b on the worktape, write c on that worktape cell, move the input and work heads in direction d and e, respectively, and enter state q."

We restrict Turing machines so that the left endmarker on the worktape is never overwritten with another symbol and the machine never moves its heads outside the endmarkers. We also require that once the machine enters its accept state, it remains in that state, and similarly for its reject state. This translates into certain formal constraints on the above definition.

Configurations and Acceptance

Intuitively, at any point in time, the worktape of the machine contains a semi-infinite string of the form $\vdash y\sqcup^\omega$, where $y \in \Gamma^*$ (that is, y is a finite-length string) and $\sqcup^\omega$ denotes the semi-infinite string of blanks

$$\sqcup\ \sqcup\ \sqcup\ \sqcup\ \sqcup\ \sqcup\ \sqcup\ \sqcup\ \sqcup\ \cdots$$

(recall that ω denotes the smallest infinite ordinal). Although the string $\vdash y\sqcup^\omega$ is infinite, it always has a finite representation, since all but finitely many of the symbols are the blank symbol $\sqcup$.

Let $x \in \Sigma^*$, $|x| = n$. We define a *configuration* of the machine on input x to be an element of $Q \times \{y\sqcup^\omega \mid y \in \Gamma^*\} \times \{0, 1, 2, \ldots, n+1\} \times \omega$. Intuitively, a configuration is a global state giving a snapshot of all relevant information about a Turing machine computation at some instant in time. The configuration (p, z, i, j) specifies a current state p of the finite control, current worktape contents z, and current positions i, j of the input and worktape heads, respectively. We denote configurations by $\alpha, \beta, \gamma, \ldots$. The *start configuration* on input $x \in \Sigma^*$ is the configuration

$$(s,\ \vdash\sqcup^\omega,\ 0,\ 0).$$

The last two components 0,0 mean that the machine is initially scanning the left endmarkers $\vdash$ on its two tapes.

We define a binary relation $\xrightarrow[M,x]{1}$ on configurations, called the *next configuration relation*, as follows. For a string $z \in \Gamma^\omega$, let z_j be the j^{th} symbol of z (the leftmost symbol is z_0), and let $z[j/b]$ denote the string obtained by replacing z_j by b in z. For example,

$$\vdash b\,a\,a\,a\,c\,a\,b\,c\,a\ \cdots [4/b] \quad = \quad \vdash b\,a\,a\,b\,c\,a\,b\,c\,a\ \cdots$$

Let $x_0 = \vdash$ and $x_{n+1} = \dashv$. The relation $\xrightarrow[M,x]{1}$ is defined by:

$$(p, z, i, j) \xrightarrow[M,x]{1} (q, z[j/b], i + d, j + e) \quad \stackrel{\text{def}}{\iff} \quad \delta(p, x_i, z_j) = (q, b, d, e).$$

Intuitively, if the worktape contains z and if M is in state p scanning the i^{th} cell of the input tape and the j^{th} cell of the worktape, and δ says that in that case M should print b on the worktape, move the input head in direction d (either -1, 0, or 1), move the worktape head in direction e, and enter state q, then immediately after that step the worktape will contain $z[j/b]$, the input head will be scanning the $i + d^{\text{th}}$ cell of the input tape, the worktape head will be scanning the $j + e^{\text{th}}$ cell of the worktape, and the new state will be q.

We define the relation $\xrightarrow[M,x]{*}$ to be the reflexive transitive closure of $\xrightarrow[M,x]{1}$. In other words,

- $\alpha \xrightarrow[M,x]{0} \alpha$
- $\alpha \xrightarrow[M,x]{n+1} \beta$ if $\alpha \xrightarrow[M,x]{n} \gamma \xrightarrow[M,x]{1} \beta$ for some γ
- $\alpha \xrightarrow[M,x]{*} \beta$ if $\alpha \xrightarrow[M,x]{n} \beta$ for some $n \geq 0$.

The machine M is said to *accept* input $x \in \Sigma^*$ if

$$(s, \vdash \sqcup^\omega, 0, 0) \quad \xrightarrow[M,x]{*} \quad (t, y, i, j)$$

for some y, i, and j, and to *reject* x if

$$(s, \vdash \sqcup^\omega, 0, 0) \quad \xrightarrow[M,x]{*} \quad (r, y, i, j)$$

for some y, i, and j. It is said to *halt* on input x if it either accepts x or rejects x. Note that it may do neither, but run infinitely on input x without ever accepting or rejecting. In that case it is said to *loop* on input x. A Turing machine M is said to be *total* if it halts on all inputs. The set $L(M)$ denotes the set of strings accepted by M.

A set of strings is called *recursively enumerable* (r.e.) if it is $L(M)$ for some Turing machine M, and *recursive* if it is $L(M)$ for some total Turing machine M.

Example 2.1: Here is a total Turing machine that accepts the set $\{a^n b^n c^n \mid n \geq 1\}$. Informally, the machine starts in its start state s, then scans to the right over the input string, writing an A on its worktape for every a it sees on the input

tape. When it sees the first symbol that is not an a on the input tape, it starts to move its worktape head to the left over the A's it has written, and continues to move its input head to the right, checking that the number of A's written on the worktape is equal to the number of b's occurring after the a's. It checks that it sees the left endmarker $\vdash$ on the worktape at the same time that it sees the first non-b on the input tape. It then continues to scan the input tape, meanwhile moving its worktape head to the right again, checking that the number of c's on the input tape is the same as the number of A's on the worktape. It accepts if it sees the right endmarker $\dashv$ on the input tape at the same time as it sees the first blank symbol $\sqcup$ on the worktape.

Formally, this machine has

$$\begin{aligned} Q &= \{s, q_1, q_2, q_3, t, r\} \\ \Sigma &= \{a, b, c\} \\ \Gamma &= \{A, \vdash, \sqcup\}. \end{aligned}$$

The start state, accept state, and reject state are s, t, r, respectively. The left and right endmarkers and blank symbol are $\vdash, \dashv, \sqcup$, respectively. The transition function δ is given by

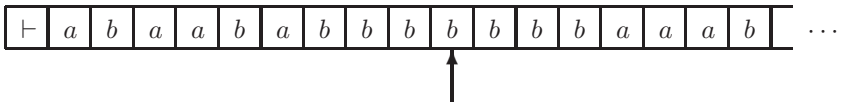
$$\begin{aligned}
\delta(s, \vdash, \vdash) &= (q_1, \vdash, 1, 1) & \delta(q_2, c, A) &= (r, -, -, -) \\
\delta(q_1, a, \sqcup) &= (q_1, A, 1, 1) & \delta(q_2, c, \vdash) &= (q_3, \vdash, 0, 1) \\
\delta(q_1, b, \sqcup) &= (q_2, \sqcup, 0, -1) & \delta(q_3, a, -) &= (r, -, -, -) \\
\delta(q_1, c, \sqcup) &= (r, -, -, -) & \delta(q_3, b, -) &= (r, -, -, -) \\
\delta(q_1, \dashv, \sqcup) &= (r, -, -, -) & \delta(q_3, c, A) &= (q_3, A, 1, 1) \\
\delta(q_2, a, -) &= (r, -, -, -) & \delta(q_3, c, \sqcup) &= (r, -, -, -) \\
\delta(q_2, b, A) &= (q_2, A, 1, -1) & \delta(q_3, \dashv, A) &= (r, -, -, -) \\
\delta(q_2, b, \vdash) &= (r, -, -, -) & \delta(q_3, \dashv, \sqcup) &= (t, -, -, -).
\end{aligned}$$

The symbol $-$ above means "don't care." Any legal value may be substituted for $-$ without affecting the behavior of the machine. Also, transitions that can never occur (for example, $\delta(q_1, a, A)$) are omitted.

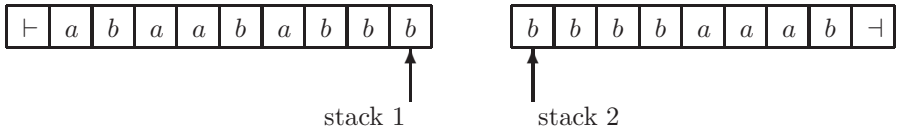
Two Stacks

A machine with a read-only input head and two stacks is as powerful as a Turing machine. Intuitively, the worktape of a Turing machine can be simulated with two stacks by storing the tape contents to the left of the head on one stack and the tape contents to the right of the head on the other stack. The motion of the head

is simulated by popping a symbol off one stack and pushing it onto the other. For example,



is simulated by



Counter Machines

A *k-counter machine* is a machine equipped with a two-way read-only input head and k integer counters, each of which can store an arbitrary nonnegative integer. In each step, the machine can test each counter for 0, and based on this information, the input symbol it is currently scanning, and its current state, it can increment or decrement its counters, move its input head one cell in either direction, and enter a new state.

A stack can be simulated with two counters as follows. We can assume without loss of generality that the stack alphabet of the stack to be simulated contains only two symbols, say 0 and 1. This is because we can encode each stack symbol as a binary number of some fixed length k, roughly the base 2 logarithm of the size of the stack alphabet; then pushing or popping one symbol is simulated by pushing or popping k binary digits. The contents of the stack can thus be regarded as a binary number whose least significant bit is on top of the stack. We maintain this binary number in the first of the two counters, and use the second to effect the stack operations. To simulate pushing a 0 onto the stack, we need to double the value in the first counter. This is done by entering a loop that repeatedly subtracts one from the first counter and adds two to the second until the first counter is 0. The value in the second counter is then twice the original value in the first counter. We can then transfer that value back to the first counter (or just switch the roles of the two counters). To push 1, the operation is the same, except that the value of the second counter is incremented once after doubling. To simulate popping, we need to divide the counter value by 2; this is done in a similar fashion.

Since a two-stack machine can simulate an arbitrary Turing machine, and since two counters can simulate a stack, it follows that a four-counter machine can simulate an arbitrary Turing machine.

However, we can do even better: a two-counter machine can simulate a four-counter machine. When the four-counter machine has the values i, j, k, ℓ in its counters, the two-counter machine will have the value $2^i 3^j 5^k 7^\ell$ in its first counter. It uses its second counter to effect the counter operations of the four-counter machine. For example, if the four-counter machine wanted to increment k (the value of the third counter), then the two-counter machine would have to multiply the value in its first counter by 5. This is done in the same way as above, adding 5 to the second counter for every 1 we subtract from the first counter. To simulate a test for zero, the two-counter machine has to determine whether the value in its first counter is divisible by 2, 3, 5, or 7, depending on which counter of the four-counter machine is being tested.

Combining these simulations, we see that two-counter machines are as powerful as arbitrary Turing machines (one-counter machines are strictly less powerful; see Exercise 2.10). However, as one can imagine, it takes an enormous number of steps of the two-counter machine to simulate one step of the Turing machine.

Nondeterministic Turing Machines

Nondeterministic Turing machines differ from deterministic Turing machines only in that the transition relation δ is not necessarily single-valued. Formally, the type of δ is now a relation

$$\delta \quad \subseteq \quad (Q \times (\Sigma \cup \{\vdash, \dashv\}) \times \Gamma) \times (Q \times \Gamma \times \{-1, 0, 1\}^2).$$

Intuitively, $((p, a, b), (q, c, d, e)) \in \delta$ means, "When in state p scanning symbols a and b on the input and worktapes, respectively, one possible move is to write c on that worktape cell, move the input and worktape heads in direction d and e, respectively, and enter state q." Since there may now be several pairs in δ with the same left-hand side (p, a, b), the next transition is not uniquely determined.

The next configuration relation $\xrightarrow[M,x]{1}$ on input x and its reflexive transitive closure $\xrightarrow[M,x]{*}$ are defined exactly as with deterministic machines. A nondeterministic Turing machine M is said to *accept* its input x if

$$(s, \vdash \sqcup^\omega, 0, 0) \quad \xrightarrow[M,x]{*} \quad (t, y, i, j)$$

for some y, i, and j; that is, if there exists a computation path from the start

configuration to an accept configuration. The main difference here is that the next configuration is not necessarily uniquely determined.

Nondeterministic algorithms are often described in terms of a "guess and verify" paradigm. This is a good way to think of nondeterministic computation informally. The machine guesses which transition to take whenever there is a choice, then checks at the end whether its sequence of guesses was correct, rejecting if not and accepting if so.

For example, to accept the set of (encodings over Σ^* of) satisfiable propositional formulas, a nondeterministic machine might guess a truth assignment to the variables of the formula, then verify its guess by evaluating the formula on that assignment, accepting if the guessed assignment satisfies the formula and rejecting if not. Each guess is a binary choice of a truth value to assign to one of the variables. This would be represented formally as a configuration with two successor configurations.

Although next configurations are not uniquely determined, the set of *possible* next configurations is. Thus there is a uniquely determined tree of possible computation sequences on any input x. The nodes of the tree are configurations, the root is the start configuration on input x, and the edges are the next configuration relation $\xrightarrow[M,x]{1}$. This tree contains an accept configuration iff the machine accepts x. The tree might also contain a reject configuration, and might also have looping computations; that is, infinite paths that neither accept nor reject. However, as long as there is at least one path that leads to acceptance, the machine is said to accept x.

Alternating Turing Machines

Another useful way to think of a nondeterministic Turing machine is as a kind of parallel machine consisting of a potentially unbounded number of processes which can spawn new subprocesses at branch points in the computation tree. Intuitively, we associate a process with each configuration in the tree. As long as the next configuration is uniquely determined, the process computes like an ordinary deterministic machine. When a branch point is encountered, say a configuration with two possible next configurations, the process spawns two subprocesses and assigns one to each next configuration. It then suspends, waiting for reports from its subprocesses. If a process enters the accept state, it reports success to its parent process and terminates. If a process enters the reject state, it reports failure to its parent and terminates. A suspended process, after receiving a report of success from at least one of its subprocesses, reports success to its parent and terminates.

If a suspended process receives a report of failure from *all* its subprocesses, it reports failure to its parent and terminates. Of course, it is possible that neither of these things happens. The machine accepts the input if the root process receives a success report. There is no explicit formal mechanism for reporting success or failure, suspending, or terminating.

Now we extend this idea to allow machines to test whether *all* subprocesses lead to success as well as whether *some* subprocess lead to success. Intuitively, each branch point in the computation tree is either an or-branch or an and-branch, depending on the state. The or-branches are handled as described above. The and-branches are just the same, except that a process suspended at an and-branch reports success to its parent iff *all* of its subprocesses report success, and reports failure to its parent iff *at least one* of its subprocesses reports failure.

Machines with this capability are called *alternating Turing machines* in reference to the alternation of and- and or-branches. They are useful in analyzing the complexity of problems with a natural alternating and/or structure, such as games or logical theories.

Formally, an alternating Turing machine is just like a nondeterministic machine, except we include an additional function $g : Q \to \{\wedge, \vee\}$ associating either $\wedge$ (and) or $\vee$ (or) with each state. A configuration (q, y, m, n) is called an *and-configuration* if $g(q) = \wedge$ and an *or-configuration* if $g(q) = \vee$. The machine is said to *accept* input x if the computation tree on input x has a finite *accepting subtree*, which is a finite subtree T containing the root such that every node c of T is either

- an accept configuration (that is, a configuration whose state is the accept state);
- an or-configuration with at least one successor in T;
- an and-configuration with all successors in T.

In fact, we can dispense with the accept and reject states entirely by defining an *accept configuration* to be an and-configuration with no successors and a *reject configuration* to be an or-configuration with no successors. For accept configurations so defined, the condition "all successors in T" is vacuously satisfied.

Alternating machines are no more powerful than ordinary deterministic Turing machines, since an ordinary Turing machine can construct the computation tree of an alternating machine in a breadth-first fashion and check for the existence of a finite accepting subtree.

Alternating Machines with Negation

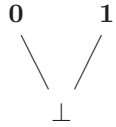
Alternating machines can also be augmented to allow *not-states* as well as and- and or-states. Perhaps contrary to expectation, these machines are no more powerful than ordinary Turing machines in terms of the sets they accept.

The formal definition of alternating machines with negation requires a bit of extra work. Here the function g is of type $Q \to \{\wedge, \vee, \neg\}$ and associates either $\wedge$ (and), $\vee$ (or), or $\neg$ (not) with each state. If $g(q) = \wedge$ (respectively, $\vee$, $\neg$), the state q is called an *and-state* (respectively, *or-state*, *not-state*) and the configuration (q, y, m, n) is called an *and-configuration* (respectively, *or-configuration*, *not-configuration*). A not-configuration is required to have exactly one successor. We also dispense with the accept and reject states, defining an *accept configuration* to be an and-configuration with no successors and a *reject configuration* to be an or-configuration with no successors.

Acceptance is defined formally in terms of a certain labeling ℓ_* of configurations with **1** (true), **0** (false), or $\perp$ (undefined). This labeling is defined as the $\sqsubseteq$-least solution ℓ of the equation

$$\ell(c) \stackrel{\text{def}}{=} \begin{cases} \bigwedge\{\ell(d) \mid c \xrightarrow[M,x]{1} d\}, & \text{if } g(c) = \wedge, \\ \bigvee\{\ell(d) \mid c \xrightarrow[M,x]{1} d\}, & \text{if } g(c) = \vee, \\ \neg\ell(d), & \text{if } g(c) = \neg \text{ and } c \xrightarrow[M,x]{1} d, \end{cases}$$

where $\sqsubseteq$ is the order defined by



i.e., $\perp \sqsubseteq \perp \sqsubseteq \mathbf{0} \sqsubseteq \mathbf{0}$ and $\perp \sqsubseteq \mathbf{1} \sqsubseteq \mathbf{1}$, and

$$\ell \sqsubseteq \ell' \iff \forall c\ \ell(c) \sqsubseteq \ell'(c),$$

and $\wedge$, $\vee$, and $\neg$ are computed on $\{\mathbf{0}, \perp, \mathbf{1}\}$ according to the following tables:

$\wedge$:	**0**	$\perp$	**1**
0	**0**	**0**	**0**
$\perp$	**0**	$\perp$	$\perp$
1	**0**	$\perp$	**1**

$\vee$:	**0**	$\perp$	**1**
0	**0**	$\perp$	**1**
$\perp$	$\perp$	$\perp$	**1**
1	**1**	**1**	**1**

$\neg$:	
0	**1**
$\perp$	$\perp$
1	**0**

In other words, $\wedge$ gives the greatest lower bound and $\vee$ gives the least upper bound

in the order $\mathbf{0} \leq \perp \leq \mathbf{1}$, and $\neg$ inverts the order.

The labeling ℓ_* can be computed as the $\sqsubseteq$-least fixpoint of the monotone map $\tau : \{\text{labelings}\} \to \{\text{labelings}\}$, where

$$\tau(\ell)(c) \stackrel{\text{def}}{=} \begin{cases} \bigwedge\{\ell(d) \mid c \xrightarrow[M,x]{1} d\}, & \text{if } g(c) = \wedge, \\ \bigvee\{\ell(d) \mid c \xrightarrow[M,x]{1} d\}, & \text{if } g(c) = \vee, \\ \neg\ell(d), & \text{if } g(c) = \neg \text{ and } c \xrightarrow[M,x]{1} d, \end{cases}$$

as provided by the Knaster–Tarski theorem (Section 1.7).

Universal Turing Machines and Undecidability

An important observation about Turing machines is that they can be *uniformly simulated*. By this we mean that there exist a special Turing machine U and a coding scheme that codes a complete description of each Turing machine M as a finite string x_M in such a way that U, given any such encoding x_M and a string y, can simulate the machine M on input y, accepting iff M accepts y. The machine U is called a *universal Turing machine*. Nowadays this is perhaps not so surprising, since we can easily imagine writing a Scheme interpreter in Scheme or a C compiler in C, but it was quite an advance when it was first observed by Turing in the 1930s; it led to the notion of the *stored-program computer*, the basic architectural paradigm underlying the design of all modern general-purpose computers today.

Undecidability of the Halting Problem

Recall that a set is *recursively enumerable* (r.e.) if it is $L(M)$ for some Turing machine M and *recursive* if it is $L(M)$ for some total Turing machine M (one that halts on all inputs, i.e., either accepts or rejects). A property φ is *decidable* (or *recursive*) if the set $\{x \mid \varphi(x)\}$ is recursive, *undecidable* if not.

Two classical examples of undecidable problems are the *halting problem* and the *membership problem* for Turing machines. Define

$$\mathsf{HP} \stackrel{\text{def}}{=} \{(x_M, y) \mid M \text{ halts on input } y\}$$
$$\mathsf{MP} \stackrel{\text{def}}{=} \{(x_M, y) \mid M \text{ accepts input } y\}.$$

These sets are r.e. but not recursive; in other words, it is undecidable for a given Turing machine M and input y whether M halts on y or whether M accepts y.

Proposition 2.2: The set HP is r.e. but not recursive.

Proof The set MP is r.e., because it is the set accepted by the universal Turing machine U. The set HP is r.e. as well, because a Turing machine can be constructed that on input (x_M, y) simulates M on input y using U, accepting if M either accepts or rejects y.

We show by contradiction that HP is not recursive. This argument is called a *diagonalization argument* and was first used by Cantor to show that the power set of a set A cannot be put in one-to-one correspondence with A. The reader will also probably notice the similarity to Russell's paradox (1.2.3).

Suppose for a contradiction that HP were recursive. Then there would be a *total* Turing machine K that decides for a given (x_M, y) whether M halts on y. Construct a Turing machine N that on input x interprets x as x_M, then determines whether M halts on input x_M by running K on (x_M, x_M). The machine K is total, so it either halts and accepts if M halts on input x_M or halts and rejects if M does not halt on input x_M. If K rejects, make N halt immediately. If K accepts, make N enter an infinite loop. Thus N halts on input x_M iff K rejects (x_M, x_M) iff M does not halt on input x_M. Now consider what happens when N is run on its own description x_N. By our construction, N halts on x_N iff N does not halt on x_N. This is a contradiction. ■

We will show by a different technique that the same proposition holds of MP (Example 2.14).

Most interesting questions about Turing machines turn out to be undecidable. For example, it is undecidable whether a given M accepts any string at all, whether M accepts a finite set, or whether M accepts a recursive set. In fact, *every* nontrivial property of r.e. sets is undecidable (Exercise 2.4).

2.2 Complexity Classes

By restricting the amount of time or space a Turing machine can use, we obtain various complexity classes. Most of these definitions are fairly robust in the sense that they are impervious to minor changes in the model, but extra care must be taken at lower levels of complexity.

Time and Space Complexity

Let $f : \mathbb{N} \to \mathbb{N}$ be a function. A (deterministic, nondeterministic, or alternating) Turing machine M is said to *run in time* $f(n)$ if for all sufficiently large n, all computation paths starting from the start configuration on any input of length n are of length at most $f(n)$. It is said to *run in space* $f(n)$ if for all sufficiently large

n, all configurations reachable from the start configuration on any input of length n use at most $f(n)$ worktape cells.

The machine M is said to run in time (respectively, space) $O(f(n))$ if it runs in time (respectively, space) $cf(n)$ for some constant c independent of n.

The machine M is said to run in *logarithmic space* if it runs in space $\log n$, where log denotes logarithm to the base 2. It is said to run in *polynomial time* if it runs in time $n^{O(1)}$; that is, if it runs in time n^k for some constant k independent of n. It is said to run in *exponential time* if it runs in time $2^{n^{O(1)}}$; that is, if it runs in time 2^{n^k} for some constant k independent of n. It is said to run in *double-exponential time* if it runs in time $2^{2^{n^{O(1)}}}$. It is said to run in *k-fold exponential time* if it runs in time $2 \uparrow_k n^{O(1)}$, where

$$\begin{aligned} 2 \uparrow_0 n &\stackrel{\text{def}}{=} n \\ 2 \uparrow_{k+1} n &\stackrel{\text{def}}{=} 2^{2 \uparrow_k n}. \end{aligned}$$

The corresponding space complexity bounds are defined analogously.

There is some disagreement in the literature as to the meaning of *exponential time*. It is often taken to mean time $2^{O(n)}$ instead of $2^{n^{O(1)}}$. We will use the latter definition, since it fits in better with results relating the complexity of deterministic and alternating Turing machines. However, in situations in which we can derive the stronger upper bound $2^{O(n)}$, we will do so and mention the bound explicitly.

The class $DTIME(f(n))$ (respectively, $NTIME(f(n))$, $ATIME(f(n))$) is the family of all sets $L(M)$ for deterministic (respectively, nondeterministic, alternating) Turing machines M running in time $f(n)$. The space complexity classes $DSPACE(f(n))$, $NSPACE(f(n))$, and $ASPACE(f(n))$ are defined similarly. We write $DTIME(n^{O(1)})$ for $\bigcup_{k \geq 0} DTIME(n^k)$, etc. A few common complexity classes have special notation:

$$\begin{aligned} LOGSPACE &\stackrel{\text{def}}{=} DSPACE(\log n) \\ NLOGSPACE &\stackrel{\text{def}}{=} NSPACE(\log n) \\ ALOGSPACE &\stackrel{\text{def}}{=} ASPACE(\log n) \\ PTIME &\stackrel{\text{def}}{=} DTIME(n^{O(1)}) \\ NPTIME &\stackrel{\text{def}}{=} NTIME(n^{O(1)}) \\ APTIME &\stackrel{\text{def}}{=} ATIME(n^{O(1)}) \\ PSPACE &\stackrel{\text{def}}{=} DSPACE(n^{O(1)}) \\ NPSPACE &\stackrel{\text{def}}{=} NSPACE(n^{O(1)}) \end{aligned}$$

$$
\begin{aligned}
APSPACE &\stackrel{\text{def}}{=} ASPACE(n^{O(1)}) \\
EXPTIME &\stackrel{\text{def}}{=} DTIME(2^{n^{O(1)}}) \\
NEXPTIME &\stackrel{\text{def}}{=} NTIME(2^{n^{O(1)}}) \\
AEXPTIME &\stackrel{\text{def}}{=} ATIME(2^{n^{O(1)}}) \\
EXPSPACE &\stackrel{\text{def}}{=} DSPACE(2^{n^{O(1)}}) \\
NEXPSPACE &\stackrel{\text{def}}{=} NSPACE(2^{n^{O(1)}}) \\
AEXPSPACE &\stackrel{\text{def}}{=} ASPACE(2^{n^{O(1)}}).
\end{aligned}
$$

These are the classes of all sets computable in, respectively: deterministic, nondeterministic, and alternating logarithmic space; deterministic, nondeterministic, and alternating polynomial time; deterministic, nondeterministic, and alternating polynomial space; deterministic, nondeterministic, and alternating exponential time; and deterministic, nondeterministic, and alternating exponential space.

The classes $PTIME$ and $NPTIME$ are more commonly known as P and NP, respectively.

A remarkable fact is that the following relationships hold among the deterministic and alternating complexity classes:

$$
\begin{array}{ccccccccc}
PTIME & \subseteq & PSPACE & \subseteq & EXPTIME & \subseteq & EXPSPACE & \subseteq & \cdots \\
\| & & \| & & \| & & \| & & \\
ALOGSPACE & \subseteq & APTIME & \subseteq & APSPACE & \subseteq & AEXPTIME & \subseteq & \cdots
\end{array}
$$

That is, the hierarchy of logarithmic space, polynomial time, polynomial space, exponential time, exponential space, double-exponential time, double-exponential space, etc. shifts by exactly one level when going from determinism to alternation.

One of the most important open problems in computational complexity is whether $P = NP$. Many important combinatorial optimization problems can be solved in nondeterministic polynomial time by a "guess and verify" method. Figuring out how to solve these problems efficiently without the guessing would have significant impact in real-world applications.

Oracle Machines and Relative Computability

It is sometimes useful to talk about computability *relative to a given set* B. The set B itself may not be computable, but we may be interested in what we could compute if we were given the power to test membership in B for free. One way to capture this idea formally is by *oracle Turing machines*. Another way will be discussed in Section 2.3.

An oracle Turing machine $M[\cdot]$ is like an ordinary deterministic Turing machine, except that it has three distinguished states, the *query state*, the *yes state*, and the *no state*, a finite *oracle alphabet* Δ, and a write-only tape called the *oracle query tape* on which the machine can write a string in Δ^*.

The machine can be equipped with an *oracle* $B \subseteq \Delta^*$, in which case we denote it by $M[B]$. This machine operates as follows. On input $x \in \Sigma^*$, $M[B]$ computes like an ordinary deterministic Turing machine, except that periodically it may decide to write a symbol on its oracle query tape. When it does so, the tape head is advanced one cell to the right. At some point, perhaps after writing several symbols on the oracle query tape, the machine may decide to enter its query state. When that happens, it automatically and immediately enters the yes state if $y \in B$ and the no state if $y \notin B$, where y is the string currently written on the oracle query tape. The oracle query tape is then automatically erased and the oracle query tape head returned to the left end of the tape. The contents of the worktape and the positions of the input and worktape heads are not altered. The machine then continues processing from that point. If the machine ever halts and accepts, then $x \in L(M[B])$. Note that the behavior of $M[B]$ may depend heavily on the oracle B.

An alternative formalism for oracle machines gives $M[B]$ an extra semi-infinite, two-way, read-only tape on which is written the characteristic function of B, where the elements of B are ordered in some reasonable way, perhaps lexicographically. If the machine wishes to know whether $y \in B$, it can scan out to the appropriate position on the tape containing the bit corresponding to y. The ordering on Δ^* should be sufficiently nice that this position is easily computable from y.

A set A is said to be *r.e. in* B if there is an oracle Turing machine $M[\cdot]$ such that $A = L(M[B])$; *co-r.e. in* B if $\sim A$ is r.e. in B; and *recursive in* B if there is an oracle Turing machine $M[\cdot]$ such that $A = L(M[B])$ and $M[B]$ is total (halts on all inputs). Again, whether or not $M[B]$ is total may depend heavily on the oracle B.

LEMMA 2.3: If A is r.e. in B, then so are the sets

$$\{z_1\#z_2\#\cdots\#z_n \mid \bigwedge_{i=1}^{n} z_i \in A\}$$

$$\{z_1\#z_2\#\cdots\#z_n \mid \bigvee_{i=1}^{n} z_i \in A\}.$$

Proof Exercise 2.1. ■

Recursive and R.E. Sets

In this section we state some basic facts about recursive and recursively enumerable (r.e.) sets that we will need in subsequent chapters. Recall that a set is *r.e.* if it is $L(M)$ for some Turing machine M and *recursive* if it is $L(M)$ for some total Turing machine M (one that halts on all inputs, i.e., either accepts or rejects). Define a set to be *co-r.e.* if its complement is r.e. A property φ is *decidable* (or *recursive*) if the set $\{x \mid \varphi(x)\}$ is recursive, *undecidable* if not.

PROPOSITION 2.4: A set is recursive iff it is both r.e. and co-r.e.

Proof If A is recursive, then a Turing machine accepting its complement $\sim A$ can be obtained by reversing the accept and reject states of a total Turing machine for A. Conversely, if both A and $\sim A$ are r.e., then a total Turing machine for A can be obtained by simulating a machine for A and a machine for $\sim A$ in parallel in a time-sharing fashion, accepting the input if the machine for A accepts and rejecting if the machine for $\sim A$ accepts; exactly one of those two events must occur. ■

Here is another useful characterization of the r.e. sets:

PROPOSITION 2.5: A set A is r.e. if and only if there exists a decidable dyadic predicate φ such that

$$A \;=\; \{x \mid \exists y\ \varphi(x, y)\}.$$

Proof If A has such a representation, then we can construct a Turing machine M for A that enumerates all y in some order, and for each one tests whether $\varphi(x, y)$, accepting if such a witness y is ever found. Conversely, if A is r.e., say $A = L(M)$, then we can take the recursive predicate $\varphi(x, y)$ to be "M accepts x in y steps." This predicate is decidable, since its truth can be determined by running M on x for y steps. ■

The Arithmetic Hierarchy

Propositions 2.4 and 2.5 are special cases of a more general relationship. Consider the following hierarchy of classes of sets, defined inductively as follows.

Recall from Section 2.2 that a set A is *r.e. in* B if there is an oracle Turing machine $M[\cdot]$ such that $A = L(M[B])$, *co-r.e. in* B if there is an oracle Turing machine $M[\cdot]$ such that $A = \sim L(M[B])$, and *recursive in* B if there is an oracle

Turing machine $M[\cdot]$ such that $A = L(M[B])$ and $M[B]$ halts on all inputs. Consider the following inductively defined hierarchy:

$$\begin{aligned}
\Sigma_1^0 &\stackrel{\text{def}}{=} \{A \mid A \text{ is r.e.}\} \\
\Pi_1^0 &\stackrel{\text{def}}{=} \{A \mid A \text{ is co-r.e.}\} \\
\Delta_1^0 &\stackrel{\text{def}}{=} \{A \mid A \text{ is recursive}\} \\
\Sigma_{n+1}^0 &\stackrel{\text{def}}{=} \{A \mid A \text{ is r.e. in } B \text{ for some } B \in \Sigma_n^0\} \\
&= \{A \mid A \text{ is r.e. in } B \text{ for some } B \in \Pi_n^0\} \\
\Pi_{n+1}^0 &\stackrel{\text{def}}{=} \{A \mid A \text{ is co-r.e. in } B \text{ for some } B \in \Sigma_n^0\} \\
&= \{A \mid A \text{ is co-r.e. in } B \text{ for some } B \in \Pi_n^0\} \\
\Delta_{n+1}^0 &\stackrel{\text{def}}{=} \{A \mid A \text{ is recursive in } B \text{ for some } B \in \Sigma_n^0\} \\
&= \{A \mid A \text{ is recursive in } B \text{ for some } B \in \Pi_n^0\}.
\end{aligned}$$

The following two theorems generalize Propositions 2.4 and 2.5, respectively.

THEOREM 2.6: For all $n \geq 0$, $\Delta_n^0 = \Sigma_n^0 \cap \Pi_n^0$.

Proof The proof is exactly like the proof of Proposition 2.4, except that all computations are done in the presence of an oracle. ■

THEOREM 2.7:

(i) $A \in \Sigma_n^0$ iff there exists a decidable $(n+1)$-ary predicate $\varphi(x, y_1, \ldots, y_n)$ such that

$$A = \{x \mid \exists y_1\ \forall y_2\ \exists y_3\ \ldots\ \mathsf{Q}_n y_n\ \varphi(x, y_1, \ldots, y_n)\},$$

where $\mathsf{Q}_i = \exists$ if i is odd, $\forall$ if i is even.

(ii) $A \in \Pi_n^0$ iff there exists a decidable $(n+1)$-ary predicate $\varphi(x, y_1, \ldots, y_n)$ such that

$$A = \{x \mid \forall y_1\ \exists y_2\ \forall y_3\ \ldots\ \mathsf{Q}_n y_n\ \varphi(x, y_1, \ldots, y_n)\},$$

where $\mathsf{Q}_i = \forall$ if i is odd, $\exists$ if i is even.

Proof We prove (i); statement (ii) follows from the fact that Π_n^0 is the class of all complements of sets in Σ_n^0.

We proceed by induction on n. The case $n = 1$ is given by Proposition 2.5. For

$n > 1$, assume first that

$$A = \{x \mid \exists y_1\ \forall y_2\ \exists y_3\ \ldots\ \mathsf{Q}_n y_n\ \varphi(x, y_1, \ldots, y_n)\}.$$

Let

$$B = \{(x, y_1) \mid \forall y_2\ \exists y_3\ \ldots\ \mathsf{Q}_n y_n\ \varphi(x, y_1, \ldots, y_n)\}.$$

By the induction hypothesis, $B \in \Pi^0_{n-1}$, and

$$A = \{x \mid \exists y_1\ (x, y_1) \in B\},$$

thus A is r.e. in B by an argument similar to that of Proposition 2.5.

Conversely, suppose $A = L(M[B])$ and $B \in \Pi^0_{n-1}$. Then $x \in A$ iff there exists a *valid computation history* y describing the computation of $M[B]$ on input x, including oracle queries and their responses, and all the responses to the oracle queries described in y are correct. Such a valid computation history might consist of a sequence of consecutive descriptions of configurations of the machine, each such configuration including a current state, tape contents, and tape head positions. Under a reasonable encoding of all this information, it is easy to check whether such a string obeys the rules of $M[\cdot]$; the only thing that is not checked easily is whether the results of the oracle queries (whether the machine enters the yes or the no state) are correct.

By Lemma 2.3, for any fixed k, the sets

$$U = \{z_1 \# z_2 \# \cdots \# z_k \mid \bigwedge_{i=1}^{k} z_i \in B\}$$

$$V = \{w_1 \# w_2 \# \cdots \# w_k \mid \bigwedge_{i=1}^{k} w_i \notin B\}$$

are in Π^0_{n-1} and Σ^0_{n-1}, respectively. By the induction hypothesis, membership in U and V can be represented by predicates with $n-1$ alternations of quantifiers beginning with $\forall$ (respectively, $\exists$) over a recursive predicate. Then the condition $x \in A$ is equivalent to the statement:

There exists a valid computation history y of $M[B]$ on input x such that if $z_1, \ldots, z_n$ are the strings that are queried of the oracle B in the computation of $M[B]$ on input x for which the response (as represented in the string y) is positive, and if $w_1, \ldots, w_m$ are the queries for which the response is negative, then $z_1 \# z_2 \# \cdots \# z_n \in U$ and $w_1 \# w_2 \# \cdots \# w_m \in V$.

By combining the representations of U and V and the recursive predicate "y is a valid computation history of $M[B]$ on input x," we can obtain a representation

of the predicate $x \in A$ with n alternations of quantifiers beginning with $\exists$ over a recursive predicate. ■

Kleene showed that the arithmetic hierarchy is strict: for all $n \geq 0$,

$$\Sigma_n^0 \cup \Pi_n^0 \quad \subset \quad \Delta_{n+1}^0,$$

and Σ_n^0 and Π_n^0 are incomparable with respect to set inclusion.

The Analytic Hierarchy

The arithmetic hierarchy relates to first-order number theory as the *analytic hierarchy* relates to *second-order number theory*, in which quantification over sets and functions is allowed. We will be primarily interested in the first level of this hierarchy, in particular the class Π_1^1 of relations over $\mathbb{N}$ definable with one universal second-order quantifier. A remarkable theorem due to Kleene states that this is exactly the class of relations over $\mathbb{N}$ definable by first-order induction. In this section we will provide a computational characterization of the classes Π_1^1 and Δ_1^1 and sketch a proof of Kleene's theorem.

Definitions of Π_1^1 and Δ_1^1

The class Π_1^1 is the class of all relations on $\mathbb{N}$ that can be defined by a prenex universal second-order number-theoretic formula. Here *prenex* means all quantifiers appear at the front of the formula and *universal* means only universal quantification over functions is allowed. Using various transformation rules to be discussed later in Section 3.4, we can assume every such formula is of the form

$$\forall f \; \exists y \; \varphi(\overline{x}, y, f),$$

where φ is quantifier free (Exercise 3.33). This formula defines the n-ary relation

$$\{\overline{a} \in \mathbb{N}^n \mid \forall f \; \exists y \; \varphi(\overline{a}, y, f)\}.$$

The class Δ_1^1 is the class of all relations on $\mathbb{N}$ that are Π_1^1 and whose complements are Π_1^1.

The Programming Language IND

We take a rather unusual approach to the subject of first-order inductive definability: we introduce a programming language IND and use it to define the inductive and hyperarithmetic sets and the recursive ordinals. This turns out to be equivalent to more conventional approaches (see for example Moschovakis (1974)), but has a

decidedly more computational flavor. Keep in mind that the relations "computed" by IND programs can be highly noncomputable.

An IND program consists of a finite sequence of labeled statements. Each statement is of one of three forms:

- assignment: $\ell:\ x := \exists \qquad \ell:\ y := \forall$
- conditional jump: $\ell:$ **if** $R(\overline{t})$ **then go to** ℓ'
- halt statement: $\ell:$ **accept** $\qquad \ell:$ **reject**.

The semantics of programs is very much like alternating Turing machines, except that the branching is infinite. The execution of an assignment statement causes countably many subprocesses to be spawned, each assigning a different element of $\mathbb{N}$ to the variable. If the statement is $x := \exists$, the branching is existential; if it is $y := \forall$, the branching is universal. The conditional jump tests the atomic formula $R(\overline{t})$, and if true, jumps to the indicated label. The **accept** and **reject** commands halt and pass a Boolean value back up to the parent. Computation proceeds as in alternating Turing machines: the input is an initial assignment to the program variables; execution of statements causes a countably branching computation tree to be generated downward, and Boolean accept (**1**) or reject (**0**) values are passed back up the tree, a Boolean $\vee$ being computed at each existential node and a Boolean $\wedge$ being computed at each universal node. The program is said to *accept* the input if the root of the computation tree ever becomes labeled with the Boolean value **1** on that input; it is said to *reject* the input if the root ever becomes labeled with the Boolean value **0** on that input; and it is said to *halt* on an input if it either accepts or rejects that input. An IND program that halts on all inputs is said to be *total*.

These notions are completely analogous to alternating Turing machines, so we forego the formalities in favor of some revealing examples.

First, we show how to simulate a few other useful programming constructs with those listed above. An unconditional jump

goto ℓ

is simulated by the statement

if $x = x$ **then go to** ℓ

More complicated forms of conditional branching can be effected by manipulation of control flow. For example, the statement

if $R(\overline{t})$ **then reject else** ℓ

is simulated by the program segment

if $R(\overline{t})$ **then go to** ℓ'
goto ℓ
ℓ': **reject**

A simple assignment is effected by guessing and verifying:

$x := y + 1$

is simulated by

$x := \exists$
if $x \neq y + 1$ **then reject**

The process spawns infinitely many subprocesses, all but one of which immediately reject!

Example 2.8: Any first-order relation is definable by a loop-free program. For example, the set of natural numbers x such that

$\exists y\ \forall z\ \exists w\ x \leq y \wedge x + z \leq w$

is defined by the program

$y := \exists$
$z := \forall$
$w := \exists$
if $x > y$ **then reject**
if $x + z \leq w$ **then accept**
reject

The converse is true too: any loop-free program defines a first-order relation.

However, IND can also define inductively definable relations that are not first-order.

Example 2.9: The reflexive transitive closure of a relation R is definable by the following program, which takes its input in the variables x, z and accepts if $(x, z) \in R^*$:

ℓ: **if** $x = z$ **then accept**
$y := \exists$

if $\neg R(x, y)$ **then reject**
$x := y$
go to ℓ

EXAMPLE 2.10: A *two-person perfect information game* consists of a binary relation **move** on a set of *boards*. The two players alternate. If the current board is x and it is player I's turn, player I chooses y such that $\mathbf{move}(x, y)$; then player II chooses z such that $\mathbf{move}(y, z)$; and so on. A player wins by *checkmate*, i.e., by forcing the opponent into a position from which there is no legal next move. Thus a checkmate position is a board y such that $\forall z\ \neg\mathbf{move}(y, z)$.

We would like to know for a given board x whether the player whose turn it is has a forced win from x. Ordinarily this might be defined as the least solution **win** of the recursive equation

$$\mathbf{win}(x) \iff \exists y\ (\mathbf{move}(x, y) \wedge \forall z\ \mathbf{move}(y, z) \to \mathbf{win}(z)).$$

The base case involving an immediate win by checkmate is included: if y is a checkmate position, then the subformula $\forall z\ \mathbf{move}(y, z) \to \mathbf{win}(z)$ is vacuously true. The least solution to this recursive equation is the least fixpoint of the monotone map τ defined by

$$\tau(R) \stackrel{\text{def}}{\iff} \{x \mid \exists y\ \mathbf{move}(x, y) \wedge \forall z\ \mathbf{move}(y, z) \to R(z)\}$$

(see Section 1.7). We can express $\mathbf{win}(x)$ with an IND program as follows:

ℓ: $y := \exists$
if $\neg\mathbf{move}(x, y)$ **then reject**
$x := \forall$
if $\neg\mathbf{move}(y, x)$ **then accept**
go to ℓ

EXAMPLE 2.11: Our last example involves well-founded relations. As observed in Section 1.3, induction and well-foundedness go hand in hand. Here is an IND program that tests whether a strict partial order $<$ is well-founded:

$x := \forall$
ℓ: $y := \forall$
if $\neg y < x$ **then accept**
$x := y$
go to ℓ

This program halts and accepts if all descending chains are finite (see Exercise 1.15).

Any property that is expressed as a least fixpoint of a monotone map defined by a positive first-order formula can be computed by an IND program. Here is what we mean by this. Let R be an n-ary relation symbol, and let $\varphi(\overline{x}, R)$ be a first-order formula with free individual variables $\overline{x} = x_1, \ldots, x_n$ and free relation variable R. Assume further that all free occurrences of R in φ are *positive*; that is, they occur in the scope of an even number of negation symbols $\neg$. For any n-ary relation B, define

$$\tau(B) \quad = \quad \{\overline{a} \mid \varphi(\overline{a}, B)\}.$$

That is, we think of φ as representing a set operator τ mapping a set of n-tuples B to another set of n-tuples $\{\overline{a} \mid \varphi(\overline{a}, B)\}$. One can show that the positivity assumption implies that the set operator τ is monotone, therefore it has a least fixpoint F_φ, which is an n-ary relation (see Section 1.7). The traditional treatment of inductive definability defines a first-order inductive relation as a projection of such a fixpoint; that is, a relation of the form

$$\{a_1, \ldots, a_m \mid F_\varphi(a_1, \ldots, a_m, b_{m+1}, \ldots, b_n)\},$$

where $b_{m+1}, \ldots, b_n$ are fixed elements of the structure. Given the formula φ and the elements $b_{m+1}, \ldots, b_n$, one can construct an IND program that assigns $b_{m+1}, \ldots, b_n$ to the variables $x_{m+1}, \ldots, x_n$, then checks whether the values of $x_1, \ldots, x_n$ satisfy F_φ by decomposing the formula top-down, executing existential assignments at existential quantifiers, executing universal assignments at universal quantifiers, using control flow for the propositional connectives, using conditional tests for the atomic formulas, and looping back to the top of the program at occurrences of the inductive variable R. The examples above involving reflexive transitive closure, games, and well-foundedness illustrate this process.

Conversely, any relation computed by an IND program is inductive in the traditional sense, essentially because the definition of acceptance for IND programs involves the least fixpoint of an inductively defined set of labelings of the computation tree.

Inductive and Hyperelementary Relations

Many of the sample IND programs of the previous section make sense when interpreted over any structure, not just $\mathbb{N}$. We define the *inductive relations* of any

structure $\mathfrak{A}$ to be those relations computable by IND programs over $\mathfrak{A}$. We define the *hyperelementary relations* of $\mathfrak{A}$ to be those relations computable by *total* IND programs over $\mathfrak{A}$, i.e., programs that halt on all inputs. Note that every first-order relation is hyperelementary, since it is computed by a loop-free program.

One can show that a relation over $\mathfrak{A}$ is hyperelementary iff it is both inductive and coinductive. If there is an IND program that accepts R and another IND program that accepts $\sim R$, then one can construct a total IND program that runs the two other programs in parallel, much as in the proof of Proposition 2.4.

Now we restrict our attention to the structure of arithmetic $\mathbb{N}$. Over this structure, the hyperelementary relations are sometimes called the *hyperarithmetic relations*.

Recursive Trees, Recursive Ordinals, and ω_1^{ck}

An ordinal α is *countable* if there exists a one-to-one function $f : \alpha \to \omega$. The ordinals $\omega \cdot 2$ and ω^2, although greater than ω, are still countable. The smallest uncountable ordinal is called ω_1.

Traditionally, a *recursive ordinal* is defined as one for which there exists a *computable* one-to-one function from it to ω under some suitable encoding of ordinals and notion of computability (see Rogers (1967)). The smallest nonrecursive ordinal is called ω_1^{ck}. It is a countable ordinal, but it looks uncountable to any computable function.

We will define recursive ordinals in terms of inductive labelings of *recursive trees*. For the purposes of this chapter, a *tree* is a nonempty prefix-closed subset of ω^*. In other words, it is a set T of finite-length strings of natural numbers such that

- $\varepsilon \in T$;
- if $xy \in T$ then $x \in T$.

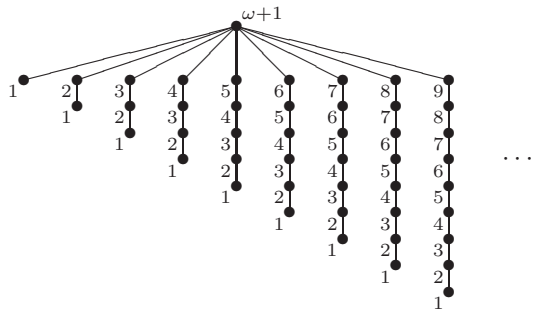
A *path* in T is a maximal subset of T linearly ordered by the prefix relation. The tree T is *well-founded* if it has no infinite paths; equivalently, if the converse of the prefix relation is a well-founded relation on T. A *leaf* is an element of T that is not a prefix of any other element of T.

Given a well-founded tree T, we define a labeling $\mathrm{ord} : T \to \mathbf{Ord}$ inductively as follows:

$$\mathrm{ord}(x) \stackrel{\mathrm{def}}{=} (\sup_{\substack{n\in\omega \\ xn\in T}} \mathrm{ord}(xn)) + 1.$$

Thus $\mathrm{ord}(x) = 1$ if x is a leaf; otherwise, $\mathrm{ord}(x)$ is determined by first determining $\mathrm{ord}(xn)$ for all $xn \in T$, then taking the supremum and adding 1.

For example, consider the tree consisting of ε and all sequences of the form $(n, \underbrace{0, 0, \ldots, 0}_{m})$ for $n \geq 0$ and $m \leq n$. The leaves are labeled 1 by ord, the next elements above the leaves are labeled 2, and so on. The root ε is labeled $\omega + 1$.



For a well-founded tree T, let $\mathrm{ord}(T)$ be the ordinal assigned to the root of T. Every $\mathrm{ord}(T)$ is a countable ordinal, and $\sup_T \mathrm{ord}(T) = \omega_1$.

Now define an ordinal to be *recursive* if it is $\mathrm{ord}(T)$ for some *recursive* tree T; that is, a tree such that the set T, suitably encoded, is a recursive set. The supremum of the recursive ordinals is ω_1^{ck}.

An alternative definition of recursive ordinals is the set of all running times of IND programs. The running time of an IND program on some input is the time it takes to label the root of the computation tree with 1 or 0. This is the closure ordinal of the inductive definition of labelings of the computation tree in the formal definition of acceptance. It is very similar to the definition of the labelings ord of recursive trees. The ordinal ω_1^{ck} is the supremum of all running times of IND programs.

Kleene's Theorem

THEOREM 2.12 (KLEENE): Over $\mathbb{N}$, the inductive relations and the Π^1_1 relations coincide, and the hyperelementary and Δ^1_1 relations coincide.

Proof sketch. First we show that every inductive relation is Π^1_1. This direction holds in any structure $\mathfrak{A}$, not just $\mathbb{N}$. Let $\varphi(\overline{x}, R)$ be a positive first-order formula with fixpoint $F_\varphi \subseteq A^n$, where A is the carrier of $\mathfrak{A}$. We can describe F_φ as the

intersection of all relations closed under φ:

$$F_\varphi(\overline{x}) \iff \forall R\ (\forall \overline{y}\ \varphi(\overline{y}, R) \to R(\overline{y})) \to R(\overline{x}).$$

This is a Π^1_1 formula.

Conversely, consider any Π^1_1 formula over $\mathbb{N}$. As mentioned earlier, we can assume without loss of generality that the formula is of the form

$$\forall f\ \exists y\ \varphi(x, y, f), \tag{2.2.1}$$

where f ranges over functions $\omega \to \omega$, y ranges over ω, and φ is quantifier free (Exercise 3.33).

Regarding a function $f : \omega \to \omega$ as the infinite string of its values $f(0)$, $f(1)$, $f(2), \ldots$, the functions f are in one-to-one correspondence with paths in the complete tree ω^*. Moreover, for any x and y, the truth of $\varphi(x, y, f)$ is determined by any finite prefix of this path that includes all arguments to f corresponding to terms appearing in $\varphi(x, y, f)$. Let $f \restriction n$ denote the finite prefix of f of length n. We can think of $f \restriction n$ either as a string of natural numbers of length n or as a partial function that agrees with f on domain $\{0, 1, \ldots, n-1\}$. Instead of (2.2.1) we can write

$$\forall f\ \exists y\ \exists n\ \varphi'(x, y, f \restriction n), \tag{2.2.2}$$

where φ' is just φ modified slightly to evaluate to **0** (false) in case n is too small to give enough information to determine whether $\varphi(x, y, f)$. Note that if $\varphi'(x, y, f \restriction n)$, then $\varphi'(x, y, f \restriction m)$ for all $m \geq n$. This says that (2.2.2) is essentially a well-foundedness condition: if we label the vertices $f \restriction n$ of the infinite tree with the truth value of $\exists y\ \varphi'(x, y, f \restriction n)$, (2.2.2) says that along every path in the tree we eventually encounter the value **1** (true). And as observed in Example 2.11, well-foundedness is inductive.

We have shown that the inductive and Π^1_1 relations over $\mathbb{N}$ coincide. Since the hyperarithmetic relations are those that are both inductive and coinductive and the Δ^1_1 relations are those that are both Π^1_1 and Σ^1_1, the hyperarithmetic and Δ^1_1 relations coincide as well. ■

Inductive is Existential over Hyperelementary

We have shown that over $\mathbb{N}$, Π^1_1 is exactly the family of sets accepted by IND programs and Δ^1_1 is the family of sets accepted by total IND programs. It is apparent from this characterization that there is a strong analogy between the inductive and the r.e. sets and between the hyperelementary and the recursive sets.

It may seem odd that that the class analogous to Σ_1^0 at the analytic level should be Π_1^1 and not Σ_1^1. This is explained by a result analogous to Proposition 2.5.

PROPOSITION 2.13: A set $A \subseteq \mathbb{N}$ is inductive iff there is a hyperelementary relation φ such that

$$\begin{aligned} A &= \{x \mid \exists \alpha < \omega_1^{\mathrm{ck}}\ \varphi(x, \alpha)\} \\ &= \{x \mid \exists y\ y \text{ encodes a recursive ordinal and } \varphi(x, y)\}. \end{aligned} \tag{2.2.3}$$

Proof sketch. If φ is hyperelementary, then we can build an IND program for (2.2.3) consisting of the statement $y := \exists$ followed by a program that in parallel checks that the Turing machine with index y accepts a well-founded recursive tree and that $\varphi(x, y)$.

Conversely, if A is inductive, say accepted by an IND program p, then we can describe A by an existential formula that says, "There exists a recursive ordinal α such that p halts and accepts x in α steps." More concretely, one would say, "There exists a recursive well-founded tree T such that on input x, p halts and accepts in $\mathrm{ord}(T)$ steps." The quantification is then over indices of Turing machines. The predicate "p halts and accepts x in $\mathrm{ord}(T)$ steps" is hyperelementary, since one can construct an IND program that runs p together with a program q that simply enumerates the tree T using existential branching, rejecting at the leaves. The program q rejects all inputs, but takes $\mathrm{ord}(T)$ steps to do it. If the simulations of p and q are performed in parallel in a time-sharing fashion as in Theorem 2.4, one step at a time in each turn, then it can be shown by induction on the computation tree that the simulating machine will accept or reject depending on which of p or q takes less (ordinal) time. ■

2.3 Reducibility and Completeness

Reducibility Relations

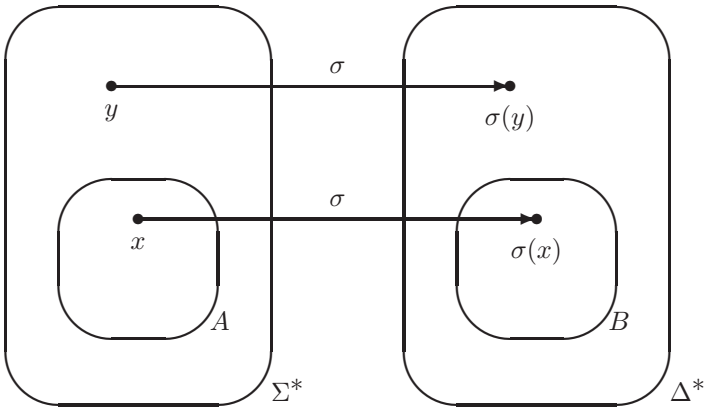
Reducibility is a common technique for comparing the complexity of different problems. Given decision problems $A \subseteq \Sigma^*$ and $B \subseteq \Delta^*$, a *(many-one) reduction* of A to B is a total computable function

$$\sigma : \Sigma^* \rightarrow \Delta^*$$

such that for all $x \in \Sigma^*$,

$$x \in A \iff \sigma(x) \in B. \tag{2.3.1}$$

In other words, strings in A must go to strings in B under σ, and strings not in A must go to strings not in B under σ. Intuitively, instances of the problem A are coded by σ as instances of the problem B. We may not know how to decide whether a given string x is in A or not, but we can apply σ to x to transform it into an instance $\sigma(x)$ of the problem B. Then a decision procedure for B would immediately give a decision procedure for A by composing it with σ.



The function σ need not be one-to-one or onto. It must, however, be *total* and *effectively computable*; that is, computable by a total Turing machine that on any input x halts with $\sigma(x)$ written on its tape. When such a reduction exists, we say that A is *reducible* to B via the map σ, and we write $A \leq_{\mathrm{m}} B$. The subscript m, which stands for "many-one," is used to distinguish this relation from other types of reducibility relations.

In order to obtain an *efficient* decision procedure for A from an *efficient* decision procedure for B, the reduction map σ must also be efficient. Many known reductions in the literature turn out to be very efficient, usually linear time or logspace. We write $A \leq_{\mathrm{m}}^{\log} B$ if the reduction map σ is computable in logspace and $A \leq_{\mathrm{m}}^{\mathrm{p}} B$ if the reduction map σ is computable in polynomial time.

The reducibility relations $\leq_{\mathrm{m}}$, $\leq_{\mathrm{m}}^{\log}$, and $\leq_{\mathrm{m}}^{\mathrm{p}}$ are transitive: if $A \leq_{\mathrm{m}} B$ and $B \leq_{\mathrm{m}} C$, then $A \leq_{\mathrm{m}} C$, and similarly for $\leq_{\mathrm{m}}^{\log}$ and $\leq_{\mathrm{m}}^{\mathrm{p}}$. This is because if σ reduces A to B and τ reduces B to C, then their composition $\sigma \circ \tau$ reduces A to C. For the relations $\leq_{\mathrm{m}}^{\log}$ and $\leq_{\mathrm{m}}^{\mathrm{p}}$, we must show that the composition of polynomial-time

computable functions is computable in polynomial time and the composition of logspace computable functions is computable in logpspace. For logspace, this is not immediate, since there is not enough space to write down an intermediate result; but with a little cleverness it can be done (Exercise 2.7).

EXAMPLE 2.14: One can reduce HP, the halting problem for Turing machines, to the membership problem MP, the problem of determining whether a given Turing machine accepts a given string (see Section 2.1). This is done by constructing from a given description x_M of a Turing machine M and string y a description x_N of a Turing machine N that accepts ε iff M halts on y. In this example,

$$\begin{aligned} A &= \mathsf{HP} \stackrel{\text{def}}{=} \{(x_M, y) \mid M \text{ halts on input } y\}, \\ B &= \mathsf{MP} \stackrel{\text{def}}{=} \{(x_M, y) \mid M \text{ accepts input } y\}. \end{aligned}$$

Given x_M and y, let N be a Turing machine that on input z does the following:

(i) erases its input z;

(ii) writes y on its tape (y is hard-coded in the finite control of N);

(iii) runs M on y (the description x_M of M is hard-coded in the finite control of N);

(iv) accepts if the computation of M on y halts.

The machine N we have constructed accepts its input z iff M halts on y. Moreover, its actions are independent of z, since it just ignores its input. Thus

$$L(N) = \begin{cases} \Sigma^*, & \text{if } M \text{ halts on } y, \\ \varnothing, & \text{if } M \text{ does not halt on } y. \end{cases}$$

In particular,

$$N \text{ accepts } \varepsilon \iff M \text{ halts on } y. \tag{2.3.2}$$

We can take our reduction σ to be the computable map $(x_M, y) \mapsto (x_N, \varepsilon)$. In this example, σ is computable in polynomial time and even in logspace. The equation (2.3.2) is just (2.3.1) in this particular case.

Here are some general results about reducibility relations that point out their usefulness in comparing the computability and complexity of decision problems.

THEOREM 2.15:

(i) If $A \leq_{\mathrm{m}} B$ and B is r.e., then so is A. Equivalently, if $A \leq_{\mathrm{m}} B$ and A is not r.e., then neither is B.

(ii) If $A \leq_{\mathrm{m}} B$ and B is recursive, then so is A. Equivalently, if $A \leq_{\mathrm{m}} B$ and A is not recursive, then neither is B.

Proof (i) Suppose $A \leq_{\mathrm{m}} B$ via the map σ and B is r.e. Let M be a Turing machine such that $B = L(M)$. Build a machine N for A as follows: on input x, first compute $\sigma(x)$, then run M on input $\sigma(x)$, accepting if M accepts. Then

$$\begin{array}{lcll} N \text{ accepts } x & \iff & M \text{ accepts } \sigma(x) & \text{definition of } N \\ & \iff & \sigma(x) \in B & \text{definition of } M \\ & \iff & x \in A & \text{by (2.3.1).} \end{array}$$

(ii) Recall from Proposition 2.4 that a set is recursive iff both it and its complement are r.e. Suppose $A \leq_{\mathrm{m}} B$ via the map σ and B is recursive. Note that $\sim A \leq_{\mathrm{m}} \sim B$ via the same σ. If B is recursive, then both B and $\sim B$ are r.e. By (i), both A and $\sim A$ are r.e., thus A is recursive. ■

We can use Theorem 2.15(i) to show that certain sets are not r.e. and Theorem 2.15(ii) to show that certain sets are not recursive. To show that a set B is not r.e., we need only give a reduction from a set A we already know is not r.e., such as the complement of the halting problem, to B. By Theorem 2.15(i), B cannot be r.e. For example, the reduction of Example 2.14, in conjunction with Proposition 2.2, shows that the membership problem MP is not decidable and that $\sim\mathsf{MP}$ is not r.e.

A similar theorem holds in the presence of complexity bounds.

THEOREM 2.16:

(i) If $A \leq_{\mathrm{m}}^{\mathrm{p}} B$ and B is computable in polynomial time, then so is A. In other words, the complexity class P is closed downward under $\leq_{\mathrm{m}}^{\mathrm{p}}$.

(ii) If $A \leq_{\mathrm{m}}^{\log} B$ and B is computable in logspace, then so is A. In other words, the complexity class $LOGSPACE$ is closed downward under $\leq_{\mathrm{m}}^{\log}$.

Proof Exercise 2.8. ■

Completeness

A set B is said to be *hard* for a complexity class $\mathcal{C}$ with respect to a reducibility relation $\leq$ (or just $\mathcal{C}$-hard, if $\leq$ is understood) if $A \leq B$ for all $A \in \mathcal{C}$. The set B is said to be *complete* for $\mathcal{C}$ with respect to $\leq$ (or just $\mathcal{C}$-complete) if it is hard for $\mathcal{C}$ with respect to $\leq$ and if in addition $B \in \mathcal{C}$.

Intuitively, if B is complete for $\mathcal{C}$, then B is a "hardest" problem for the class $\mathcal{C}$ in the sense that it is in $\mathcal{C}$ and encodes every other problem in $\mathcal{C}$.

For example, the Boolean satisfiability problem—to determine, given a propositional formula, whether it is satisfiable—is *NP*-complete with respect to $\leq_{\mathrm{m}}^{\log}$. This is known as Cook's theorem (see Hopcroft and Ullman (1979); Kozen (1991b)).

The following proposition points out the significance of these concepts.

Proposition 2.17: Let $\mathcal{B}$ and $\mathcal{C}$ be complexity classes, $\mathcal{B} \subseteq \mathcal{C}$. Suppose also that $\mathcal{B}$ is closed downward under the reducibility relation $\leq$; that is, if $A \leq B$ and $B \in \mathcal{B}$, then $A \in \mathcal{B}$. If a set B is $\mathcal{C}$-complete with respect to $\leq$, then $B \in \mathcal{B}$ if and only if $\mathcal{B} = \mathcal{C}$.

In other words, the question of whether the two complexity classes are equal reduces to the question of whether the single problem B is in $\mathcal{B}$. For example, $P = NP$ if and only if the Boolean satisfiability problem is in P.

Proof If $\mathcal{B} = \mathcal{C}$, then $B \in \mathcal{B}$, since $B \in \mathcal{C}$. Conversely, suppose $B \in \mathcal{B}$. Since B is $\mathcal{C}$-hard, every element of $\mathcal{C}$ reduces to B; and since $\mathcal{B}$ is closed downward under $\leq$, all those elements are in $\mathcal{B}$. Thus $\mathcal{C} \subseteq \mathcal{B}$. ■

The complexity class *coNP* is the class of sets $A \subseteq \Sigma^*$ whose complements $\sim A = \Sigma^* - A$ are in *NP*. Usually *NP*-hardness and *coNP*-hardness are taken with respect to the reducibility relation $\leq_{\mathrm{m}}^{\mathrm{p}}$.

Proposition 2.18:

(i) $A \leq_{\mathrm{m}}^{\mathrm{p}} B$ iff $\sim A \leq_{\mathrm{m}}^{\mathrm{p}} \sim B$.

(ii) A is *NP*-hard iff $\sim A$ is *coNP*-hard.

(iii) A is *NP*-complete iff $\sim A$ is *coNP*-complete.

(iv) If A is *NP*-complete then $A \in coNP$ iff $NP = coNP$.

It is unknown whether $NP = coNP$.

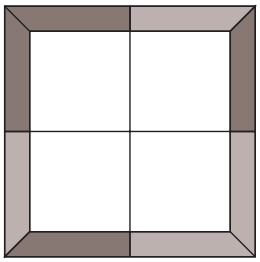
Tiling Problems

In this section we describe a family of *tiling problems* that are complete for various complexity classes. These problems will serve as generic problems that we can use in establishing other completeness results by reduction.

Let C be a finite set of *colors*. A *tile* is a square with colored sides. The *type* of a tile is a mapping $\{\text{north}, \text{south}, \text{east}, \text{west}\} \to C$ that gives the color of each side.

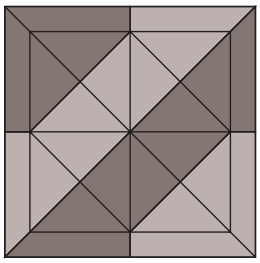
Say we are given a set of tile types and a square $n \times n$ grid consisting of n^2 cells, each of which can hold one tile. The boundary of the grid is colored with colors from C. We would like to know whether it is possible to place tiles in the cells, one tile to each cell, such that the colors on all adjacent edges match. We can use as many tiles of each of the given types as we like, but we are not allowed to rotate the tiles.

For example, the tiling problem with 2×2 grid



and tile types

has exactly one solution, namely



We call this problem the *first tiling problem.*

PROPOSITION 2.19: The first tiling problem is *NP*-complete.

Proof The problem is in *NP*, since we can guess a tiling and verify quickly that the color constraints are satisfied.

To show that the problem is *NP*-hard, we show how to encode the computation of an arbitrary one-tape nondeterministic polynomial-time-bounded Turing machine on some input as an instance of a tiling problem. Let M be such a machine and let x be an input to M, $n = |x|$. Without loss of generality, the single tape is read/write and serves as both input and worktape. Say the machine M runs in time n^k for some fixed $k \geq 1$ independent of n, and let $N = n^k$. The grid will be $N \times N$. The sequence of colors along the south edges of the tiles in the j^{th} row will represent a possible configuration of M at time j. The color of the south edge of the tile at position i, j will give the symbol occupying the i^{th} tape cell and will indicate whether that cell is being scanned at time j, and if so, will give the current state. For example, this color might say, "M is currently in state q scanning this tape cell, and the symbol currently occupying this cell is a." The color of the north edge will represent similar information at time $j + 1$. The types of the tiles will be chosen so that only legal moves of the machine are represented. Because the tape head can move left and right, information must also move sideways, and the east/west colors are used for that. The east/west colors will say whether the head is crossing the line between this cell and an adjacent one. The possible colors on the north edge of a tile will be determined by the colors of the other three edges and the transition rules of M. The colors along the south boundary of the grid will describe the start configuration, and those along the north boundary will describe the accept configuration. The resulting tiling problem will have a solution iff M has an accepting computation on input x.

Formally, let Q be the set of states, Σ the input alphabet, Γ the work alphabet, $\sqcup$ the blank symbol, $\vdash$ the left endmarker, s the start state, t the accept state, and δ the transition relation. We assume that $\Sigma \subseteq \Gamma, \vdash, \sqcup \in \Gamma - \Sigma$, and

$$\delta \quad \subseteq \quad (Q \times \Gamma) \times (Q \times \Gamma \times \{\text{left,right}\}).$$

If $((p, a), (q, b, d)) \in \delta$, this says that when the machine is in state p scanning symbol a, it can write b on the current tape cell, move in direction d, and enter state q. The north/south colors are

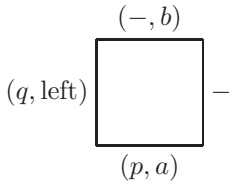
$$(Q \cup \{-\}) \times \Gamma$$

and the east/west colors are

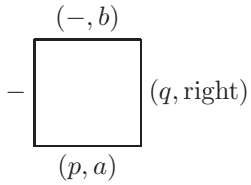
$$(Q \times \{\text{left,right}\}) \cup \{-\}.$$

The north/south color (q, a) for $q \in Q$ indicates that the tape head is currently scanning this tape cell, the machine is in state q, and the symbol currently written on this tape cell is a. The north/south color $(-, a)$ indicates that the tape head is not currently scanning this tape cell, and the symbol currently written on this tape cell is a. The east/west color (q, d) indicates that the head is crossing the line between these two tape cells in direction d and is about to enter state q. The east/west color $-$ indicates that the head is not currently crossing the line between these two tape cells.

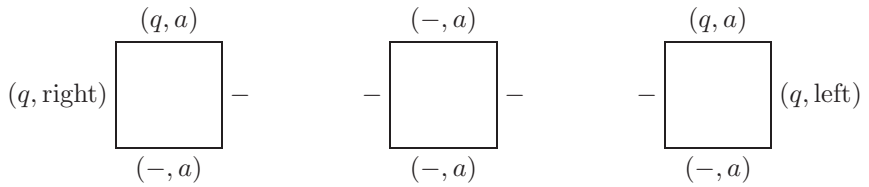
The types of the tiles will be all types of the form



whenever $((p, a), (q, b, \text{left})) \in \delta$ and



whenever $((p, a), (q, b, \text{right})) \in \delta$, as well as all types of the form



The colors along the south boundary of the grid will represent the start configuration of M on input $x = a_1 a_2 \cdots a_n$:

$(s, \vdash)\ (-, a_1)\ (-, a_2)\ \cdots\ (-, a_n)\ (-, \sqcup)\ \cdots\ (-, \sqcup)$.

The colors along the north boundary of the grid will represent the accept configuration of M:

$(t, \vdash)\ (-, \sqcup)\ (-, \sqcup)\ \cdots\ (-, \sqcup)$

(we can assume without loss of generality that M erases its tape and moves its head all the way to the left before accepting). The colors along the east and west boundaries of the grid are all $-$.

Now any accepting computation history of M on input x yields a tiling, and any tiling represents an accepting computation history, because the local consistency conditions of the tiling say that M starts in its start configuration on input x, runs according to the transition rules δ, and ends up in an accept configuration. Therefore this instance of the tiling problem has a solution iff M accepts x. ■

Now consider a variant of the tiling problem in which we use an infinite $\omega \times \omega$ grid. The south boundary is colored with a pattern consisting of a finite string of colors followed by an infinite string of a single color, say blue. The west boundary is colored only blue. The coloring of the south and west boundaries is part of the problem specification. There is no north or east boundary. Everything else is the same as above. We call this problem the *second tiling problem*.

PROPOSITION 2.20: The second tiling problem is Π_1^0-complete (that is, co-r.e.-complete). The problem is still Π_1^0-hard even if we restrict the south boundary to be colored with a single color.

Proof sketch. The problem is in Π_1^0, since the whole grid can be tiled if and only if all southwest $n \times n$ subgrids can be tiled.[1] This is a Π_1^0 statement.

To show that the problem is Π_1^0-hard, we construct an instance of the tiling

1 This is not obvious! The proof uses König's lemma. See Exercise 2.6.

problem that simulates a given deterministic Turing machine on a given input as in the proof of Proposition 2.19, except that there is no bound on the size or number of configurations. All tile types with the color of the accept state on the south edge are omitted; thus the tiling can be extended indefinitely if and only if the machine does not accept the input. For any fixed Turing machine M, this construction constitutes a many-one reduction from $\sim L(M)$ to solvable instances of the tiling problem.

To show that a single color for the south boundary suffices, we could instead encode the set

$$\{x_M \mid M \text{ does not accept the empty string } \varepsilon\}$$

by constructing for a given x_M an instance of the tiling problem that simulates M on the empty input. This is a well-known Π_1^0-complete problem (Exercise 2.3). ■

We next consider a slightly different version of the problem. In this version, we still restrict colorings of the south boundary to consist of a finite string of colors followed by an infinite string of blue, but the coloring of the south boundary is not specified. The problem is to decide whether there exists such a coloring of the south boundary under which the tiling can be extended indefinitely. This variant is called the *third tiling problem*.

PROPOSITION 2.21: The third tiling problem is Σ_2^0-complete.

Proof sketch. The problem is in Σ_2^0, since the selection of the boundary coloring requires a single existential quantifier; then the statement that the grid can be tiled under that boundary coloring is Π_1^0, as in Proposition 2.20.

To show that the problem is Σ_2^0-hard, we can encode the complement of the universality problem for Turing machines. The universality problem is: given a Turing machine, does it accept all strings? In other words, is it the case that for all input strings, there exists a halting computation on that input string? This is a well-known Π_2^0-complete problem.

As in the proof of Proposition 2.20, we can construct from a given Turing machine an instance of the tiling problem that simulates the Turing machine. Here, however, the various colorings of the south boundary will represent the possible input strings. The finite multicolored part of the boundary coloring will represent the input string, and the infinite blue string to its right will represent infinitely many blank symbols on the Turing machine's tape to the right of the input. For a given boundary coloring, the tiling can be extended indefinitely iff the Turing machine on the input string corresponding to that boundary coloring does not halt. ■

As a final variant, consider the question of whether an $\omega \times \omega$ tiling problem has a solution in which a particular color, say red, is used infinitely often. We call this version the *fourth tiling problem*.

PROPOSITION 2.22: The fourth tiling problem is Σ_1^1-complete.

Proof sketch. The problem is in Σ_1^1 because it can be expressed with a second-order existential formula: a second-order existential quantifier can be used to select a tiling; and for a given tiling, being a solution and using red infinitely often are first-order properties.

To show that the problem is Σ_1^1-hard, we reduce to it the non-well-foundedness of recursive trees $T \subseteq \omega^*$. The construction given in the proof of Kleene's theorem (Theorem 2.12) shows that this problem is Σ_1^1-hard. Build a Turing machine that, given a recursive tree T, guesses a sequence $n_0, n_1, n_2, \ldots$ nondeterministically. After the k^{th} guess, it checks whether the sequence $n_0, n_1, n_2, \ldots, n_{k-1}$ guessed so far is in T. If so, it enters a special red state. The tree is non-well-founded iff it has an infinite path, which happens iff there is a computation of the machine that enters the red state infinitely often. This can be coded as an instance of the fourth tiling problem as above. ■

2.4 Bibliographical Notes

Turing machines were introduced by Turing (1936). Originally they were presented in the form of *enumeration machines*, since Turing was interested in enumerating the decimal expansions of computable real numbers and values of real-valued functions. Turing also introduced the concept of nondeterminism in his original paper, although he did not develop the idea. Alternating Turing machines were introduced in Chandra et al. (1981).

The basic properties of the r.e. sets were developed by Kleene (1943) and Post (1943, 1944). Universal Turing machines and the application of Cantor's diagonalization technique to prove the undecidability of the halting problem appeared in the original paper of Turing (1936). Reducibility relations were discussed by Post (1944). These fundamental ideas led to the development of *recursive function theory*; see Rogers, Jr. (1967); Soare (1987); Kleene (1952) for an introduction to this field.

Counter automata were studied by Fischer (1966), Fischer et al. (1968), and Minsky (1961).

Recursive function theory has been extended upward and downward. The upward extension deals with the arithmetic and analytic hierarchies, so-called *generalized* or *α-recursion theory*, descriptive set theory, and inductive definability. See Rogers, Jr. (1967); Soare (1987); Barwise (1975); Moschovakis (1974, 1980) for an introduction to these subjects. Kleene's theorem on the relation between inductive definability and Π^1_1 was proved in Kleene (1955). The programming language IND was introduced in Harel and Kozen (1984).

The downward extension is computational complexity theory. This subject got its start in the late 1960s and early 1970s; some seminal papers are Hartmanis and Stearns (1965); Karp (1972); Cook (1971). A good source on the theory of *NP*-completeness and completeness for other complexity classes is Garey and Johnson (1979). The tiling problems of Section 2.3 are from Harel (1985).

Exercises

2.1. Prove Lemma 2.3.

2.2. Prove that ω_1 is the smallest set of ordinals containing 0 and closed under successor and suprema of countable sets.

2.3. Show that the set

$\{x_M \mid M \text{ accepts the empty string } \varepsilon\}$

is Σ^0_1-complete. (*Hint.* Use a reduction similar to the one of Example 2.14.) Use this to argue that the tiling problem of Proposition 2.20 remains Π^0_1-hard even if the south boundary is restricted to be colored with a single color.

2.4. (Rice (1953, 1956)) A property $\varphi(x)$ of strings is called a *property of the r.e. sets* if for all Turing machines M and N, if $L(M) = L(N)$ then $\varphi(x_M) = \varphi(x_N)$, where x_M is the code of M as described in Section 2.1. It is *nontrivial* if there exist M and N such that $\varphi(x_M)$ is true and $\varphi(x_N)$ is false. Examples of nontrivial properties of r.e. sets are: "M accepts a finite set," "$L(M)$ is recursive," and "M accepts the empty string." Examples of properties that are not nontrivial properties of the r.e. sets are: "M has more than 21 states" (not a property of r.e. sets) and "there exists a Turing machine accepting twice as many strings as M" (not nontrivial). Prove that *every* nontrivial property of the r.e. sets is undecidable. (*Hint.* Encode the halting problem HP. Let Z and N be Turing machines such that $L(Z) = \varnothing$ and

$\varphi(x_N) \neq \varphi(x_Z)$. Build a Turing machine that accepts $L(N)$ if a given M halts on x_M and $\varnothing$ otherwise.)

2.5. (a) A tree $T \subseteq \omega^*$ is *finitely branching* if for every $x \in T$ there are only finitely many $n \in \omega$ such that $xn \in T$. Prove *König's lemma*: every finitely branching tree with infinitely many vertices has an infinite path.
(b) Give a counterexample showing that (a) is false without the finite-branching assumption.

2.6. In the second tiling problem (Proposition 2.20), we needed to know that the whole $\omega \times \omega$ grid can be tiled if and only if all southwest $n \times n$ subgrids can be tiled. Show that this is so. (*Hint.* Use either Exercise 2.5 or the compactness of propositional logic.)

2.7. Prove that the reducibility relation $\leq_{\mathrm{m}}^{\log}$ defined in 2.3 is transitive.

2.8. Prove Theorem 2.16.

2.9. Show that Proposition 2.22 holds even if the problem requires infinitely many occurrences of red on the westernmost column of tiles.

2.10. (a) Prove that the halting problem for one-counter Turing machines is decidable.
(b) Using (a), show that there exists an r.e. set that is accepted by no one-counter machine. Conclude that one-counter machines are strictly less powerful than arbitrary Turing machines.

2.11. Prove that the halting problem for IND programs is Π_1^1-complete.

3 Logic

3.1 What is Logic?

Logic typically consists of three ingredients:

• A *language* is a collection of well-formed expressions to which meaning can be assigned. The symbols of the language, together with the formal rules for distinguishing well-formed expressions from arbitrary aggregates of symbols, are called the *syntax* of the language.

• A *semantics* tells how to interpret well-formed expressions as statements *about* something. The "something" can be mathematical objects such as groups or graphs or the natural numbers or everyday things such as cars, employees, or the weather. The statements of the language talk about the properties of and relationships among these objects.

• A *deductive system* consists of a collection of rules that can be applied to derive, in a purely mechanical way, interesting facts about and relationships among the semantic objects. The facts and relationships are those expressible in the language.

All three of these aspects—language, semantics, and deductive system—can be adapted to particular applications and to particular levels of expressibility as desired.

In this chapter we will introduce several classical logical systems:

• *propositional logic*, the logic of uninterpreted assertions and the basic propositional connectives: *and*, *or*, *not*, *if... then...*, and *if and only if* (Section 3.2);

• *equational logic*, the logic of equality (Section 3.3);

• *first-order predicate logic*, the logic of individual elements and their properties, especially those properties expressible using universal and existential quantification (Section 3.4);

• *infinitary logic*, a variant of predicate logic allowing certain infinite expressions (Section 3.6);

• *modal logic*, the logic of *possibility* and *necessity* (Section 3.7).

For each system, we will discuss its syntax, semantics, and deductive apparatus and derive some elementary results.

Languages

The language of a given logic typically consists of an *application-specific* part and an *application-independent* part. The application-specific part consists of those constructs that are tailored to the application at hand and may not make sense in other contexts. For example, when talking about properties of the natural numbers $\mathbb{N} = \{0, 1, 2, \ldots\}$, a symbol $+$ denoting addition and a symbol $\cdot$ denoting multiplication are quite relevant, but less so if we are discussing properties of graphs. For this reason, the language of number theory and the language of graph theory look very different. In a medical expert system, the objects under discussion may be diseases, treatments, symptoms, and medications, and their relationships with one another. These differences might be called "horizontal differences" among logical systems.

There are also "vertical differences." Even within a specific application domain, there may be several possible levels of expressiveness, depending on the complexity of the properties that need to be expressed and reasoned about. A good rule of thumb when designing a logic is to make it no more detailed than necessary to handle the task at hand. This allows irrelevant information to be suppressed, so that it cannot clutter or confuse the process of deduction. If propositional letters and propositional connectives suffice to express an assertion, it is pointless to include, say, variables, constants and functions; if we are trying to establish the equality of two terms, we might as well ignore quantifiers. Classical mathematical logic—the logic of mathematical objects such as groups, graphs, or topological spaces—divides neatly into various levels of expressiveness, reflecting these differences. The same will hold true for Dynamic Logic.

Models, Satisfaction, and Validity

Nowadays, logicians generally take pains to develop formal semantics for new logics they invent. This was not always so. Up until about the 1940s, logic existed in purely syntactic form—the "sacred" form, as it is called by van Dalen (1994). Logicians did not think much about formal semantics or interpretation, but just worked mechanically with the syntax and proof apparatus, guided by intuition. But it was always clear that expressions had meaning, at least on some level. Number theory was always about the natural numbers and set theory about sets, even though we might not have been completely sure what these objects were.

A more recent approach, the *model theoretic* approach, attempts to define the meaning of expressions rigorously as true/false statements about formally defined mathematical objects. These objects are usually called *structures* or *models*.

Included in the specification of the structure is a mapping that tells how to interpret the basic symbols of the language in that structure. Certain syntactic expressions are designated as *sentences*, which are formulas that take on well-defined truth values under a given interpretation mapping. This "profane" view of logic is one of the great contributions of Alfred Tarski.

If a sentence φ is true in a structure $\mathfrak{A}$, then we say that φ is *satisfied* in the structure $\mathfrak{A}$ or that $\mathfrak{A}$ is a *model* of φ, and write $\mathfrak{A} \vDash \varphi$. If φ is satisfied in every possible structure, we say that φ is *valid* and write $\vDash \varphi$.

In some instances we wish to limit the class of structures under consideration. Often the class of structures of interest is itself specified by a set of sentences. For example, in group theory, we are interested in groups, which are mathematical objects satisfying certain properties that can be expressed in the language of groups. If Φ is a set of sentences (finite or infinite), we say that $\mathfrak{A}$ is a model of Φ and write $\mathfrak{A} \vDash \Phi$ if $\mathfrak{A}$ is a model of every element of Φ. For example, Φ might be the set of sentences defining groups; by definition, a structure is a group iff it is a model of this set. We write $\Phi \vDash \varphi$ if φ is satisfied in every model of Φ, and say that φ is a *logical consequence* of Φ. The set of logical consequences of Φ is called the *theory of* Φ and is denoted $\mathbf{Th}\,\Phi$. For example, group theory is the set of logical consequences of the set of sentences defining groups; that is, the set of all sentences that are true in all groups.

There are modern logical systems for which the semantics is not fully defined, either because set theory does not provide adequate support or there is not complete agreement on how expressions should be interpreted. Indeed, there exist systems for which it is still not understood how to give any complete and rigorous formal interpretation at all.

Deduction

Many different types of deductive systems have been proposed for the various logics we will consider in this book: sequent systems, natural deduction, tableau systems, and resolution, to name a few. Each of these systems has its advantages and disadvantages.

For uniformity and consistency, we will concentrate on one style of deductive system called a *Hilbert system* after the mathematician David Hilbert, who advocated its use in mechanizing mathematics. A Hilbert system consists of a set of *axioms*, or sentences in the language that are postulated to be true, and *rules of inference* of the form

$$\frac{\varphi_1, \quad \varphi_2, \quad \ldots, \quad \varphi_n}{\psi}$$

from which new theorems can be derived. The statements $\varphi_1, \ldots, \varphi_n$ above the line are called the *premises* of the rule and the statement ψ below the line is called the *conclusion*. In Hilbert systems, usually the axioms are emphasized and the rules of inference are few and very basic.

A *proof* in a Hilbert system is a sequence $\varphi_1, \ldots, \varphi_m$ of statements such that each φ_i is either an axiom or the conclusion of a rule all of whose premises occur earlier in the sequence. The sequence $\varphi_1, \ldots, \varphi_m$ is said to be a *proof of* φ_m, and φ_m is called a *theorem* of the system. We write $\vdash \varphi$ if φ is a theorem.

More generally, we may wish to reason in the presence of extra assumptions. If Φ is a set of sentences (finite or infinite), we write $\Phi \vdash \varphi$ if there is a proof of φ using the elements of Φ as if they were extra axioms. In other words, $\Phi \vdash \varphi$ if there is a sequence $\varphi_1, \ldots, \varphi_m$ of statements such that each φ_i is either an axiom, an element of Φ, or the conclusion of a rule all of whose premises occur earlier in the sequence, and $\varphi = \varphi_m$. If $\Phi \vdash \varphi$, we say that φ is a *deductive consequence* of Φ. A theorem is just a deductive consequence of the empty set of assumptions.

Soundness and Completeness

A deductive system $\vdash$ is said to be *sound* with respect to a semantics $\models$ if for all sentences φ,

$$\vdash \varphi \implies \models \varphi;$$

that is, every theorem is valid. A deductive system $\vdash$ is said to be *complete* with respect to $\models$ if for all φ,

$$\models \varphi \implies \vdash \varphi;$$

that is, every valid sentence is a theorem.

Consistency and Refutability

Many logics have a negation operator $\neg$. For such logics, we say that a formula φ is *refutable* if $\neg\varphi$ is a theorem; that is, if $\vdash \neg\varphi$. If the logic contains a conjunction operator $\wedge$ as well, then we say that a set Φ of sentences is *refutable* if some finite conjunction of elements of Φ is refutable. We say that φ or Φ is *consistent* if it is not refutable.

Axiom Schemes

In many cases, axioms and rules are given as *schemes*, which are rules standing for infinitely many *instances*. For example, the rule

$$\frac{\varphi, \quad \psi}{\varphi \wedge \psi}$$

says that from the truth of φ and ψ we can infer that the single statement $\varphi \wedge \psi$ is true. This really stands for infinitely many rules, one for each possible choice of φ and ψ. The rules themselves are the instances of this scheme obtained by substituting a particular φ and ψ. For example,

$$\frac{p \to q, \quad q \to p}{(p \to q) \wedge (q \to p)}$$

might be one such instance.

3.2 Propositional Logic

We often need to make basic deductions such as:

If p implies q, and if q is false, then p must also be false.

This deduction is valid independent of the truth or falsity of p and q. Propositional logic formalizes this type of reasoning.

Syntax

The basic symbols of propositional logic are the *propositional letters* $p, q, r, \ldots$ representing atomic assertions, which can be either true or false. We assume that there are countably many such symbols available.

In addition to these symbols, there are *propositional operators* or *connectives*

- $\wedge$ *conjunction, "and"*;
- $\vee$ *disjunction, "or"*;
- $\neg$ *negation, "not"*;
- $\mathbf{1}$ *truth*;
- $\mathbf{0}$ *falsity*;
- $\to$ *implication, "if... then..."*;
- $\leftrightarrow$ *equivalence, "if and only if", "iff"*;

and parentheses. We will actually take only $\to$ and $\mathbf{0}$ as primitive symbols and define all the others in terms of them.

Propositions or *propositional formulas*, denoted $\varphi, \psi, \rho, \ldots$, are built up inductively according to the following rules:

- all atomic propositions $p, q, r, \ldots$ and $\mathbf{0}$ are propositions;
- if φ and ψ are propositions, then so is $\varphi \to \psi$.

We define the other propositional operators as follows:

$$\begin{aligned}
\neg\varphi &\stackrel{\text{def}}{=} \varphi \to \mathbf{0} \\
\mathbf{1} &\stackrel{\text{def}}{=} \neg\mathbf{0} \\
\varphi \vee \psi &\stackrel{\text{def}}{=} (\neg\varphi) \to \psi \\
\varphi \wedge \psi &\stackrel{\text{def}}{=} \neg((\neg\varphi) \vee (\neg\psi)) \\
\varphi \leftrightarrow \psi &\stackrel{\text{def}}{=} (\varphi \to \psi) \wedge (\psi \to \varphi).
\end{aligned}$$

Note that by replacing left-hand sides with right-hand sides, we can systematically remove the operators $\leftrightarrow$, $\wedge$, $\vee$, $\mathbf{1}$, and $\neg$ and reduce every formula to a formula over $\to$ and $\mathbf{0}$ only.

Parentheses and Precedence

We parenthesize using the symbols ( ) where necessary to ensure unique readability. For example, if we just write

$$p \vee q \wedge r \tag{3.2.1}$$

it is not clear whether we intend

$$(p \vee q) \wedge r \qquad \text{or} \qquad p \vee (q \wedge r); \tag{3.2.2}$$

and these two expressions have very different meaning.

However, if we used parentheses everywhere, they would quickly get out of hand. We can avoid the proliferation of parentheses by assigning a *precedence* to the connectives, which tells which ones bind more tightly than others. The precedence is:

- the negation symbol $\neg$ has highest precedence (binds most tightly);
- the conjunction and disjunction symbols $\wedge$ and $\vee$ have next highest and equal precedence;

- the implication symbol $\rightarrow$ has next highest precedence; and
- the equivalence symbol $\leftrightarrow$ has lowest precedence.

For example, the expression

$$\neg\varphi \rightarrow \psi \quad \leftrightarrow \quad \neg\psi \rightarrow \varphi \tag{3.2.3}$$

should be read

$$((\neg\varphi) \rightarrow \psi) \quad \leftrightarrow \quad ((\neg\psi) \rightarrow \varphi).$$

If we want (3.2.3) to be read another way, say

$$(\neg((\varphi \rightarrow \psi) \leftrightarrow \neg\psi)) \quad \rightarrow \quad \varphi,$$

then we have to use parentheses.

We associate symbols of equal precedence from left to right. Thus the expression (3.2.1) should be read as the left-hand expression in (3.2.2).

In the text, we often also use spacing to help with readability, as we have done in (3.2.3); but formally, spacing has no significance.

Metasymbols, or symbols that we use as abbreviations for English phrases in our discourse, always have lower precedence than symbols in the language under study. For example, the meta-expression

$$\vDash \psi \quad \Longrightarrow \quad \vDash \varphi \rightarrow \psi$$

says, "if ψ is valid, then so is $\varphi \rightarrow \psi$," as if written

$$(\vDash \psi) \quad \Longrightarrow \quad (\vDash (\varphi \rightarrow \psi))$$

with parentheses. The symbol $\rightarrow$, which is a propositional connective, has higher precedence than the metasymbols $\vDash$ and $\Longrightarrow$. It is important to distinguished the propositional implication symbol $\rightarrow$ from the meta-implication symbol $\Longrightarrow$. The distinction is necessary because we are *using* propositional logic even as we define it.

Semantics

The truth or falsity of a proposition depends on the truth or falsity of the atomic propositions appearing in it. For example, the proposition $p \wedge q$ (read: "p and q") is true iff both of the propositions p and q are true; and the proposition $\neg p$ (read: "not p") is true iff p is false.

There are two possible *truth values*, which we denote by **0** (false) and **1** (true).

We can think of the atomic propositions $p, q, r, \ldots$ as variables ranging over the set $\{\mathbf{0}, \mathbf{1}\}$. Any assignment of truth values to the atomic propositions appearing in a proposition φ automatically determines a truth value for φ inductively, as described formally below.

We define a *truth assignment* to be a map

$$u : \{p, q, r, \ldots\} \quad \rightarrow \quad \{\mathbf{0}, \mathbf{1}\}.$$

The value $u(p)$ is the *truth value* of the atomic proposition p under the truth assignment u. Any such map extends inductively to all propositions as follows:

$$\begin{aligned} u(\mathbf{0}) &\stackrel{\text{def}}{=} \mathbf{0} \\ u(\varphi \rightarrow \psi) &\stackrel{\text{def}}{=} \begin{cases} \mathbf{1}, & \text{if } u(\varphi) = \mathbf{0} \text{ or } u(\psi) = \mathbf{1} \\ \mathbf{0}, & \text{otherwise.} \end{cases} \end{aligned}$$

It follows that

$$\begin{aligned} u(\varphi \wedge \psi) &= \begin{cases} \mathbf{1}, & \text{if } u(\varphi) = u(\psi) = \mathbf{1} \\ \mathbf{0}, & \text{otherwise} \end{cases} \\ u(\varphi \vee \psi) &= \begin{cases} \mathbf{1}, & \text{if } u(\varphi) = \mathbf{1} \text{ or } u(\psi) = \mathbf{1} \\ \mathbf{0}, & \text{otherwise} \end{cases} \\ u(\neg\varphi) &= \begin{cases} \mathbf{0}, & \text{if } u(\varphi) = \mathbf{1} \\ \mathbf{1}, & \text{otherwise} \end{cases} \\ u(\mathbf{1}) &= \mathbf{1} \\ u(\varphi \leftrightarrow \psi) &= \begin{cases} \mathbf{1}, & \text{if } u(\varphi) = u(\psi) \\ \mathbf{0}, & \text{otherwise.} \end{cases} \end{aligned}$$

We say that the truth assignment u *satisfies* φ if $u(\varphi) = \mathbf{1}$, and write $u \vDash \varphi$ and $u(\varphi) = \mathbf{1}$ interchangeably. If Φ is any set of propositions, finite or infinite, we say that u *satisfies* Φ and write $u \vDash \Phi$ if u satisfies all the propositions in Φ. A proposition or set of propositions is *satisfiable* if there is a truth assignment that satisfies it.

The formula φ is said to be *valid* if $u \vDash \varphi$ for all u. A valid formula is also called a *(propositional) tautology*. We write $\vDash \varphi$ to indicate that φ is a tautology. A tautology is a formula that is always true, no matter what the truth values of its atomic propositions are.

Observe that $\vDash \varphi$ iff $\neg\varphi$ is not satisfiable.

EXAMPLE 3.1: The following are some examples of basic tautologies:

(i) $\varphi \vee \neg\varphi$
(ii) $\neg\neg\varphi \leftrightarrow \varphi$
(iii) $\psi \rightarrow (\varphi \rightarrow \psi)$
(iv) $\varphi \rightarrow \psi \leftrightarrow \neg\varphi \vee \psi$
(v) $\varphi \leftrightarrow \psi \leftrightarrow (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$
(vi) $\neg\varphi \wedge \neg\psi \leftrightarrow \neg(\varphi \vee \psi)$
(vii) $\varphi \wedge (\psi \vee \rho) \leftrightarrow (\varphi \wedge \psi) \vee (\varphi \wedge \rho)$
(viii) $\varphi \vee (\psi \wedge \rho) \leftrightarrow (\varphi \vee \psi) \wedge (\varphi \vee \rho)$
(ix) $\varphi \wedge \varphi \leftrightarrow \varphi$
(x) $\varphi \vee \varphi \leftrightarrow \varphi$
(xi) $\varphi \rightarrow \psi \leftrightarrow \neg\psi \rightarrow \neg\varphi$
(xii) $\neg\varphi \leftrightarrow \varphi \rightarrow \mathbf{0}$
(xiii) $\varphi \vee \psi \leftrightarrow \neg(\neg\varphi \wedge \neg\psi)$
(xiv) $\varphi \wedge \psi \leftrightarrow \neg(\neg\varphi \vee \neg\psi)$
(xv) $(\varphi \wedge \psi) \vee (\neg\varphi \wedge \rho) \leftrightarrow (\varphi \rightarrow \psi) \wedge (\neg\varphi \rightarrow \rho)$.

If φ and ψ take the same truth values on all truth assignments, we say that φ and ψ are *equivalent* and write $\varphi \equiv \psi$. Note that φ and ψ are equivalent iff $\varphi \leftrightarrow \psi$ is a tautology.

We have defined all the propositional connectives in terms of $\rightarrow$ and $\mathbf{0}$. This was by no means the only possible choice of a primitive set of connectives. For example, because of (iv), we could have defined $\rightarrow$ in terms of $\neg$ and $\vee$. A set of connectives is called *complete* if every formula is equivalent to a formula containing only those connectives. For example, the sets $\{\rightarrow, \mathbf{0}\}$, $\{\vee, \neg\}$, and $\{\wedge, \neg\}$ are all complete. We study some of these properties in the exercises (Exercises 3.2 and 3.3).

Although there are infinitely many propositional letters, it follows from the definition that $u(\varphi)$ depends only on $u(p)$ for p appearing in φ. This observation gives a decision procedure for satisfiability and validity:

THEOREM 3.2: Given any φ, it is decidable whether φ is satisfiable.

Proof Suppose φ contains propositional letters $p_1, \ldots, p_n$. For each possible truth assignment $u : \{p_1, \ldots, p_n\} \rightarrow \{\mathbf{0}, \mathbf{1}\}$, compute $u(\varphi)$ inductively according to the rules given above. Then φ is satisfiable iff $u(\varphi) = \mathbf{1}$ for at least one such u. ■

COROLLARY 3.3: It is decidable whether φ is valid.

Proof Check whether $\neg\varphi$ is satisfiable. ■

The decision problem of Theorem 3.2 is the Boolean satisfiability problem discussed in Section 2.3. The decision procedure given in the proof of that theorem, naively implemented, takes exponential time in the size of φ, since there are 2^n possible truth assignments $u : \{p_1, \ldots, p_n\} \to \{\mathbf{0}, \mathbf{1}\}$. The question of whether there exists a polynomial-time algorithm is equivalent to the $P = NP$ problem (see Section 2.3).

Set-Theoretic Representation

Let S be a set. The propositional operators $\vee$, $\wedge$, $\neg$, $\mathbf{0}$, and $\mathbf{1}$ behave very much like certain set-theoretic operators on subsets of S, namely $\cup$ (union), $\cap$ (intersection), $\sim$ (complementation in S), $\varnothing$ (emptyset), and S, respectively. This correspondence is more than just coincidental. If we take S to be set of truth assignments $u : \{p, q, r, \ldots\} \to \{\mathbf{0}, \mathbf{1}\}$ and define

$$\varphi' \quad = \quad \{u \in S \mid u \vDash \varphi\}$$

then the map $'$, which takes propositions to subsets of S, is a *homomorphism* with respect to these operators:

THEOREM 3.4:

$$\begin{aligned}
(\varphi \wedge \psi)' &= \varphi' \cap \psi' \\
(\varphi \vee \psi)' &= \varphi' \cup \psi' \\
(\neg\varphi)' &= S - \varphi' \\
\mathbf{1}' &= S \\
\mathbf{0}' &= \varnothing.
\end{aligned}$$

Moreover,

$$\begin{aligned}
\vDash \varphi &\iff \varphi' = S \\
\varphi \text{ is satisfiable} &\iff \varphi' \neq \varnothing \\
\vDash \varphi \to \psi &\iff \varphi' \subseteq \psi' \\
\vDash \varphi \leftrightarrow \psi &\iff \varphi \equiv \psi \iff \varphi' = \psi'.
\end{aligned}$$

Proof Exercise 3.1. ■

A Deductive System

We will discuss two Hilbert-style deductive systems for propositional logic and prove their soundness and completeness. For the sake of simplicity, our first system will be rather meager. Later we will consider a richer system that includes all the propositional operators and that is much easier to work with in practice; but for the sake of proving completeness, we restrict our attention to formulas over $\rightarrow$ and $\mathbf{0}$ only. This assumption is without loss of generality, since the other operators are defined from these.

Our system consists of three axioms and a rule of inference.

Axiom System 3.5:

(S) $(\varphi \rightarrow (\psi \rightarrow \sigma)) \ \rightarrow \ ((\varphi \rightarrow \psi) \rightarrow (\varphi \rightarrow \sigma))$

(K) $\varphi \ \rightarrow \ (\psi \rightarrow \varphi)$

(DN) $((\varphi \rightarrow \mathbf{0}) \rightarrow \mathbf{0}) \rightarrow \varphi$

(MP) $\dfrac{\varphi, \quad \varphi \rightarrow \psi}{\psi}$.

The axiom (DN) is called the *law of double negation*. Considering $\neg\varphi$ as an abbreviation for $\varphi \rightarrow \mathbf{0}$, this law takes the form $\neg\neg\varphi \rightarrow \varphi$. The rule of inference (MP) is called *modus ponens*.

The following are some sample derivations in this system.

Example 3.6: Let us start off with something very simple: $\varphi \rightarrow \varphi$. Here is a proof. Let $Q \stackrel{\text{def}}{=} \varphi \rightarrow \varphi$.

(i) $\varphi \rightarrow (Q \rightarrow \varphi)$

(ii) $\varphi \rightarrow Q$

(iii) $(\varphi \rightarrow (Q \rightarrow \varphi)) \ \rightarrow \ ((\varphi \rightarrow Q) \rightarrow (\varphi \rightarrow \varphi))$

(iv) $(\varphi \rightarrow Q) \rightarrow (\varphi \rightarrow \varphi)$

(v) $\varphi \rightarrow \varphi$.

Statements (i) and (ii) are both instances of (K); (iii) is an instance of (S); (iv) follows from (i) and (iii) by modus ponens; and (v) follows from (ii) and (iv) by modus ponens.

EXAMPLE 3.7: We prove the transitivity of implication:

$$(\psi \to \sigma) \quad \to \quad ((\varphi \to \psi) \to (\varphi \to \sigma)). \tag{3.2.4}$$

Let

$$P \stackrel{\text{def}}{=} \psi \to \sigma$$
$$Q \stackrel{\text{def}}{=} \varphi \to (\psi \to \sigma)$$
$$R \stackrel{\text{def}}{=} (\varphi \to \psi) \to (\varphi \to \sigma).$$

The theorem (3.2.4) we would like to prove is $P \to R$. Here is a proof.

(i) $P \to Q$
(ii) $Q \to R$
(iii) $(Q \to R) \ \to \ (P \to (Q \to R))$
(iv) $P \to (Q \to R)$
(v) $(P \to (Q \to R)) \ \to \ ((P \to Q) \to (P \to R))$
(vi) $(P \to Q) \to (P \to R)$
(vii) $P \to R$.

Statement (i) is an instance of (K); (ii) is an instance of (S); (iii) is an instance of (K); (iv) follows from (ii) and (iii) by modus ponens; (v) is an instance of (S); (vi) follows from (iv) and (v) by modus ponens; and (vii) follows from (i) and (vi) by modus ponens.

EXAMPLE 3.8: We show that the statement

(EFQ) $\quad \mathbf{0} \to \varphi$

is a theorem. The name EFQ stands for *e falso quodlibet* ("from falsity, anything you like"). Let us abbreviate $\varphi \to \mathbf{0}$ by $\neg\varphi$. Here is a proof of (EFQ):

(i) $\mathbf{0} \to \neg\neg\varphi$
(ii) $\neg\neg\varphi \to \varphi$
(iii) $(\neg\neg\varphi \to \varphi) \to ((\mathbf{0} \to \neg\neg\varphi) \to (\mathbf{0} \to \varphi))$
(iv) $(\mathbf{0} \to \neg\neg\varphi) \to (\mathbf{0} \to \varphi)$
(v) $\mathbf{0} \to \varphi$.

Statement (i) is an instance of (K), since it is really $\mathbf{0} \to (\neg\varphi \to \mathbf{0})$; (ii) is just (DN); (iii) is an instance of the theorem proved in Example 3.7; (iv) follows from

(ii) and (iii) by modus ponens; and (v) follows from (i) and (iv) by modus ponens.

If we omit the axiom (DN) and take (K), (S), and (EFQ) as axioms along with the rule (MP), we get a weaker system called *intuitionistic propositional logic*. The propositional tautology (DN) is not provable in this system; see Exercise 3.5.

The Deduction Theorem

Here is a useful theorem about this system that has to do with reasoning in the presence of assumptions. It says that if the proposition ψ can be derived in the presence of an extra assumption φ, then the proposition $\varphi \to \psi$ can be derived without any assumptions; that is, the assumption φ can be coded into the theorem itself.

THEOREM 3.9 (DEDUCTION THEOREM): Let Φ be a finite or infinite set of propositions. Then

$$\Phi \cup \{\varphi\} \vdash \psi \iff \Phi \vdash \varphi \to \psi.$$

Proof First suppose $\Phi \vdash \varphi \to \psi$. Certainly $\Phi \cup \{\varphi\} \vdash \varphi \to \psi$. Also $\Phi \cup \{\varphi\} \vdash \varphi$ by a one-line proof. Therefore $\Phi \cup \{\varphi\} \vdash \psi$ by modus ponens. This was the easy direction.

Conversely, suppose $\Phi \cup \{\varphi\} \vdash \psi$. We proceed by induction on the length of proofs to show that $\Phi \vdash \varphi \to \psi$. Consider the last step in a proof of ψ under the assumptions $\Phi \cup \{\varphi\}$. If $\psi \in \Phi$, then $\Phi \vdash \psi$, and $\Phi \vdash \psi \to (\varphi \to \psi)$ by (K), therefore $\Phi \vdash \varphi \to \psi$ by modus ponens. If $\psi = \varphi$, then $\Phi \vdash \varphi \to \varphi$ by the theorem proved in Example 3.6. Finally, if ψ is the conclusion of an application of modus ponens, then there is a σ such that $\Phi \cup \{\varphi\} \vdash \sigma \to \psi$ and $\Phi \cup \{\varphi\} \vdash \sigma$ by shorter proofs. By the induction hypothesis, $\Phi \vdash \varphi \to (\sigma \to \psi)$ and $\Phi \vdash \varphi \to \sigma$. By (S),

$$\Phi \vdash (\varphi \to (\sigma \to \psi)) \quad \to \quad ((\varphi \to \sigma) \to (\varphi \to \psi));$$

then by two applications of modus ponens, $\Phi \vdash \varphi \to \psi$. ■

Completeness

In this section we prove the completeness of Axiom System 3.5. First, we observe that the system is sound, since the axioms (K), (S), and (DN) are tautologies and the rule (MP) preserves validity; therefore by induction, every formula φ such that $\vdash \varphi$ is a tautology. The more interesting part is the converse: every tautology has a proof in this system.

First we prove a preliminary lemma whose proof contains most of the work. We will also find this lemma useful later on when we study compactness. Recall from Section 3.1 that a finite or infinite set Φ of formulas is *refutable* if $\neg\varphi$ is a theorem, where φ is some finite conjunction of elements of Φ. In light of Theorem 3.9, this is equivalent to saying that $\Phi \vdash \mathbf{0}$. The set Φ is *consistent* if it is not refutable.

LEMMA 3.10: If Φ is consistent, then it is satisfiable.

Proof Suppose Φ is consistent. Then Φ is contained in a setwise maximal consistent set $\widehat{\Phi}$; that is, a consistent set such that $\Phi \subseteq \widehat{\Phi}$ and any proper superset of $\widehat{\Phi}$ is refutable. Such a set $\widehat{\Phi}$ can be obtained as follows. Line up all the propositions $\varphi_0, \varphi_1, \varphi_2, \ldots$ in some order. Set $\Phi_0 \stackrel{\text{def}}{=} \Phi$. For each φ_i, set

$$\Phi_{i+1} \stackrel{\text{def}}{=} \begin{cases} \Phi_i \cup \{\varphi_i\}, & \text{if } \Phi_i \cup \{\varphi_i\} \text{ is consistent,} \\ \Phi_i, & \text{otherwise.} \end{cases}$$

Let $\widehat{\Phi} \stackrel{\text{def}}{=} \bigcup_i \Phi_i$. The set $\widehat{\Phi}$ is consistent, since each Φ_i is consistent and only finitely many formulas can be used in a refutation; and it is maximal, since each φ_i was included unless it was inconsistent with formulas already taken.

We now claim that for each φ, exactly one of the following holds: $\varphi \in \widehat{\Phi}$ or $\varphi \to \mathbf{0} \in \widehat{\Phi}$. Certainly not both are true, because then we would have $\widehat{\Phi} \vdash \mathbf{0}$ by (MP), contradicting the fact that $\widehat{\Phi}$ is consistent. But if neither is true, then $\widehat{\Phi} \cup \{\varphi\}$ and $\widehat{\Phi} \cup \{\varphi \to \mathbf{0}\}$ must both be inconsistent; thus $\widehat{\Phi} \cup \{\varphi\} \vdash \mathbf{0}$ and $\widehat{\Phi} \cup \{\varphi \to \mathbf{0}\} \vdash \mathbf{0}$. By the deduction theorem (Theorem 3.9), $\widehat{\Phi} \vdash \varphi \to \mathbf{0}$ and $\widehat{\Phi} \vdash (\varphi \to \mathbf{0}) \to \mathbf{0}$; therefore by (MP), $\widehat{\Phi} \vdash \mathbf{0}$, a contradiction. It follows that $\widehat{\Phi}$ is *deductively closed* in the sense that if $\widehat{\Phi} \vdash \varphi$, then $\varphi \in \widehat{\Phi}$.

Now we construct a truth assignment satisfying $\widehat{\Phi}$. Set

$$u(\varphi) \stackrel{\text{def}}{=} \begin{cases} \mathbf{1}, & \text{if } \varphi \in \widehat{\Phi}, \\ \mathbf{0}, & \text{if } \varphi \notin \widehat{\Phi}. \end{cases}$$

This certainly satisfies $\widehat{\Phi}$; we only have to show that it is a legal truth assignment. According to the definition, we need to show

$$u(\varphi \to \psi) = \mathbf{1} \iff u(\varphi) = \mathbf{0} \text{ or } u(\psi) = \mathbf{1},$$

or in other words,

$$\varphi \to \psi \in \widehat{\Phi} \iff \varphi \notin \widehat{\Phi} \text{ or } \psi \in \widehat{\Phi}.$$

Suppose first that $\varphi \to \psi \in \widehat{\Phi}$. If $\varphi \notin \widehat{\Phi}$, we are done. Otherwise $\varphi \in \widehat{\Phi}$, in which case $\psi \in \widehat{\Phi}$ by modus ponens.

Conversely, if $\psi \in \widehat{\Phi}$, then $\varphi \to \psi \in \widehat{\Phi}$ by (K) and modus ponens. If $\varphi \notin \widehat{\Phi}$, then $\varphi \to \mathbf{0} \in \widehat{\Phi}$. Then by (EFQ) and transitivity of implication (Examples 3.8 and 3.7 respectively), $\varphi \to \psi \in \widehat{\Phi}$.

Also, $u(\mathbf{0}) = \mathbf{0}$ because $\mathbf{0} \notin \widehat{\Phi}$; otherwise $\widehat{\Phi}$ would be trivially inconsistent. ■

THEOREM 3.11 (COMPLETENESS): If $\Phi \vDash \varphi$ then $\Phi \vdash \varphi$.

Proof

$$\begin{array}{lll} \Phi \vDash \varphi & \Longrightarrow & \Phi \cup \{\varphi \to \mathbf{0}\} \text{ is unsatisfiable} \\ & \Longrightarrow & \Phi \cup \{\varphi \to \mathbf{0}\} \text{ is refutable} \qquad \text{by Lemma 3.10} \\ & \Longrightarrow & \Phi \vdash (\varphi \to \mathbf{0}) \to \mathbf{0} \qquad \text{by the Deduction Theorem} \\ & \Longrightarrow & \Phi \vdash \varphi \qquad \text{by (DN).} \end{array}$$

■

Compactness

Let Φ be a set of propositions, finite or infinite. Recall that a set Φ is *satisfiable* if there is a truth assignment u such that $u \vDash \varphi$ for every $\varphi \in \Phi$. Let us say that Φ is *finitely satisfiable* if every finite subset of Φ is satisfiable.

THEOREM 3.12 (COMPACTNESS OF PROPOSITIONAL LOGIC): Let Φ be any set of propositions. Then Φ is finitely satisfiable iff it is satisfiable.

Proof Trivially, any satisfiable set is finitely satisfiable. The interesting direction is that finite satisfiability implies satisfiability.

Suppose Φ is finitely satisfiable. By the soundness of the system 3.5, every finite subset of Φ is consistent. Since refutations can only use finitely many formulas, the set Φ itself is consistent. By Lemma 3.10, Φ is satisfiable. ■

Compactness has many applications. For example, one can show using compactness that an infinite graph is k-colorable iff every finite subgraph is k-colorable.

The term *compactness* is from topology. Let S be the topological space whose points are the truth assignments $u : \{p, q, r, \ldots\} \to \{\mathbf{1}, \mathbf{0}\}$ and whose basic open sets are the sets $\{\varphi' \mid \varphi \text{ is a proposition}\}$. A family of sets has the *finite intersection property* if every finite subfamily has a nonempty intersection. A topological space

is *compact* if every family Δ of closed sets with the finite intersection property has a nonempty intersection $\bigcap \Delta$. Theorem 3.12 asserts exactly that the topological space S is compact.

An Equational System

Here is another complete deductive system for propositional logic. Whereas Axiom System 3.5 is as austere as possible, this system errs in the opposite direction. No attempt has been made to reduce it to the minimum possible number of costructs. It is thus much richer and more suitable for reasoning.

The system is an equational-style system for deriving theorems of the form $\varphi \leftrightarrow \psi$. Recall that $\varphi \equiv \psi$ iff $\models \varphi \leftrightarrow \psi$. The relation $\equiv$ is an equivalence relation on formulas. Later on in Section 3.3 we will give a general introduction to equational logic. In that section, the system we are about to present would be called *Boolean algebra* (see Exercise 3.8).

Let φ be a proposition containing only the propositional letters $p_1, \ldots, p_n$. Let S be the set of all 2^n truth assignments to $p_1, \ldots, p_n$. As observed in Section 3.2, $\models \varphi$ iff $\varphi' = S$, where

$$\varphi' \quad = \quad \{u \in S \mid u \models \varphi\}.$$

Define a *literal* of $\{p_1, \ldots, p_n\}$ to be a propositional letter p_i or its negation $\neg p_i$. There is a one-to-one correspondence between truth assignments u and conjunctions of literals

$$q_1 \ \wedge \ q_2 \ \wedge \ \cdots \ \wedge \ q_n,$$

where each q_i is either p_i or $\neg p_i$. Such a formula is called an *atom* of $\{p_1, \ldots, p_n\}$. For each truth assignment u there is exactly one atom satisfied by u, and each atom is satisfied by exactly one truth assignment. For example, the atom

$$p_1 \ \wedge \ p_2 \ \wedge \ \neg p_3$$

of $\{p_1, p_2, p_3\}$ corresponds to the truth assignment $u(p_1) = u(p_2) = \mathbf{1}$, $u(p_3) = \mathbf{0}$.

If $\alpha_1, \ldots, \alpha_k$ are atoms, then the disjunction

$$\alpha_1 \ \vee \ \alpha_2 \ \vee \ \cdots \ \vee \ \alpha_k$$

is satisfied by exactly those truth assignments corresponding to the atoms $\alpha_1, \ldots, \alpha_k$. It follows that every subset of S is φ' for some φ. If $\models \varphi$, then $\varphi' = S$, in which case φ is equivalent to the disjunction of all 2^n possible atoms.

Our deductive system is a Hilbert-style system of equational axioms and rules.

It can be used to show that any formula φ is equivalent to another formula of a special form, called complete disjunctive normal form. A formula is in *complete disjunctive normal form* if it is a disjunction of atoms of $\{p_1, \ldots, p_n\}$ in which each atom occurs at most once. The complete disjunctive normal form consisting of no atoms is $\mathbf{0}$.

Axiom System 3.13:

(i) $\mathbf{1}$

(ii) De Morgan laws:

$$\neg(\varphi \wedge \psi) \quad \leftrightarrow \quad \neg\varphi \vee \neg\psi$$
$$\neg(\varphi \vee \psi) \quad \leftrightarrow \quad \neg\varphi \wedge \neg\psi$$

(iii) Law of double negation:

$$\neg\neg\varphi \quad \leftrightarrow \quad \varphi$$

(iv) Associative laws:

$$(\varphi \wedge \psi) \wedge \rho \quad \leftrightarrow \quad \varphi \wedge (\psi \wedge \rho)$$
$$(\varphi \vee \psi) \vee \rho \quad \leftrightarrow \quad \varphi \vee (\psi \vee \rho)$$

(v) Commutative laws:

$$\varphi \vee \psi \quad \leftrightarrow \quad \psi \vee \varphi$$
$$\varphi \wedge \psi \quad \leftrightarrow \quad \psi \wedge \varphi$$

(vi) Distributive laws:

$$\varphi \vee (\psi \wedge \rho) \quad \leftrightarrow \quad (\varphi \vee \psi) \wedge (\varphi \vee \rho)$$
$$\varphi \wedge (\psi \vee \rho) \quad \leftrightarrow \quad (\varphi \wedge \psi) \vee (\varphi \wedge \rho)$$

(vii) Idempotency laws:

$$\varphi \vee \varphi \quad \leftrightarrow \quad \varphi$$
$$\varphi \wedge \varphi \quad \leftrightarrow \quad \varphi$$

(viii) Zero-one laws:

$$\begin{aligned}
\varphi \wedge \mathbf{0} &\leftrightarrow \mathbf{0} \\
\varphi \vee \mathbf{0} &\leftrightarrow \varphi \\
\varphi \vee \neg\varphi &\leftrightarrow \mathbf{1} \\
\varphi \wedge \mathbf{1} &\leftrightarrow \varphi \\
\varphi \vee \mathbf{1} &\leftrightarrow \mathbf{1} \\
\varphi \wedge \neg\varphi &\leftrightarrow \mathbf{0}
\end{aligned}$$

(ix) Elimination of implications:

$$\begin{aligned}
(\varphi \rightarrow \psi) &\leftrightarrow \neg\varphi \vee \psi \\
(\varphi \leftrightarrow \psi) &\leftrightarrow (\varphi \wedge \psi) \vee (\neg\varphi \wedge \neg\psi)
\end{aligned}$$

(x) Substitution of equals for equals:

$$\frac{\varphi \leftrightarrow \psi, \quad \sigma[p/\varphi]}{\sigma[p/\psi]}$$

where $\sigma[p/\varphi]$ denotes a formula σ with the proposition φ substituted for all occurrences of the atomic proposition p.

The substitution rule (x) says that if we have established the equivalence of two expressions φ and ψ, and we have proved a theorem containing φ as a subexpression, then we may substitute ψ for φ in that theorem and the resulting formula will be a theorem.

THEOREM 3.14: Axiom System 3.13 is sound.

Proof We need to show that every theorem derivable in the system is valid. Using induction on the lengths of proofs, it suffices to show that all the axioms are valid and the rule of inference preserves validity.

The validity of the axioms can be established by reasoning set theoretically with φ'. For example, for (v), we need to show

$$\models \varphi \vee \psi \leftrightarrow \psi \vee \varphi.$$

But this is true since

$$(\varphi \vee \psi)' = (\psi \vee \varphi)' = \varphi' \cup \psi'$$

and $\cup$ is commutative. Axiom (i) is valid since $\mathbf{1}' = S$. We leave the verification of the remaining axioms as exercises (Exercise 3.6).

In order to establish the soundness of the substitution rule (x), we observe first that if $\varphi \leftrightarrow \psi$ is a tautology then so is $\sigma[p/\varphi] \leftrightarrow \sigma[p/\psi]$. In other words, if $\varphi \equiv \psi$, then $\sigma[p/\varphi] \equiv \sigma[p/\psi]$. This can be proved by induction on the depth of σ. If $\sigma = p$, then the substitution gives $\varphi \equiv \psi$, which is true by assumption. If $\sigma = q \neq p$, then the substitution gives $q \equiv q$.

For the induction step, we need only observe that if $\sigma_1 \equiv \sigma_2$ and $\tau_1 \equiv \tau_2$, then $\sigma_1 \to \tau_1 \equiv \sigma_2 \to \tau_2$, and similarly for the other operators.

Now if $\vDash \varphi \leftrightarrow \psi$, then $\vDash \sigma[p/\varphi] \leftrightarrow \sigma[p/\psi]$, as we have just shown. If in addition $\vDash \sigma[p/\varphi]$, then $\vDash \sigma[p/\psi]$. Thus if both premises of the inference rule (x) are valid, then so is the conclusion. Since every axiom is valid and the rule of inference preserves validity, any theorem derivable in this system is valid. ■

THEOREM 3.15: Axiom System 3.13 is complete.

Proof sketch. Intuitively, our axioms and inference rule allow us to transform any proposition φ into an equivalent join of atoms of $p_1, \ldots, p_n$, the propositional letters occurring in φ. If φ is valid, then this join of atoms must contain all 2^n possible atoms, in which case the formula can be further transformed to $\mathbf{1}$.

In the following, we use axioms (iv) and (v) implicitly to rearrange parentheses and formulas in conjunctions and disjunctions. Thus we may write $\varphi_1 \vee \varphi_2 \vee \cdots \vee \varphi_k$ without parentheses and without regard to the order of the φ_i.

Starting with φ, first apply the axioms (ix) in the left-to-right direction to replace any subexpression of the form $\varphi \to \psi$ with $\neg\varphi \vee \psi$ and any subexpression of the form $\varphi \leftrightarrow \psi$ with $(\varphi \wedge \psi) \vee (\neg\varphi \wedge \neg\psi)$ until the resulting term has no occurrence of $\to$ or $\leftrightarrow$. Now apply the De Morgan laws (ii) in the left-to-right direction to move all occurrences of $\neg$ inward. Whenever a subexpression of the form $\neg\neg\varphi$ occurs, replace it by φ using (iii). Keep doing this until all occurrences of $\neg$ are applied only to atomic p. Use the second distributive law (vi) in the left-to-right direction to move occurrences of $\vee$ outward and occurrences of $\wedge$ inward until obtaining *disjunctive normal form*: a disjunction of conjunctions of literals p or $\neg p$.

If some p occurs twice in one of the conjunctions, use (vii) to get rid of the double occurrence. If p and $\neg p$ both occur, use (viii) in the left-to-right direction. If neither p nor $\neg p$ occurs in a conjunction ψ, use (viii) and then (vi) to replace ψ with $(\psi \wedge p) \vee (\psi \wedge \neg p)$. The result is a disjunction of atoms of $p_1, \ldots, p_n$. Since φ was valid and the transformations preserve validity (Theorem 3.14), the resulting formula is valid; thus all 2^n atoms must appear in the disjunction, since otherwise

the truth assignment corresponding to an omitted atom would falsify the formula.

Now we use the rules in the opposite direction to reduce the formula to **1**. Note that each atom of $p_1, \ldots, p_n$ is of the form $\psi \wedge p_n$ or $\psi \wedge \neg p_n$, where ψ is an atom of $p_1, \ldots, p_{n-1}$. Moreover, since all 2^n atoms of $p_1, \ldots, p_n$ occur in the disjunction, both $\psi \wedge p_n$ and $\psi \wedge \neg p_n$ occur for each of the 2^{n-1} atoms ψ of $p_1, \ldots, p_{n-1}$. For each such ψ, apply the second distributive law (vi) in the right-to-left direction to $(\psi \wedge p_n) \vee (\psi \wedge \neg p_n)$ to obtain $\psi \wedge (p \vee \neg p)$, then (viii) to obtain $\psi \wedge \mathbf{1}$, then (viii) again to obtain ψ. We are left with the disjunction of all 2^{n-1} atoms of $p_1, \ldots, p_{n-1}$. Continue in this fashion until we are left with **1**. ■

3.3 Equational Logic

Equational logic is a formalization of equational reasoning. The properties of pure equality are captured in the axioms for equivalence relations: *reflexivity*, *symmetry*, and *transitivity* (see Section 1.3). In the presence of function symbols, a fourth rule of *congruence* is added.

The language of equational logic is a sublanguage of first order logic, so it makes sense to treat this special case before moving on to full first-order logic in Section 3.4.

Syntax

A *signature* or *vocabulary* consists of a set Σ of *function symbols*, each with an associated *arity* (number of input places). There is only one relation symbol, the *equality symbol* $=$, and it is of arity 2.[1] Function symbols of arity 0, 1, 2, 3, and n are called *nullary, unary, binary, ternary*, and *n-ary*, respectively. Nullary symbols are often called *constants*.

EXAMPLE 3.16: The signature for groups consists of function symbols $\cdot$, $^{-1}$, and 1, where $\cdot$ is a binary symbol for multiplication, $^{-1}$ is a unary symbol for multiplicative inverse, and 1 is a nullary (constant) symbol for the multiplicative identity.

EXAMPLE 3.17: The signature of Boolean algebra consists of the function symbols $\wedge, \vee, \neg, \mathbf{0}, \mathbf{1}$ with arities $2, 2, 1, 0, 0$ respectively.

1 When we discuss first-order logic in Section 3.4, the signature may also include other relation symbols of various arities.

We will work with an arbitrary but fixed signature Σ of function symbols. We use the symbols $f, g, h, \ldots$ to denote typical elements of Σ and $a, b, c, \ldots$ to denote typical constants in Σ. So that we do not have to keep writing "...where f is n-ary," we will adopt the convention that any use of the expression $f(t_1, \ldots, t_n)$ carries with it the implicit proviso that the symbol f is of arity n.

The language of equational logic is built from the symbols of Σ, the binary equality symbol $=$, a countable set X of *individual variables* $x, y, \ldots$, and parentheses.

A *term* is a well formed expression built from the function symbols and variables. By "well formed," we mean that it respects the arities of all the symbols, where variables are considered to have arity 0. Terms are denoted $s, t, \ldots$. Formally, terms are defined inductively:

- any variable x is a term;
- if $t_1, \ldots, t_n$ are terms and f is n-ary, then $f(t_1, \ldots, t_n)$ is a term.

Note that every constant symbol c is a term: this is the case $n = 0$ in the second clause above. The following is a typical term, where f is binary, g is unary, c is a constant, and x, y are variables:

$$f(f(x, g(c)), g(f(y, c))).$$

The set of all terms over Σ and X is denoted $T_\Sigma(X)$. A term is called a *ground term* if it contains no variables. The set of ground terms over Σ is denoted T_Σ. We sometimes write $t(x_1, \ldots, x_n)$ to indicate that all variables occurring in t are among $x_1, \ldots, x_n$. (It is not necessary that all of $x_1, \ldots, x_n$ appear in t.)

EXAMPLE 3.18: Terms over the signature of Boolean algebra described in Example 3.17 are exactly the propositional formulas over $\wedge$, $\vee$, $\neg$, $\mathbf{0}$, and $\mathbf{1}$ as described in Section 3.2. The propositional letters are the variables.

Over an abstract signature Σ, we always write terms in *prefix notation*, which means function symbol first: $f(t_1, \ldots, t_n)$. In various applications, however, we often use infix or postfix notation for certain operators as dictated by custom. For example, the binary Boolean operators $\vee$ and $\wedge$ are written in infix: $s \vee t$, $s \wedge t$. The unary Boolean operator $\neg$ is customarily written in prefix as $\neg t$, whereas the unary group inverse operator $^{-1}$ is customarily written in postfix as t^{-1}.

An *equation* is a formal expression $s = t$ consisting of two terms separated by the equality symbol $=$. A *Horn formula* is a formal expression of the form

$$s_1 = t_1 \wedge \cdots \wedge s_k = t_k \quad \rightarrow \quad s = t, \tag{3.3.1}$$

where the $s_i = t_i$ and $s = t$ are equations. When $k = 0$, the Horn formula (3.3.1) is equivalent to the equation $s = t$.

Semantics

Σ-algebras

Terms and equations over an alphabet Σ take on meaning when interpreted over an algebraic structure called a Σ-*algebra* (or just an *algebra* when Σ is understood). This is a structure

$$\mathfrak{A} \quad = \quad (A,\ \mathfrak{m}_{\mathfrak{A}})$$

consisting of

- a nonempty set A called the *carrier* or *domain* of $\mathfrak{A}$, the elements of which are called *individuals*;
- a *meaning function* $\mathfrak{m}_{\mathfrak{A}}$ that assigns an n-ary function $\mathfrak{m}_{\mathfrak{A}}(f) : A^n \rightarrow A$ to each n-ary function symbol $f \in \Sigma$. We abbreviate $\mathfrak{m}_{\mathfrak{A}}(f)$ by $f^{\mathfrak{A}}$.

We regard 0-ary functions $A^0 \rightarrow A$ as just elements of A. Thus the constant symbol $c \in \Sigma$ is interpreted as an element $c^{\mathfrak{A}} \in A$. The carrier of $\mathfrak{A}$ is sometimes denoted $|\mathfrak{A}|$.

When we discuss the syntax of first-order logic in Section 3.4, the signature will include relation symbols p of various arities, and the meaning function $\mathfrak{m}_{\mathfrak{A}}$ will assign relations $\mathfrak{m}_{\mathfrak{A}}(p) = p^{\mathfrak{A}}$ on A to those symbols as well. In equational logic, however, there is only one (binary) relation symbol $=$, and unless we say otherwise, its interpretation $=^{\mathfrak{A}}$ in $\mathfrak{A}$ is always assumed to be the binary *identity relation*

$$\{(a, a) \mid a \in A\}.$$

As is customary, we omit the superscript $\mathfrak{A}$ from $=^{\mathfrak{A}}$ and denote by $=$ both the equality symbol itself and its meaning as the identity relation on A. To complicate matters further, we also use $=$ as our metasymbol for equality. The proper interpretation should be clear from context.

EXAMPLE 3.19: The signature of Boolean algebra was described in Example 3.17. Let K be a set, and let 2^K denote the powerset of K. We define an algebra $\mathfrak{B}$ for this

signature consisting of carrier 2^K, $\vee^{\mathfrak{B}}$ the operation of set union, $\wedge^{\mathfrak{B}}$ the operation of set intersection, $\neg^{\mathfrak{B}}$ the operation of set complementation in K, $\mathbf{0}^{\mathfrak{B}}$ the empty set, and $\mathbf{1}^{\mathfrak{B}}$ the set K. The algebra of sets of truth assignments described in Section 3.2 is an example of such a Boolean algebra.

EXAMPLE 3.20: For any signature Σ, the set of terms $T_\Sigma(X)$ forms a Σ-algebra, where each $f \in \Sigma$ is given the *syntactic interpretation*

$$f^{T_\Sigma(X)}(t_1, \ldots, t_n) \quad = \quad f(t_1, \ldots, t_n).$$

There is no meaning associated with the f appearing on the right-hand side; it is merely a symbol, and the expression $f(t_1, \ldots, t_n)$ is merely a term—a syntactic object. The $f^{T_\Sigma(X)}$ on the left-hand side, however, is a semantic object; it is the function $f^{T_\Sigma(X)} : T_\Sigma(X)^n \to T_\Sigma(X)$ which on input $t_1, \ldots, t_n$ gives the term $f(t_1, \ldots, t_n)$ as result.

The algebra $T_\Sigma(X)$ is called a *term algebra*.

Subalgebras and Generating Sets

Let $\mathfrak{A}$ and $\mathfrak{B}$ be two Σ-algebras with carriers A and B, respectively. The algebra $\mathfrak{A}$ is a *subalgebra* of $\mathfrak{B}$ if $A \subseteq B$ and $f^{\mathfrak{A}} = f^{\mathfrak{B}} \restriction A^n$ for all n-ary $f \in \Sigma$.

If $C \subseteq B$, the *subalgebra of* $\mathfrak{B}$ *generated by* C is the smallest subalgebra of $\mathfrak{B}$ containing C. Its domain is the smallest subset of B containing C and the constants $c^{\mathfrak{B}}$ and closed under the action of the functions $f^{\mathfrak{B}}$, as described in Section 1.7.

The set C is called a *generating set* of $\mathfrak{B}$, and is said to *generate* $\mathfrak{B}$, if the subalgebra of $\mathfrak{B}$ generated by C is $\mathfrak{B}$ itself. For example, X generates the term algebra $T_\Sigma(X)$.

Homomorphisms

Homomorphisms are structure-preserving functions between Σ-algebras. Formally, if $\mathfrak{A} = (A, \mathfrak{m}_{\mathfrak{A}})$ and $\mathfrak{B} = (B, \mathfrak{m}_{\mathfrak{B}})$ are Σ-algebras, a *homomorphism* $h : \mathfrak{A} \to \mathfrak{B}$ is a function $h : A \to B$ that commutes with the distinguished functions in the sense that for any $a_1, \ldots, a_n \in A$ and $f \in \Sigma$,

$$h(f^{\mathfrak{A}}(a_1, \ldots, a_n)) \quad = \quad f^{\mathfrak{B}}(h(a_1), \ldots, h(a_n)). \tag{3.3.2}$$

This includes the case $n = 0$ (constants), for which (3.3.2) reduces to

$$h(c^{\mathfrak{A}}) \quad = \quad c^{\mathfrak{B}}.$$

A homomorphism is called a *monomorphism* if it is one-to-one, an *epimorphism* if it is onto, and an *isomorphism* if it is both. An algebra $\mathfrak{B}$ is a *homomorphic image* of $\mathfrak{A}$ if there is an epimorphism $h : \mathfrak{A} \to \mathfrak{B}$.

The identity map $\mathfrak{A} \to \mathfrak{A}$ is an isomorphism, and if $g : \mathfrak{A} \to \mathfrak{B}$ and $h : \mathfrak{B} \to \mathfrak{C}$ are homomorphisms, then so is $g \circ h : \mathfrak{A} \to \mathfrak{C}$. Because of (3.3.2), any homomorphism is uniquely determined by its action on a generating set.

The *kernel* of a homomorphism $h : \mathfrak{A} \to \mathfrak{B}$ is the relation

$$\ker h \stackrel{\text{def}}{=} \{(a, b) \mid h(a) = h(b)\} \tag{3.3.3}$$

on $\mathfrak{A}$.

Valuations and Substitutions

A homomorphism $u : T_\Sigma(X) \to \mathfrak{A}$ defined on a term algebra over variables X is called a *valuation*. A valuation is uniquely determined by its values on X, since X generates $T_\Sigma(X)$. Moreover, any map $u : X \to \mathfrak{A}$ extends uniquely to a valuation $u : T_\Sigma(X) \to \mathfrak{A}$ by induction using (3.3.2).

Note that in the case $n = 0$, we have $u(c) = c^{\mathfrak{A}}$. Note also that the value of $u(t)$ depends only on the values $u(x)$ for those x appearing in t. In particular, a ground term t always has a fixed value in $\mathfrak{A}$, independent of the valuation u. Thus in the case of ground terms t, we can write $t^{\mathfrak{A}}$ for $u(t)$.

A homomorphism $u : T_\Sigma(X) \to T_\Sigma(Y)$ from one term algebra to another is called a *substitution*. We can think of these maps as substitutions in the usual sense: the value of the substitution u applied to a term $t \in T_\Sigma(X)$ is the term in $T_\Sigma(Y)$ obtained by substituting the term $u(x)$ for all occurrences of x in t simultaneously for all $x \in X$:

$$u(t) \quad = \quad t[x/u(x) \mid x \in X].$$

If $u : T_\Sigma(X) \to T_\Sigma(Y)$ is a substitution and $s, t \in T_\Sigma(X)$, then the term $u(s)$ and the equation $u(s) = u(t)$ are called *substitution instances* over $T_\Sigma(Y)$ of s and $s = t$, respectively.

Satisfaction

We say that the Σ-algebra $\mathfrak{A}$ *satisfies* the equation $s = t$ under valuation $u : T_\Sigma(X) \to \mathfrak{A}$, and write $\mathfrak{A}, u \vDash s = t$, if $u(s) = u(t)$; that is, if $u(s)$ and $u(t)$ are the same element of A. More generally, we say that the Σ-algebra $\mathfrak{A}$ *satisfies* the Horn

formula

$$s_1 = t_1 \wedge \cdots \wedge s_k = t_k \quad \rightarrow \quad s = t$$

under valuation $u : T_\Sigma(X) \to \mathfrak{A}$, and write

$$\mathfrak{A}, u \quad \vDash \quad s_1 = t_1 \wedge \cdots \wedge s_k = t_k \rightarrow s = t,$$

if either

- $u(s_i) \neq u(t_i)$ for some i, $1 \leq i \leq k$; or
- $u(s) = u(t)$.

If φ is an equation or a Horn formula, we write $\mathfrak{A} \vDash \varphi$ if $\mathfrak{A}, u \vDash \varphi$ for all valuations u and say that φ is *valid* in $\mathfrak{A}$, or that $\mathfrak{A}$ *satisfies* φ, or that $\mathfrak{A}$ is a *model* of φ.[2] If Φ is a set of equations or Horn formulas, we write $\mathfrak{A} \vDash \Phi$ if $\mathfrak{A}$ satisfies all elements of Φ and say that $\mathfrak{A}$ *satisfies* Φ, or that $\mathfrak{A}$ is a *model* of Φ. We denote by $\mathbf{Mod}\,\Phi$ the class of all models of Φ.

Let $\mathbf{Th}\,\mathfrak{A}$ denote the set of equations over X that are valid in $\mathfrak{A}$:

$$\mathbf{Th}\,\mathfrak{A} \quad \stackrel{\text{def}}{=} \quad \{s = t \mid \mathfrak{A} \vDash s = t\}.$$

If $\mathcal{D}$ is a class of Σ-algebras, let $\mathbf{Th}\,\mathcal{D}$ denote the set of equations valid in all elements of $\mathcal{D}$:

$$\mathbf{Th}\,\mathcal{D} \quad \stackrel{\text{def}}{=} \quad \bigcap_{\mathfrak{A} \in \mathcal{D}} \mathbf{Th}\,\mathfrak{A}.$$

The set $\mathbf{Th}\,\mathcal{D}$ is called the *equational theory* of $\mathcal{D}$. An equation $s = t$ is called a *logical consequence* of Φ if $s = t$ is satisfied by all models of Φ; that is, if $s = t \in \mathbf{Th}\,\mathbf{Mod}\,\Phi$.

Example 3.21: Any group satisfies the following equations:

$$\begin{aligned} x \cdot (y \cdot z) &= (x \cdot y) \cdot z \\ x^{-1} \cdot x &= 1 \\ x \cdot x^{-1} &= 1 \\ x \cdot 1 &= x \\ 1 \cdot x &= x. \end{aligned} \tag{3.3.4}$$

2 Thus we think of the variables in φ as *universally quantified*. This terminology will make sense after we have introduced the universal quantifier $\forall$ in Section 3.4.

In fact, a group is *defined* to be any algebra over the signature of groups that satisfies these five equations. A group is *Abelian* or *commutative* if it satisfies the extra equation $x \cdot y = y \cdot x$.

The equational theory of groups is the set of equations that are true in all groups. This is the set of logical consequences of the axioms (3.3.4).

Varieties

As noted in Example 3.21, the class of groups and the class of Abelian groups are defined by sets of equations. Such classes are called *equationally defined classes* or *varieties*.

Formally, a class $\mathcal{C}$ of Σ-algebras is an *equationally defined class* or *variety* if there is a set of equations Φ over $T_\Sigma(X)$ such that $\mathcal{C} = \mathbf{Mod}\,\Phi$.

The following are examples of varieties:

EXAMPLE 3.22: A semigroup is any structure with an associative binary operation $\cdot$; i.e., it is an algebraic structure over an alphabet consisting of a single binary operator $\cdot$ and satisfying the equation $x \cdot (y \cdot z) = (x \cdot y) \cdot z$.

EXAMPLE 3.23: Monoids are semigroups with a left and right identity element 1. In other words, the nullary symbol 1 is added to the signature along with equations $1 \cdot x = x$ and $x \cdot 1 = x$.

EXAMPLE 3.24: Rings are algebraic structures over the alphabet $+, \cdot, -, 0, 1$ of arity $2, 2, 1, 0, 0$, respectively, defined by equations that say that the structure under the operations $+, -, 0$ forms an Abelian group, that the structure under the operations $\cdot, 1$ forms a monoid, and that the following *distributive laws* describing the interaction of the additive and multiplicative structure hold:

$$\begin{aligned} x \cdot (y + z) &= (x \cdot y) + (x \cdot z) \\ (x + y) \cdot z &= (x \cdot z) + (y \cdot z). \end{aligned}$$

A ring is *commutative* if it satisfies the extra equation $x \cdot y = y \cdot x$.

EXAMPLE 3.25: *Semilattices* and *lattices* as described in Section 1.5 are varieties. Complete lattices are not (Exercise 3.34). The signature for lattices is $\vee, \wedge, \bot, \top$ (join, meet, bottom, top) of arity $2, 2, 0, 0$, respectively. The signature for semilattices is $\vee, \bot$ only. No special symbol for $\leq$ is necessary, because $x \leq y$ can be considered an abbreviation for $x \vee y = y$. This is true for semilattices, but not for

partial orders in general (Exercise 3.24).

EXAMPLE 3.26: A *Boolean algebra* is an algebra over the signature described in Example 3.17 satisfying (ii)–(viii) of Axiom System 3.13, regarding $\leftrightarrow$ as equality (see Exercise 3.8). Boolean algebra is the algebraic analog of propositional logic.

EXAMPLE 3.27: A *vector space* over the real numbers $\mathbb{R}$ is an Abelian group with a unary operator for each real number a denoting the operation of scalar multiplication by a, and satisfying the following infinite set of equations:

$$\begin{aligned} a(x+y) &= ax+ay \\ (a+b)x &= ax+bx \\ (ab)x &= a(bx) \end{aligned}$$

for all $a, b \in \mathbb{R}$. We can regard this as a Σ-algebra over an infinite Σ containing the signature $+, -, 0$ of Abelian groups as well as infinitely many unary symbols, one for each $a \in \mathbb{R}$.

THEOREM 3.28: Any variety $\mathcal{C}$ is closed under homomorphic images. In other words, if $h : \mathfrak{A} \to \mathfrak{B}$ is an epimorphism and $\mathfrak{A} \in \mathcal{C}$, then $\mathfrak{B} \in \mathcal{C}$.

Proof It suffices to show that if $h : \mathfrak{A} \to \mathfrak{B}$ is an epimorphism and $\mathfrak{A} \vDash s = t$, then $\mathfrak{B} \vDash s = t$.

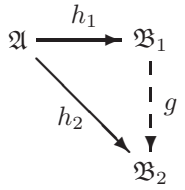
Let X be a set of variables containing all the variables occurring in s and t. Let $v : T_\Sigma(X) \to \mathfrak{B}$ be an arbitrary valuation. Define the function $u : X \to \mathfrak{A}$ such that $h(u(x)) = v(x)$. This is always possible, since h is onto. The function u extends uniquely to a homomorphism $u : T_\Sigma(X) \to \mathfrak{A}$. Since homomorphisms are determined by their values on a generating set, and since the valuations $u \circ h$ and v agree on the generating set X, they are equal.

Now since $\mathfrak{A} \vDash s = t$, we have that $u(s) = u(t)$. Then

$$v(s) = h(u(s)) = h(u(t)) = v(t).$$

Since v was arbitrary, $\mathfrak{B} \vDash s = t$. ■

LEMMA 3.29: Let $h_1 : \mathfrak{A} \to \mathfrak{B}_1$ and $h_2 : \mathfrak{A} \to \mathfrak{B}_2$ be homomorphisms defined on $\mathfrak{A}$ such that h_1 is an epimorphism and $\ker h_1$ refines $\ker h_2$; that is, $\ker h_1 \subseteq \ker h_2$. Then there exists a unique homomorphism $g : \mathfrak{B}_1 \to \mathfrak{B}_2$ such that $h_2 = h_1 \circ g$.



h_1 an epimorphism

ker $h_1 \subseteq$ ker h_2

Proof Since h_1 is an epimorphism, any element of $\mathfrak{B}_1$ is of the form $h_1(a)$ for some $a \in \mathfrak{A}$. To satisfy the lemma, we had better define $g(h_1(a)) = h_2(a)$. This determines the function g uniquely, provided it is well defined. But it is well defined, for if a' is any other element of $\mathfrak{A}$ such that $h_1(a') = h_1(a)$, then $h_2(a') = h_2(a)$. Moreover, g is a homomorphism: if $b_i = h_1(a_i)$, $1 \leq i \leq n$, then

$$\begin{aligned} g(f^{\mathfrak{B}_1}(b_1, \ldots, b_n)) &= g(f^{\mathfrak{B}_1}(h_1(a_1), \ldots, h_1(a_n))) \\ &= g(h_1(f^{\mathfrak{A}}(a_1, \ldots, a_n))) \\ &= h_2(f^{\mathfrak{A}}(a_1, \ldots, a_n)) \\ &= f^{\mathfrak{B}_2}(h_2(a_1), \ldots, h_2(a_n)) \\ &= f^{\mathfrak{B}_2}(g(h_1(a_1)), \ldots, g(h_1(a_n))) \\ &= f^{\mathfrak{B}_2}(g(b_1), \ldots, g(b_n)). \end{aligned}$$

■

Congruences

A *congruence* $\equiv$ on a Σ-algebra $\mathfrak{A}$ with carrier A is an equivalence relation on A that respects the functions $f^{\mathfrak{A}}$, $f \in \Sigma$, in the sense that

$$a_i \equiv b_i,\ 1 \leq i \leq n \implies f^{\mathfrak{A}}(a_1, \ldots, a_n) \equiv f^{\mathfrak{A}}(b_1, \ldots, b_n). \tag{3.3.5}$$

The $\equiv$-*congruence class* of an element a is the set

$$[a] \stackrel{\text{def}}{=} \{b \mid b \equiv a\}.$$

Since any congruence is an equivalence relation, the congruences classes partition A in the sense that they are pairwise disjoint and their union is A (Section 1.3).

EXAMPLE 3.30: The identity relation and the universal relation are always congruences. They are the finest and coarsest congruences, respectively, on any algebra. The congruence classes of the identity relation are all singleton sets, and the universal relation has one congruence class consisting of all elements.

EXAMPLE 3.31: On the ring of integers $\mathbb{Z}$, any positive integer n defines a congruence

$$a \equiv_n b \iff b - a \text{ is divisible by } n.$$

In the number theory literature, this relation is commonly written $a \equiv b\ (n)$ or $a \equiv b \pmod{n}$. There are n congruence classes, namely $[0], [1], \ldots, [n-1]$. The congruence class $[0]$ is the set of all multiples of n.

EXAMPLE 3.32: An *ideal* of a commutative ring $\mathfrak{R}$ is a nonempty subset I such that if $a, b \in I$ then $a + b \in I$, and if $a \in I$ and $b \in \mathfrak{R}$, then $ab \in I$. If $\mathfrak{R}$ is any commutative ring and I is an ideal in $\mathfrak{R}$, then the relation

$$a \equiv_I b \iff b - a \in I$$

is a congruence. This relation is often written $a \equiv b\ (I)$. Conversely, given any congruence $\equiv$, the congruence class $[0]$ is an ideal. Example 3.31 is a special case.

EXAMPLE 3.33: A subgroup $\mathfrak{H}$ of a group $\mathfrak{G}$ is a *normal* (or *self-conjugate*) subgroup of $\mathfrak{G}$, in symbols $\mathfrak{H} \triangleleft \mathfrak{G}$, if for all $x \in \mathfrak{H}$ and $a \in \mathfrak{G}$, $a^{-1}xa \in \mathfrak{H}$. If $\mathfrak{G}$ is any group and $\mathfrak{H} \triangleleft \mathfrak{G}$, then the relation

$$a \equiv_{\mathfrak{H}} b \iff b^{-1}a \in \mathfrak{H}$$

is a congruence. Conversely, given any congruence on $\mathfrak{G}$, the congruence class of the identity element is a normal subgroup.

Let $\mathfrak{A}$ be a Σ-algebra with carrier A and let S be any binary relation on A. By considerations of Section 1.7, there is a unique minimal congruence on $\mathfrak{A}$ containing S, called the *congruence closure* of S. It is the least relation that contains S and is closed under the monotone set operators

$$R \mapsto \iota \tag{3.3.6}$$

$$R \mapsto R^- \tag{3.3.7}$$

$$R \mapsto R \circ R \tag{3.3.8}$$

$$R \mapsto \{(f^{\mathfrak{A}}(a_1, \ldots, a_n), f^{\mathfrak{A}}(b_1, \ldots, b_n)) \mid f \in \Sigma \text{ and } (a_i, b_i) \in R,\ 1 \le i \le n\} \tag{3.3.9}$$

corresponding to reflexivity, symmetry, transitivity, and congruence, respectively.

It follows from the results of Section 1.7 that the set of congruences on $\mathfrak{A}$ under the partial order of refinement forms a complete lattice. The meet of a set of congruences $\equiv_i$ is their intersection $\bigcap_i \equiv_i$, and the join of the $\equiv_i$ is the congruence

generated by $\bigcup_i \equiv_i$; that is, the smallest congruence containing all the $\equiv_i$.

The Quotient Construction

The following theorem shows that there is a strong relationship between homomorphisms and congruences. Its proof illustrates an important construction called the *quotient construction*.

THEOREM 3.34:

(i) The kernel of any homomorphism is a congruence.

(ii) Any congruence is the kernel of a homomorphism.

Proof Statement (i) follows in a straightforward way from the definition of homomorphism and congruence and is left as an exercise (Exercise 3.17).

To show (ii), we need to construct, given a congruence $\equiv$ on $\mathfrak{A}$, a Σ-algebra $\mathfrak{B}$ and a homomorphism $h : \mathfrak{A} \to \mathfrak{B}$ with kernel $\equiv$. It will turn out that the homomorphism h we construct is an epimorphism; thus $\mathfrak{B}$ is a homomorphic image of $\mathfrak{A}$. Moreover, up to isomorphism, $\mathfrak{B}$ is the unique homomorphic image of $\mathfrak{A}$ under a homomorphism with kernel $\equiv$. This construction is known as the *quotient construction*.

Let A be the carrier of $\mathfrak{A}$. For $a \in A$, let $[a]$ denote the $\equiv$-congruence class of a, and define

$$A/\equiv \stackrel{\text{def}}{=} \{[a] \mid a \in A\}.$$

Define the Σ-algebra

$$\mathfrak{A}/\equiv \stackrel{\text{def}}{=} (A/\equiv,\ \mathfrak{m}_{\mathfrak{A}/\equiv})$$

where

$$f^{\mathfrak{A}/\equiv}([a_1], \ldots, [a_n]) \stackrel{\text{def}}{=} [f^{\mathfrak{A}}(a_1, \ldots, a_n)]. \tag{3.3.10}$$

We must argue that the function $f^{\mathfrak{A}/\equiv}$ is well defined; that is, if $[a_i] = [b_i]$ for $1 \le i \le n$, then $[f^{\mathfrak{A}}(a_1, \ldots, a_n)] = [f^{\mathfrak{A}}(b_1, \ldots, b_n)]$. But this is precisely (3.3.5).

The Σ-algebra $\mathfrak{A}/\equiv$ is called the *quotient of* $\mathfrak{A}$ *by* $\equiv$ or $\mathfrak{A}$ *modulo* $\equiv$. Moreover, by (3.3.10), the map $a \mapsto [a]$ is a homomorphism $\mathfrak{A} \to \mathfrak{A}/\equiv$, which we call the *canonical homomorphism*. Since $[a] = [b]$ iff $a \equiv b$, the kernel of the canonical homomorphism is $\equiv$. ■

In Example 3.31, the quotient $\mathbb{Z}/\equiv_n$ is the ring of integers modulo n. In Example 3.32, we defined a congruence $\equiv_I$ on a commutative ring in terms of an ideal I. Similarly, in Example 3.33, we defined a congruence $\equiv_{\mathfrak{H}}$ on a group $\mathfrak{G}$ in terms of a normal subgroup $\mathfrak{H}$. By Theorem 3.34, these congruences are kernels of homomorphisms. In ring theory and group theory, the *kernel* is normally defined to be the ideal or normal subgroup itself, not the congruence it generates. However, there is a one-to-one correspondence between congruences and ideals of a commutative ring and between congruences and normal subgroups of a group, so the definition of kernel as given in (3.3.3) subsumes these as special cases.

The relationship between congruences and homomorphisms is even stronger than Theorem 3.34 would suggest. As mentioned in Section 3.3, the set of congruences on $\mathfrak{A}$ under the partial order of refinement forms a complete lattice. Similarly, consider the class of all epimorphisms with domain $\mathfrak{A}$. For two such epimorphisms $h_1 : \mathfrak{A} \to \mathfrak{B}_1$ and $h_2 : \mathfrak{A} \to \mathfrak{B}_2$, let us write $h_1 \leq h_2$ if there exists an epimorphism $g : \mathfrak{B}_1 \to \mathfrak{B}_2$ such that $h_2 = h_1 \circ g$ and $h_1 \approx h_2$ if both $h_1 \leq h_2$ and $h_2 \leq h_1$. Using Lemma 3.29, one can show that $h_1 \approx h_2$ iff there is an isomorphism $\iota : \mathfrak{B}_1 \to \mathfrak{B}_2$ such that $h_2 = h_1 \circ \iota$ (Exercise 3.19). The set of $\approx$-classes of epimorphisms on $\mathfrak{A}$ forms a complete lattice under the partial order $\leq$.

THEOREM 3.35: Up to $\approx$, the congruences on $\mathfrak{A}$ and the epimorphisms on $\mathfrak{A}$ are in one-to-one correspondence under the map that associates an epimorphism with its kernel. This correspondence is an isomorphism of lattices.

Proof Exercise 3.20. ■

Free Algebras

One recurring phenomenon in algebra that turns out to be very useful is the notion of a *free algebra*. The essential idea is that for any set Φ of equations and for any set Y, there is an algebra generated by Y that satisfies Φ and all of its logical consequences, but no more. Thus it is as "free" of extra equations as possible, satisfying only those equations it is forced to satisfy by Φ. The free algebra is unique up to isomorphism.

The free algebra on generators Y satisfying Φ can be constructed as the quotient of $T_\Sigma(Y)$ modulo the smallest congruence containing all substitution instances over $T_\Sigma(Y)$ of equations in Φ. We denote this quotient by $T_\Sigma(Y)/\Phi$.

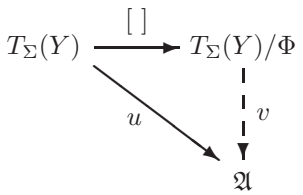
In more detail, assume that Φ is a set of equations over $T_\Sigma(X)$. (No relationship between X and Y is assumed.) Let $\equiv$ be the smallest congruence on $T_\Sigma(Y)$

containing all pairs $u(s) \equiv u(t)$ for any substitution $u : T_\Sigma(X) \to T_\Sigma(Y)$ and $s = t \in \Phi$. For example, if $f, g, a \in \Sigma$ are of arity $2, 1, 0$ respectively, $x, y \in X$, $z, w \in Y$, and $f(x, y) = f(y, x)$ is in Φ, then $f(g(z), f(a, w)) \equiv f(f(a, w), g(z))$. The algebra $T_\Sigma(Y)/\Phi$ is defined to be the quotient $T_\Sigma(Y)/\equiv$.

The following theorem asserts formally the key property of "freeness" that we described intuitively above.

THEOREM 3.36:

(i) $T_\Sigma(Y)/\Phi$ is a model of Φ;
(ii) for any model $\mathfrak{A}$ of Φ and for any valuation $u : T_\Sigma(Y) \to \mathfrak{A}$, there exists a unique homomorphism $v : T_\Sigma(Y)/\Phi \to \mathfrak{A}$ such that $u = [\,] \circ v$, where $[\,] : T_\Sigma(Y) \to T_\Sigma(Y)/\Phi$ is the canonical homomorphism.



Proof (i) Let $u : T_\Sigma(X) \to T_\Sigma(Y)/\Phi$ be any valuation and let $s = t$ be any equation in Φ. We wish to show that $T_\Sigma(Y)/\Phi, u \vDash s = t$; that is, $u(s) = u(t)$. For $x \in X$, let $v(x) \in T_\Sigma(Y)$ be any element of the congruence class $u(x)$. Then for any $x \in X$, $u(x) = [v(x)]$. Extend v to a substitution $v : T_\Sigma(X) \to T_\Sigma(Y)$. Since the two valuations u and $v \circ [\,]$ agree on the generating set X, they are equal. But $v(s) \equiv v(t)$ by definition of $\equiv$, therefore $u(s) = u(t)$ and $T_\Sigma(Y)/\Phi, u \vDash s = t$. Since u was arbitrary, $T_\Sigma(Y)/\Phi \vDash s = t$.

(ii) The kernel of u is a congruence, and since $\mathfrak{A} \vDash \Phi$, it contains all substitution instances of equations in Φ. Since $\equiv$, the kernel of the canonical homomorphism $[\,]$, is the minimal such congruence, $\equiv$ refines $\ker u$. The result then follows from Lemma 3.29. ■

EXAMPLE 3.37: The free monoid on generators A is the asterate A^*, the set of finite length strings over A with the operation of concatenation and identity element ε, the empty string.

EXAMPLE 3.38: The free commutative ring on generators X is the ring of polynomials $\mathbb{Z}[X]$. In particular, the free commutative ring on no generators is $\mathbb{Z}$.

EXAMPLE 3.39: The free vector space over $\mathbb{R}$ on n generators is just the Euclidean space $\mathbb{R}^n$.

A Deductive System

A sound and complete deductive system for equational logic can be obtained from the definition of congruence.

AXIOM SYSTEM 3.40:

(REF) $s = s$

(SYM) $$\frac{s = t}{t = s}$$

(TRANS) $$\frac{s = t, \quad t = u}{s = u}$$

(CONG) $$\frac{s_i = t_i,\ 1 \le i \le n}{f(s_1, \ldots, s_n) = f(t_1, \ldots, t_n)}.$$

These are the laws of *reflexivity*, *symmetry*, *transitivity*, and *congruence*, respectively. Let X be a set of variables and let Φ be a set of equations over $T_\Sigma(X)$. A *proof* of an equation $s = t$ from assumptions Φ in this system consists of a sequence of equations containing $s = t$ in which each equation is either

- a substitution instance of an equation in Φ, or
- a consequence of an axiom or rule whose premises occur earlier in the sequence.

We write $\Phi \vdash s = t$ if there is a proof of $s = t$ from assumptions Φ.

Starting from a set of equations A on a term algebra, the four rules of Axiom System 3.40 generate exactly the congruence closure of A. This is because the rules implement exactly the monotone operators (3.3.6)–(3.3.9) defining congruence closure. If A is the set of all substitution instances over $T_\Sigma(Y)$ of equations in Φ, the rules will therefore generate exactly those pairs $s = t$ such that $T_\Sigma(Y)/\Phi, [\,] \vDash s = t$.

THEOREM 3.41 (SOUNDNESS AND COMPLETENESS OF EQUATIONAL LOGIC): Let Φ be a set of equations over $T_\Sigma(X)$ and let $s = t$ be an equation over $T_\Sigma(Y)$. Let $[\,] : T_\Sigma(Y) \to T_\Sigma(Y)/\Phi$ be the canonical homomorphism. The following three statements are equivalent:

(i) $\Phi \vdash s = t$;

(ii) $[s] = [t]$;
(iii) $s = t \in \mathbf{Th}\,\mathbf{Mod}\,\Phi$.

Proof (i) $\Longleftrightarrow$ (ii) The four rules of Axiom System 3.40 implement exactly the four monotone set operators (3.3.6)–(3.3.9) on $T_\Sigma(Y)$ defining congruence closure. Let T be this set of operators and let $T^\dagger$ be the associated closure operator as described in Section 1.7. Let R be the set of substitution instances over $T_\Sigma(Y)$ of equations in Φ. By Theorem 1.12, $T^\dagger(R)$ is the congruence closure of R, and this is ker [ ] by definition. But by the definition of proof,

$$\{s = t \mid \Phi \vdash s = t\} \quad = \quad T^\omega(R),$$

and $T^\omega(R) = T^\dagger(R)$ since the operators T are finitary (Exercises 1.17 and 1.18).

(iii) $\Longrightarrow$ (ii) Since $[s] = [t]$ iff $T_\Sigma(Y)/\Phi, [\ ] \vDash s = t$, this follows immediately from Theorem 3.36(i).

(ii) $\Longrightarrow$ (iii) Let $\mathfrak{A}$ be any model of Φ. By Theorem 3.36(ii), for any valuation $u : T_\Sigma(Y) \to \mathfrak{A}$ there exists a valuation $v : T_\Sigma(Y)/\Phi \to \mathfrak{A}$ such that $u = [\ ] \circ v$. Since $[s] = [t]$ by assumption, we have $u(s) = v([s]) = v([t]) = u(t)$. Since u was arbitrary, $\mathfrak{A} \vDash s = t$. Since $\mathfrak{A}$ was arbitrary, $s = t$ is a logical consequence of Φ. ■

The HSP Theorem

We conclude this section with a remarkable theorem of Birkhoff that characterizes varieties in terms of closure properties. It states that a class of algebras is a variety—that is, it is defined by equations—if and only if it is closed under the formation of subalgebras, products, and homomorphic images.

Homomorphic images and subalgebras were defined in Section 3.3. Products are defined as follows. Recall from Section 1.2 that if $\{A_i \mid i \in I\}$ is an indexed family of sets, the *Cartesian product* of the A_i is the set $\prod_{i\in I} A_i$ of all functions $a : I \to \bigcup_{i\in I} A_i$ such that $a(i) \in A_i$. We write a_i for $a(i)$ and think of an element $a \in \prod_{i\in I} A_i$ as a tuple $(a_i \mid i \in I)$ whose components are indexed by I. If $\{\mathfrak{A}_i \mid i \in I\}$ is an indexed family of algebras, where the carrier of $\mathfrak{A}_i$ is A_i, the *product* $\mathfrak{A} = \prod_{i\in I} \mathfrak{A}_i$ is the algebra whose carrier is $A = \prod_{i\in I} A_i$ and whose distinguished functions $f^{\mathfrak{A}} : A^n \to A$ are defined componentwise: $f^{\mathfrak{A}}(a)_i = f^{\mathfrak{A}_i}(a_i)$.

Let Φ be a set of equations over variables X. Recall from Section 3.3 that $\mathbf{Mod}\,\Phi$ denotes the class of models of Φ, that is, $\mathbf{Mod}\,\Phi = \{\mathfrak{A} \mid \mathfrak{A} \vDash \Phi\}$; if $\mathfrak{A}$ is a Σ-algebra, then $\mathbf{Th}\,\mathfrak{A}$ denotes the set of equations valid in $\mathfrak{A}$; and if $\mathcal{D}$ is a class of Σ-algebras, then $\mathbf{Th}\,\mathcal{D}$ denotes the set of equations valid in all elements of $\mathcal{D}$. Recall that a *variety* is a class of algebras of the form $\mathbf{Mod}\,\Phi$ for some set of equations Φ.

We define the operators **H**, **S**, and **P** that when applied to a class $\mathcal{D}$ of algebras give the class of (isomorphic copies of) homomorphic images, subalgebras, and products of algebras in $\mathcal{D}$, respectively. Thus we write $\mathbf{H}\,\mathbf{S}\,\mathbf{P}\,\mathcal{D}$ for the class of all homomorphic images of subalgebras of products of algebras in $\mathcal{D}$. We write $\{\mathbf{H},\mathbf{S},\mathbf{P}\}^*\,\mathcal{D}$ for the smallest class of algebras containing $\mathcal{D}$ and closed under the formation of homomorphic images, subalgebras, and products.

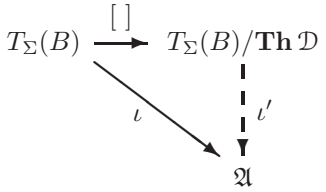
THEOREM 3.42: Let $\mathcal{D}$ be a class of algebras. Then

$$\mathbf{Mod}\,\mathbf{Th}\,\mathcal{D} \;=\; \mathbf{H}\,\mathbf{S}\,\mathbf{P}\,\mathcal{D} \;=\; \{\mathbf{H},\mathbf{S},\mathbf{P}\}^*\,\mathcal{D}.$$

Proof We show first that $\{\mathbf{H},\mathbf{S},\mathbf{P}\}^*\,\mathcal{D} \subseteq \mathbf{Mod}\,\mathbf{Th}\,\mathcal{D}$. Surely $\mathcal{D} \subseteq \mathbf{Mod}\,\mathbf{Th}\,\mathcal{D}$. Any product of algebras satisfying $\mathbf{Th}\,\mathcal{D}$ also satisfies $\mathbf{Th}\,\mathcal{D}$, because an equation holds in the product iff it holds in all the factor algebras. Any subalgebra of an algebra $\mathfrak{A}$ satisfying $\mathbf{Th}\,\mathcal{D}$ also satisfies $\mathbf{Th}\,\mathcal{D}$, because any valuation over the subalgebra is a valuation over $\mathfrak{A}$, therefore must verify any equation in $\mathbf{Th}\,\mathcal{D}$. Finally, any homomorphic image of an algebra satisfying $\mathbf{Th}\,\mathcal{D}$ also satisfies $\mathbf{Th}\,\mathcal{D}$ by Theorem 3.28.

The inclusion $\mathbf{H}\,\mathbf{S}\,\mathbf{P}\,\mathcal{D} \subseteq \{\mathbf{H},\mathbf{S},\mathbf{P}\}^*\,\mathcal{D}$ is obvious.

Finally, we show that $\mathbf{Mod}\,\mathbf{Th}\,\mathcal{D} \subseteq \mathbf{H}\,\mathbf{S}\,\mathbf{P}\,\mathcal{D}$. Suppose $\mathfrak{A} \in \mathbf{Mod}\,\mathbf{Th}\,\mathcal{D}$. We wish to show that $\mathfrak{A} \in \mathbf{H}\,\mathbf{S}\,\mathbf{P}\,\mathcal{D}$. Let B be any set of generators of $\mathfrak{A}$. Considering B as just a set, form the free algebra $T_\Sigma(B)/\mathbf{Th}\,\mathcal{D}$. By Theorem 3.36(ii), the valuation $\iota : T_\Sigma(B) \to \mathfrak{A}$ such that $\iota \restriction B$ is the identity factors through $T_\Sigma(B)/\mathbf{Th}\,\mathcal{D}$, giving a homomorphism $\iota' : T_\Sigma(B)/\mathbf{Th}\,\mathcal{D} \to \mathfrak{A}$.



Moreover, since B generates $\mathfrak{A}$, ι' is an epimorphism, thus $\mathfrak{A}$ is a homomorphic image of $T_\Sigma(B)/\mathbf{Th}\,\mathcal{D}$. It thus suffices to show that $T_\Sigma(B)/\mathbf{Th}\,\mathcal{D} \in \mathbf{S}\,\mathbf{P}\,\mathcal{D}$.

For each pair s,t in $T_\Sigma(B)$ such that $[s] \neq [t]$, the equation $s = t$ cannot be in $\mathbf{Th}\,\mathcal{D}$. Thus there must exist an algebra $\mathfrak{B}_{s,t} \in \mathcal{D}$ and valuation $u_{s,t} : T_\Sigma(B) \to \mathfrak{B}_{s,t}$ such that $u_{s,t}(s) \neq u_{s,t}(t)$. Set

$$\mathfrak{B} \;=\; \prod_{[s]\neq[t]} \mathfrak{B}_{s,t}$$

and let $u : T_\Sigma(B) \to \mathfrak{B}$ be the valuation

$$u(r) \quad = \quad \prod_{[s]\neq[t]} u_{s,t}(r).$$

Since all components $\mathfrak{B}_{s,t}$ are models of $\mathbf{Th}\,\mathcal{D}$, so is their product $\mathfrak{B}$. By Theorem 3.36(ii), u factors through $T_\Sigma(B)/\mathbf{Th}\,\mathcal{D}$ as $[\ ] \circ v$, where $v : T_\Sigma(B)/\mathbf{Th}\,\mathcal{D} \to \mathfrak{B}$. Moreover, v is injective, since if $[s] \neq [t]$, then $v(s) \neq v(t)$ in at least one component, namely $\mathfrak{B}_{s,t}$. Thus $T_\Sigma(B)/\mathbf{Th}\,\mathcal{D}$ is isomorphic under v to a subalgebra of the product $\mathfrak{B}$. ■

Corollary 3.43 (Birkhoff): Let $\mathcal{D}$ be a class of Σ-algebras. The following are equivalent:

(i) $\mathcal{D}$ is a variety;
(ii) $\mathcal{D} = \mathbf{H}\,\mathbf{S}\,\mathbf{P}\,\mathcal{D}$;
(iii) $\mathcal{D} = \{\mathbf{H}, \mathbf{S}, \mathbf{P}\}^*\,\mathcal{D}$.

Proof That (ii) and (iii) are equivalent and imply (i) are immediate from Theorem 3.42. That (i) implies (ii) follows from Theorem 3.42 and the fact that for any set of formulas Φ, $\mathbf{Mod}\,\Phi = \mathbf{Mod}\,\mathbf{Th}\,\mathbf{Mod}\,\Phi$ (Exercise 3.21). ■

3.4 Predicate Logic

First-order predicate logic is the logic of predicates and quantification ($\forall$, $\exists$) over elements of a structure.

Syntax

Syntactically, we start with a countable signature as with equational logic, except that we include some *relation* or *predicate symbols* $p, q, r, \ldots$ in addition to the function symbols $f, g, \ldots$. A *signature* or *vocabulary* then consists of a set Σ of function and relation symbols, each with an associated *arity* (number of inputs). Function and relation symbols of arity 0, 1, 2, 3, and n are called *nullary, unary, binary, ternary*, and *n-ary*, respectively. Nullary elements are often called *constants*. One of the relation symbols may be the binary *equality symbol* $=$. In most applications, Σ is finite.

The language consists of:

- the function and relation symbols in Σ
- a countable set X of *individual variables* $x, y, \ldots$
- the propositional connectives $\to$ and $\mathbf{0}$
- the universal quantifier symbol $\forall$ ("for all")
- parentheses.

As in Section 3.2, the other propositional connectives $\vee$, $\wedge$, $\mathbf{1}$, $\neg$, and $\leftrightarrow$ can all be defined in terms of $\to$ and $\mathbf{0}$. Similarly, we will define below the existential quantifier $\exists$ ("there exists") in terms of $\forall$.

Terms $s, t, \ldots$ are exactly as in equational logic (see Section 3.3). A term is a *ground term* if it contains no variables.

Formulas $\varphi, \psi, \ldots$ are defined inductively. A formula is either

- an *atomic formula* $p(t_1, \ldots, t_n)$, where p is an n-ary relation symbol and $t_1, \ldots, t_n$ are terms; or
- $\varphi \to \psi$, $\mathbf{0}$, or $\forall x\ \varphi$, where φ and ψ are formulas and x is a variable.

Intuitively, in the formula $\forall x\ \varphi$, we think of φ as a property of an object x; then the formula $\forall x\ \varphi$ says that that property φ holds for all objects x.

The other propositional operators are defined from $\to$ and $\mathbf{0}$ as described in Section 3.2. The quantifier $\exists$ is defined as follows:

$$\exists x\ \varphi \quad \overset{\text{def}}{\iff} \quad \neg\forall x\ \neg\varphi. \tag{3.4.1}$$

Intuitively, in the formula $\exists x\ \varphi$, we again think of φ as a property of an object x; then the formula $\exists x\ \varphi$ says that that there exists an object x for which the property φ holds. The formal definition (3.4.1) asserts the idea that there exists an x for which φ is true if and only if it is not the case that for all x, φ is false.

As with propositional logic, we will assume a natural precedence of the operators and use parentheses where necessary to ensure that a formula can be read in one and only one way. The precedence of the propositional operators is the same as in Section 3.2. The quantifier $\forall$ binds more tightly than the propositional operators; thus $\forall x\ \varphi \to \psi$ should be parsed as $(\forall x\ \varphi) \to \psi$.

The family of languages we have just defined will be denoted collectively by $L_{\omega\omega}$. The two subscripts ω refer to the fact that we allow only finite (that is, $< \omega$) conjunctions and disjunctions and finitely many variables.

EXAMPLE 3.44: The first-order language of number theory is suitable for expressing properties of the natural numbers $\mathbb{N}$. The signature consists of binary function

symbols + and · (written in infix), constants 0 and 1, and binary relation symbol = (also written in infix). A typical term is $(x+1)\cdot y$ and a typical atomic formula is $x+y=z$. The formula

$$\forall x\ \exists y\ (x \le y \wedge \forall z\ (z \mid y \rightarrow (z = 1 \vee z = y)))$$

expresses the statement that there are infinitely many primes. Here $s \le t$ is an abbreviation for $\exists w\ s + w = t$ and $s \mid t$ (read "s divides t") is an abbreviation for $\exists w\ s \cdot w = t$.

Scope, Bound and Free Variables

Let Q be either $\forall$ or $\exists$. If $\mathsf{Q}x\ \varphi$ occurs as a subformula of some formula ψ, then that occurrence of φ in ψ is called the *scope* of that occurrence of $\mathsf{Q}x$ in ψ. An occurrence of a variable y in ψ that occurs in a term is a *free occurrence of y in ψ* if it is not in the scope of any quantifier $\mathsf{Q}y$ with the same variable y. If $\mathsf{Q}y\ \varphi$ occurs as a subformula of ψ and y occurs free in φ, then that occurrence of y is said to be *bound to* that occurrence of $\mathsf{Q}y$. Thus an occurrence of y in ψ is bound to the $\mathsf{Q}y$ with smallest scope containing that occurrence of y, if such a $\mathsf{Q}y$ exists; otherwise it is free.

We say that a term t is *free for y in φ* if no free occurrence of y in φ occurs in the scope of a quantifier $\mathsf{Q}x$, where x occurs in t. This condition says that it is safe to substitute t for free occurrences of y in φ without fear of some variable x of t being inadvertently captured by a quantifier.

Example 3.45: In the formula

$$\exists x\ ((\forall y\ \exists x\ q(x,y)) \wedge p(x,y,z)),$$

the scope of the first $\exists x$ is $(\forall y\ \exists x\ q(x,y)) \wedge p(x,y,z)$, the scope of the $\forall y$ is $\exists x\ q(x,y)$, and the scope of the second $\exists x$ is $q(x,y)$. The occurrence of x in $q(x,y)$ is bound to the second $\exists x$. The x in $p(x,y,z)$ occurs free in the subformula $(\forall y\ \exists x\ q(x,y)) \wedge p(x,y,z)$ but is bound to the first $\exists x$. The occurrence of y in $q(x,y)$ is bound to the $\forall y$, but the occurrence of y in $p(x,y,z)$ is free. The only occurrence of z in the formula is a free occurrence. The term $f(x)$ is not free for either y or z in the formula, because substitution of $f(x)$ for y or z would result in the capture of x by the first $\exists x$.

Note that the adjectives "free" and "bound" apply not to variables but to *occurrences* of variables in a formula. A formula may have free and bound occurrences

of the same variable. For example, the variable y in the formula of Example 3.45 has one free and one bound occurrence. Note also that occurrences of variables in quantifiers—occurrences of the form $\forall y$ and $\exists y$—do not figure in the definition of free and bound.

A variable is called a *free variable* of a formula φ if it has a free occurrence in φ. The notation $\varphi[x_1/t_1, \ldots, x_n/t_n]$ or $\varphi[x_i/t_i \mid 1 \leq i \leq n]$ denotes the formula φ with all free occurrences of x_i replaced with t_i, $1 \leq i \leq n$. The substitution is done for all variables simultaneously. Note that $\varphi[x/s, y/t]$ can differ from $\varphi[x/s][y/t]$ if s has an occurrence of y. Although notationally similar, the substitution operator $[x/t]$ should not be confused with the function-patching operator defined in Section 1.3.

We occasionally write $\varphi(x_1, \ldots, x_n)$ to indicate that all free variables of φ are among $x_1, \ldots, x_n$. The variables $x_1, \ldots, x_n$ need not all appear in $\varphi(x_1, \ldots, x_n)$, however. When $\varphi = \varphi(x_1, \ldots, x_n)$, we sometimes write $\varphi(t_1, \ldots, t_n)$ instead of $\varphi[x_1/t_1, \ldots, x_n/t_n]$.

A formula is a *closed formula* or *sentence* if it contains no free variables. The *universal closure* of a formula φ is the sentence obtained by preceding φ with enough universal quantifiers $\forall x$ to bind all the free variables of φ.

Semantics

A *relational structure* over signature Σ is a structure $\mathfrak{A} = (A, \mathfrak{m}_{\mathfrak{A}})$ where A is a nonempty set, called the *carrier* or *domain* of $\mathfrak{A}$, and $\mathfrak{m}_{\mathfrak{A}}$ is a function assigning an n-ary function $f^{\mathfrak{A}} : A^n \to A$ to each n-ary function symbol $f \in \Sigma$ and an n-ary relation $p^{\mathfrak{A}} \subseteq A^n$ to each n-ary relation symbol $p \in \Sigma$. As with equational logic, nullary functions are considered elements of A; thus constant symbols $c \in \Sigma$ are interpreted as elements $c^{\mathfrak{A}} \in A$.

As in equational logic, we define a *valuation* to be a Σ-homomorphism $u : T_\Sigma(X) \to \mathfrak{A}$. A valuation u is uniquely determined by its values on the variables X.

Given a valuation u, we define $u[x/a]$ to be the new valuation obtained from u by changing the value of x to a and leaving the values of the other variables intact; thus

$$\begin{aligned} u[x/a](y) &\stackrel{\text{def}}{=} u(y), \quad y \neq x, \\ u[x/a](x) &\stackrel{\text{def}}{=} a. \end{aligned}$$

This is the same as in equational logic.

The *satisfaction relation* $\vDash$ is defined inductively as follows:

$$\begin{aligned}
\mathfrak{A}, u \vDash p(t_1, \ldots, t_n) &\overset{\text{def}}{\iff} p^{\mathfrak{A}}(u(t_1), \ldots, u(t_n)) \\
\mathfrak{A}, u \vDash \varphi \to \psi &\overset{\text{def}}{\iff} (\mathfrak{A}, u \vDash \varphi \Longrightarrow \mathfrak{A}, u \vDash \psi) \\
\mathfrak{A}, u \vDash \forall x\ \varphi &\overset{\text{def}}{\iff} \text{for all } a \in A,\ \mathfrak{A}, u[x/a] \vDash \varphi.
\end{aligned}$$

It follows that

$$\begin{aligned}
\mathfrak{A}, u \vDash \varphi \vee \psi &\iff \mathfrak{A}, u \vDash \varphi \text{ or } \mathfrak{A}, u \vDash \psi \\
\mathfrak{A}, u \vDash \varphi \wedge \psi &\iff \mathfrak{A}, u \vDash \varphi \text{ and } \mathfrak{A}, u \vDash \psi \\
\mathfrak{A}, u \vDash \neg\varphi &\iff \mathfrak{A}, u \nvDash \varphi;\ \text{that is, if it is } \textit{not} \text{ the case that } \mathfrak{A}, u \vDash \varphi \\
\mathfrak{A}, u \vDash \exists x\ \varphi &\iff \text{there exists an } a \in A \text{ such that } \mathfrak{A}, u[x/a] \vDash \varphi.
\end{aligned}$$

Also, $\mathfrak{A}, u \nvDash \mathbf{0}$ and $\mathfrak{A}, u \vDash \mathbf{1}$.

If $\mathfrak{A}, u \vDash \varphi$, we say that φ is *true in* $\mathfrak{A}$ *under valuation* u, or that $\mathfrak{A}, u$ is a *model* of φ, or that $\mathfrak{A}, u$ *satisfies* φ. If Φ is a set of formulas, we write $\mathfrak{A}, u \vDash \Phi$ if $\mathfrak{A}, u \vDash \varphi$ for all $\varphi \in \Phi$ and say that $\mathfrak{A}, u$ *satisfies* Φ. If φ is true in all models of Φ, we write $\Phi \vDash \varphi$ and say that φ is a *logical consequence*[3] of Φ. If $\varnothing \vDash \varphi$, we write $\vDash \varphi$ and say that φ is *valid*.

It can be shown that if φ is a sentence, then $\vDash$ does not depend on the valuation u; that is, if $\mathfrak{A}, u \vDash \varphi$ for some u, then $\mathfrak{A}, u \vDash \varphi$ for all u (Exercise 3.29). In this case, we omit the u and just write $\mathfrak{A} \vDash \varphi$. If Φ is a set of sentences, then $\mathfrak{A} \vDash \Phi$ means that $\mathfrak{A} \vDash \varphi$ for all $\varphi \in \Phi$.

Two formulas φ, ψ are said to be *logically equivalent* if $\vDash \varphi \leftrightarrow \psi$.

The following lemma establishes a relationship between the function-patching operator $[x/a]$ on valuations and the substitution operator $[x/t]$ on terms and formulas.

Lemma 3.46:

(i) For any valuation u and terms $s, t \in T_\Sigma(X)$,

$$u[x/u(t)](s) \quad = \quad u(s[x/t]).$$

(ii) If t is free for x in φ, then

$$\mathfrak{A}, u[x/u(t)] \vDash \varphi \iff \mathfrak{A}, u \vDash \varphi[x/t].$$

3 This notion of logical consequence is slightly different from the one used in equational logic (Section 3.3). There, the free variables of formulas were assumed to be implicitly universally quantified. We abandon that assumption here because we have explicit quantification.

Proof (i) Proceeding by induction on the structure of s, if s is the variable x, then

$$u[x/u(t)](x) \quad = \quad u(t) \quad = \quad u(x[x/t]).$$

If s is a variable y different from x, then

$$u[x/u(t)](y) \quad = \quad u(y) \quad = \quad u(y[x/t]).$$

Finally, if $s = f(t_1, \ldots, t_n)$, then

$$\begin{aligned} u[x/u(t)](f(t_1, \ldots, t_n)) &= f^{\mathfrak{A}}(u[x/u(t)](t_1), \ldots, u[x/u(t)](t_n)) \\ &= f^{\mathfrak{A}}(u(t_1[x/t]), \ldots, u(t_n[x/t])) \\ &= u(f(t_1[x/t], \ldots, t_n[x/t])) \\ &= u(f(t_1, \ldots, t_n)[x/t]). \end{aligned}$$

(ii) We proceed by induction on the structure of φ. For atomic formulas, using (i) we have that

$$\begin{aligned} \mathfrak{A}, u[x/u(t)] \vDash p(t_1, \ldots, t_n) &\iff p^{\mathfrak{A}}(u[x/u(t)](t_1), \ldots, u[x/u(t)](t_n)) \\ &\iff p^{\mathfrak{A}}(u(t_1[x/t]), \ldots, u(t_n[x/t])) \\ &\iff \mathfrak{A}, u \vDash p(t_1[x/t], \ldots, t_n[x/t]) \\ &\iff \mathfrak{A}, u \vDash p(t_1, \ldots, t_n)[x/t]. \end{aligned}$$

For formulas of the form $\varphi \to \psi$, if t is free for x in $\varphi \to \psi$, then t is free for x in both φ and ψ. Then

$$\begin{aligned} \mathfrak{A}, u[x/u(t)] \vDash \varphi \to \psi &\iff (\mathfrak{A}, u[x/u(t)] \vDash \varphi \implies \mathfrak{A}, u[x/u(t)] \vDash \psi) \\ &\iff (\mathfrak{A}, u \vDash \varphi[x/t] \implies \mathfrak{A}, u \vDash \psi[x/t]) \\ &\iff \mathfrak{A}, u \vDash \varphi[x/t] \to \psi[x/t] \\ &\iff \mathfrak{A}, u \vDash (\varphi \to \psi)[x/t]. \end{aligned}$$

Finally, for formulas of the form $\forall y\ \varphi$, if x has no free occurrence in $\forall y\ \varphi$, then the result is a straightforward consequence of Exercise 3.29. This includes the case $y = x$. Otherwise, y is different from x and t is free for x in φ. Since φ contains a free occurrence of x, t must not contain an occurrence of y, therefore $u(t) = u[y/a](t)$.

Then

$$\begin{aligned}
\mathfrak{A}, u[x/u(t)] \vDash \forall y\ \varphi \quad &\iff \quad \text{for all } a \in \mathfrak{A}, \quad \mathfrak{A}, u[x/u(t)][y/a] \vDash \varphi \\
&\iff \quad \text{for all } a \in \mathfrak{A}, \quad \mathfrak{A}, u[y/a][x/u(t)] \vDash \varphi \\
&\iff \quad \text{for all } a \in \mathfrak{A}, \quad \mathfrak{A}, u[y/a][x/u[y/a](t)] \vDash \varphi \\
&\iff \quad \text{for all } a \in \mathfrak{A}, \quad \mathfrak{A}, u[y/a] \vDash \varphi[x/t] \\
&\iff \quad \mathfrak{A}, u \vDash \forall y\ (\varphi[x/t]) \\
&\iff \quad \mathfrak{A}, u \vDash (\forall y\ \varphi)[x/t].
\end{aligned}$$

■

LEMMA 3.47: The following formulas are valid under the provisos indicated:

(i) $\forall x\ (\varphi \to \psi)\ \to\ (\forall x\ \varphi \to \forall x\ \psi)$;
(ii) $\forall x\ \varphi\ \to\ \varphi[x/t]$, provided t is free for x in φ;
(iii) $\varphi \to \forall x\ \varphi$, provided x does not occur free in φ.

Proof (i) Suppose that $\mathfrak{A}, u \vDash \forall x\ (\varphi \to \psi)$ and $\mathfrak{A}, u \vDash \forall x\ \varphi$. Then for any $a \in \mathfrak{A}$, $\mathfrak{A}, u[x/a] \vDash \varphi \to \psi$ and $\mathfrak{A}, u[x/a] \vDash \varphi$. By the semantics of $\to$,

$$\mathfrak{A}, u[x/a] \vDash \varphi \quad \Longrightarrow \quad \mathfrak{A}, u[x/a] \vDash \psi,$$

therefore $\mathfrak{A}, u[x/a] \vDash \psi$. Since a was arbitrary, $\mathfrak{A}, u \vDash \forall x\ \psi$. We have shown that

$$\mathfrak{A}, u \vDash \forall x\ (\varphi \to \psi) \quad \Longrightarrow \quad (\mathfrak{A}, u \vDash \forall x\ \varphi \implies \mathfrak{A}, u \vDash \forall x\ \psi),$$

which by the semantics of $\to$ implies that

$$\mathfrak{A}, u \quad \vDash \quad \forall x\ (\varphi \to \psi)\ \to\ (\forall x\ \varphi \to \forall x\ \psi).$$

Since $\mathfrak{A}$ and u were arbitrary, (i) is valid.

(ii) If $\mathfrak{A}, u \vDash \forall x\ \varphi$, then $\mathfrak{A}, u[x/u(t)] \vDash \varphi$. Since t is free for x in φ, by Lemma 3.46(ii), $\mathfrak{A}, u \vDash \varphi[x/t]$. Thus $\mathfrak{A}, u \vDash \forall x\ \varphi \to \varphi[x/t]$. Since $\mathfrak{A}, u$ was arbitrary, $\forall x\ \varphi \to \varphi[x/t]$ is valid.

(iii) Since the truth value of $\mathfrak{A}, u \vDash \varphi$ is independent of $u(x)$ if x does not occur free in φ (Exercise 3.29), we have

$$\begin{aligned}
\mathfrak{A}, u \vDash \varphi \quad &\Longrightarrow \quad \text{for any } a \in \mathfrak{A},\ \mathfrak{A}, u[x/a] \vDash \varphi \\
&\Longrightarrow \quad \mathfrak{A}, u \vDash \forall x\ \varphi.
\end{aligned}$$

Combining these implications, we have

$\mathfrak{A}, u \models \varphi \to \forall x\ \varphi.$

■

Prenex Form

A formula is in *prenex form* if it is of the form

$\mathsf{Q}_1 x_1\ \mathsf{Q}_2 x_2\ \ldots\ \mathsf{Q}_k x_k\ \varphi,$

where each Q_i is either $\forall$ or $\exists$ and φ is quantifier-free. The following lemmas will allow us to transform any formula to an equivalent formula in prenex form.

LEMMA 3.48 (CHANGE OF BOUND VARIABLE): If y is free for x in φ and if y does not occur free in φ, then the formula

$\forall x\ \varphi \leftrightarrow \forall y\ \varphi[x/y]$

is valid.

Proof ($\to$) By Lemma 3.47(ii), the formula

$\forall x\ \varphi \to \varphi[x/y]$

is valid, therefore so is

$\forall y\ (\forall x\ \varphi \to \varphi[x/y]).$

But then

$\forall y\ \forall x\ \varphi \to \forall y\ \varphi[x/y]$
$\forall x\ \varphi \to \forall y\ \forall x\ \varphi$

are valid by Lemma 3.47(i) and (iii) respectively, therefore

$\forall x\ \varphi \to \forall y\ \varphi[x/y]$

is valid.

($\leftarrow$) Since y is free for x in φ, every free occurrence of x in φ turns into a free occurrence of y in $\varphi[x/y]$. Since y does not occur free in φ, every free occurrence of y in $\varphi[x/y]$ must have come from a free occurrence of x in φ. Also, x does not occur free in $\varphi[x/y]$, since y was substituted for all free occurrences of x; and x is free

for y in $\varphi[x/y]$, since y could not have replaced a bound occurrence of x. It follows that $\varphi[x/y][y/x] = \varphi$, thus the situation is completely symmetric to the previous case, and the reverse implication follows from the argument above. ■

Neither of the two provisos in the statement of Lemma 3.48 can be omitted. We must have y free for x in φ, as can be seen by taking φ to be the formula $\exists y\ y = x + 1$ interpreted over $\mathbb{N}$; and we must not have y occurring free in φ, as can be seen by taking φ to be the formula $y \neq x + 1$ interpreted over $\mathbb{N}$.

The practical significance of Lemma 3.48 is that it can be used to change bound variable names to avoid capture during substitution. Say we wish to substitute into a formula φ a term t with an occurrence of x that would be captured by a quantifier $\forall x$. We can avoid the capture by replacing the x in the quantifier $\forall x$ and all free occurrences of x in the scope of the $\forall x$ with y, where y is a new variable (one with no occurrences in φ). The lemma says that the resulting formula is equivalent.

LEMMA 3.49: If x does not occur free in ψ, then the following formulas are valid:

$$\begin{aligned}
(\forall x\ \varphi) \to \psi &\quad\leftrightarrow\quad \exists x\ (\varphi \to \psi)\\
(\exists x\ \varphi) \to \psi &\quad\leftrightarrow\quad \forall x\ (\varphi \to \psi)\\
\psi \to (\forall x\ \varphi) &\quad\leftrightarrow\quad \forall x\ (\psi \to \varphi)\\
\psi \to (\exists x\ \varphi) &\quad\leftrightarrow\quad \exists x\ (\psi \to \varphi).
\end{aligned}$$

Proof Exercise 3.31. ■

A special case of the first two formulas of Lemma 3.49 are the formulas

$$\begin{aligned}
\neg\forall x\ \varphi &\quad\leftrightarrow\quad \exists x\ \neg\varphi\\
\neg\exists x\ \varphi &\quad\leftrightarrow\quad \forall x\ \neg\varphi,
\end{aligned}$$

which are essentially the definition of $\exists$.

LEMMA 3.50 (PRENEX FORM): Every formula is equivalent to a formula in prenex form.

Proof Quantifiers can be moved outward outside all occurrences of the propositional operator $\to$ by applying the rules of Lemma 3.49 from left to right. If we wish to apply one of these rules at some point and cannot because of the proviso regarding free variables, then Lemma 3.48 can be used to rename the bound variables. ■

A Deductive System

In this section we give a complete Hilbert-style deductive system for first-order logic.

AXIOM SYSTEM 3.51: The axioms of our deductive system consist of the laws of propositional logic and the universal closures of the valid formulas of Lemma 3.47:

(i) $\forall x\ (\varphi \to \psi)\ \to\ (\forall x\ \varphi \to \forall x\ \psi)$;

(ii) $\forall x\ \varphi\ \to\ \varphi[x/t]$, provided t is free for x in φ;

(iii) $\varphi \to \forall x\ \varphi$, provided x does not occur free in φ.

There are two rules of inference:

(MP) $\dfrac{\varphi,\quad \varphi \to \psi}{\psi}$

(GEN) $\dfrac{\varphi}{\forall x\ \varphi}$.

When reasoning in the presence of assumptions, the rule (GEN) may only be applied with the proviso that x does not occur free in any assumption.

The rule (MP) is the rule *modus ponens* of propositional logic (Section 3.2). The rule (GEN) is known as the *generalization rule*.

This system is easily shown to be sound (Exercise 3.32). Intuitively, the generalization rule is sound because if one could prove $\varphi(x)$ without any assumptions about x, then $\varphi(x)$ is true for arbitrary x.

The Deduction Theorem

THEOREM 3.52 (DEDUCTION THEOREM): For any set of formulas Φ and formulas φ, ψ,

$$\Phi \cup \{\varphi\} \vdash \psi \quad \Longleftrightarrow \quad \Phi \vdash \varphi \to \psi.$$

Proof The proof is identical to the corresponding proof for propositional logic (Theorem 3.9), except that in the direction ($\Longrightarrow$) there is an extra case for the rule (GEN). Suppose $\Phi \cup \{\varphi\} \vdash \forall x\ \psi$ by an application of the rule (GEN). Then $\Phi \cup \{\varphi\} \vdash \psi$ by a shorter proof, and x is not free in φ or any formula of Φ. By the induction hypothesis, $\Phi \vdash \varphi \to \psi$. By (GEN), $\Phi \vdash \forall x\ (\varphi \to \psi)$. By Axiom 3.51(i)

and an application of modus ponens,

$$\Phi \quad \vdash \quad \forall x\ \varphi \to \forall x\ \psi.$$

But since x does not occur free in φ, by Axiom 3.51(iii) and the transitivity of implication we have

$$\Phi \quad \vdash \quad \varphi \to \forall x\ \psi.$$

■

Completeness

As with propositional logic, we will prove completeness of the system 3.51 by proving that every consistent set of formulas Φ, finite or infinite, has a model. However, the situation is complicated somewhat by the presence of quantifiers. We must ensure that the model we construct contains a witness a for every existential formula $\exists x\,\psi$ in Φ. We use a technique of Henkin (1949) in which we include extra variables to provide these witnesses.

We augment the language with the new variables as follows. Let X_0 be the original set of variables, and let L_0 be the original set of formulas over these variables. Now suppose X_n and L_n have been constructed. For each formula $\varphi \in L_n$, let x_φ be a new variable. Let

$$X_{n+1} \quad \stackrel{\text{def}}{=} \quad X_n \cup \{x_\varphi \mid \varphi \in L_n\},$$

and let L_{n+1} be the language augmented with these new variables. Let

$$X_\omega \quad \stackrel{\text{def}}{=} \quad \bigcup_n X_n$$
$$L_\omega \quad \stackrel{\text{def}}{=} \quad \bigcup_n L_n.$$

The sets X_ω and L_ω are still countable, because they are countable unions of countable sets (Exercise 1.21).

Now let $\Psi \subseteq L_\omega$ be the set of all formulas of the form

$$\exists x\ \psi \quad \to \quad \psi[x/x_{\exists x\psi}]. \tag{3.4.2}$$

Intuitively, this formula says that if there exists an element x satisfying ψ at all, then the value of $x_{\exists x\psi}$ gives such an element.

LEMMA 3.53: Let $\Phi \subseteq L_0$. If Φ is consistent, then so is $\Phi \cup \Psi$.

Proof Suppose $\Phi \cup \Psi$ is refutable. Then there is a minimal finite subset Ψ' of Ψ such that $\Phi \cup \Psi'$ is refutable. Also, there must be an existential formula $\varphi = \exists x\, \psi$ and $\varphi \to \psi[x/x_\varphi] \in \Psi'$ such that x_φ does not appear in any other formula in $\Phi \cup \Psi'$: surely x_φ does not occur in any formula of Φ, since $\Phi \subseteq L_0$; and if x_φ occurs in $\exists y \rho \to \rho[y/x_{\exists y \rho}] \in \Psi'$, then x_φ occurs in $\exists y \rho$, therefore $x_{\exists y \rho}$ was introduced strictly later in the inductive definition of X_ω. This can be avoided by choosing x_φ to be the latest such variable introduced among those appearing in Ψ'.

Let $\Psi'' \stackrel{\text{def}}{=} \Psi' - \{\varphi \to \psi[x/x_\varphi]\}$. Then

$$\Phi \cup \Psi'' \cup \{\varphi \to \psi[x/x_\varphi]\} \quad \vdash \quad \mathbf{0},$$

therefore by the deduction theorem (Theorem 3.52),

$$\Phi \cup \Psi'' \quad \vdash \quad \neg(\varphi \to \psi[x/x_\varphi]).$$

By propositional logic, we have

$$\Phi \cup \Psi'' \quad \vdash \quad \varphi$$
$$\Phi \cup \Psi'' \quad \vdash \quad \neg\psi[x/x_\varphi].$$

By (GEN),

$$\Phi \cup \Psi'' \quad \vdash \quad \forall x_\varphi\, \neg\psi[x/x_\varphi];$$

changing the bound variable (Exercise 3.30) then gives

$$\Phi \cup \Psi'' \quad \vdash \quad \forall x\, \neg\psi.$$

Since $\varphi = \exists x\, \psi$, this leads immediately to a refutation of $\Phi \cup \Psi''$, contradicting the minimality of Ψ'. ■

THEOREM 3.54 (COMPLETENESS): The deductive system (3.51) is complete; that is, any consistent set of formulas has a model.

Proof Suppose Φ is consistent. By Lemma 3.53, so is $\Phi \cup \Psi$. Extend $\Phi \cup \Psi$ to a maximal consistent set $\widehat{\Phi}$ as in the proof of Lemma 3.10. As argued there, for all $\varphi \in L_\omega$, either $\varphi \in \widehat{\Phi}$ or $\neg\varphi \in \widehat{\Phi}$, and $\widehat{\Phi}$ is deductively closed in the sense that if $\widehat{\Phi} \vdash \psi$ then $\psi \in \widehat{\Phi}$.

Now we construct a model $\mathfrak{A}$ from $\widehat{\Phi}$. The domain of $\mathfrak{A}$ will be the set of terms

$T_\Sigma(X_\omega)$. The function symbols f are interpreted in $\mathfrak{A}$ syntactically:

$$f^{\mathfrak{A}}(t_1, \ldots, t_n) \stackrel{\text{def}}{=} f(t_1, \ldots, t_n).$$

The truth value of atomic formulas is defined as follows:

$$p^{\mathfrak{A}}(t_1, \ldots, t_n) = \mathbf{1} \iff p(t_1, \ldots, t_n) \in \widehat{\Phi}.$$

Let $u : X_\omega \to T_\Sigma(X_\omega)$ be the valuation $x \mapsto x$. The unique homomorphic extension $u : T_\Sigma(X_\omega) \to T_\Sigma(X_\omega)$ is the identity map. (Here all terms, including variables x, are both syntactic and semantic objects.) We prove by induction on the structure of formulas that for all $\varphi \in L_\omega$,

$$\mathfrak{A}, u \vDash \varphi \iff \varphi \in \widehat{\Phi}.$$

The basis of the induction,

$$\mathfrak{A}, u \vDash p(t_1, \ldots, t_n) \iff p(t_1, \ldots, t_n) \in \widehat{\Phi},$$

is by the definition of $p^{\mathfrak{A}}$. The inductive argument for $\to$ is the same as in the propositional case (Lemma 3.10). Finally, for the case of the existential quantifier, we show that

$$\mathfrak{A}, u \vDash \exists y\ \varphi \iff \exists y\ \varphi \in \widehat{\Phi}.$$

By the definition of the meaning of $\exists$,

$$\mathfrak{A}, u \vDash \exists y\ \varphi \iff \exists t \in T_\Sigma(X_\omega)\ \mathfrak{A}, u[y/t] \vDash \varphi.$$

Assume without loss of generality that all quantified variables in φ have been renamed so as to be different from variables appearing in t; thus t is free for y in φ. Then

$$\begin{aligned}
&\exists t \in T_\Sigma(X_\omega)\ \mathfrak{A}, u[y/t] \vDash \varphi \\
&\quad \iff \exists t \in T_\Sigma(X_\omega)\ \mathfrak{A}, u \vDash \varphi[y/t] && \text{by Lemma 3.46(ii)} \\
&\quad \iff \exists t \in T_\Sigma(X_\omega)\ \varphi[y/t] \in \widehat{\Phi} && \text{by the induction hypothesis} \\
&\quad \iff \exists y\ \varphi \in \widehat{\Phi}.
\end{aligned}$$

In the last step, the direction ($\Longrightarrow$) is from Axiom 3.51(ii) and the direction ($\Longleftarrow$) is from the fact that the formulas (3.4.2) are included in $\widehat{\Phi}$. ■

Completeness with Equality

First-order logic with equality typically means that the binary equality symbol $=$ is included in the signature Σ and that we restrict the semantics to include only models in which $=$ is interpreted as the identity relation. As it turns out, this is actually not much of a restriction: a structure $\mathfrak{A}$ for which $=^{\mathfrak{A}}$ is not equality but obeys all the laws of equality (Axioms 3.55(ii) and (iii) below) can be collapsed by a quotient construction to give an equivalent model in which $=$ is interpreted as the identity relation.

AXIOM SYSTEM 3.55: The axioms and rules of inference for first order logic with equality are:

(i) Axiom System 3.51 for first-order logic;

(ii) Axiom System 3.40 for equational logic;

(iii) the rule

$$\frac{s_i = t_i, \quad 1 \leq i \leq n}{p(s_1, \ldots, s_n) \leftrightarrow p(t_1, \ldots, t_n)}.$$

We regard (iii) as part of the rule (CONG) of Axiom System 3.40.

THEOREM 3.56 (COMPLETENESS WITH EQUALITY): Axiom System 3.55 is complete for first order logic with equality; that is, any consistent set of formulas has a model.

Proof sketch. The proof is the same as without equality (Theorem 3.54), except that instead of the term model $T_\Sigma(X_\omega)$, we take its quotient by the congruence

$$s \equiv t \quad \stackrel{\text{def}}{\Longleftrightarrow} \quad s = t \in \widehat{\Phi}.$$

The new rule (iii) ensures that $p^{\mathfrak{A}}$ is well-defined on $\equiv$-congruence classes. ■

Compactness

We proved in Section 3.4 that any consistent set of first-order formulas has a model. The *compactness theorem* is an immediate consequence of this. Recall that a set of formulas Φ is *finitely satisfiable* if all finite subsets of Φ have a model.

THEOREM 3.57 (COMPACTNESS): A set Φ of first-order formulas is satisfiable if and only if it is finitely satisfiable.

Proof The proof is the same as for propositional logic (Theorem 3.12), using Theorem 3.54. ■

The Löwenheim–Skolem Theorem

Our proof of the completeness theorem constructed models from terms, or in the presence of equality, from congruence classes of terms. Since the language has only countably many terms, the models constructed were countable (either finite or countably infinite). We thus have

COROLLARY 3.58: Let Φ be a countable set of formulas. If Φ has a model, then it has a countable model.

The first part of Theorem 3.59 below is a slight strengthening of this.

THEOREM 3.59 (LÖWENHEIM–SKOLEM): Let Φ be a countable set of formulas.

(i) If Φ has an infinite model, then it has a countably infinite model.

(ii) If Φ has a countably infinite model, then it has a model of every infinite cardinality.

Parts (i) and (ii) of Theorem 3.59 are known as the *downward* and *upward Löwenheim–Skolem theorem*, respectively.

Proof sketch. (i) Suppose $\mathfrak{A}, u \vDash \Phi$, $\mathfrak{A}$ infinite. Let

$$\Phi^+ \stackrel{\text{def}}{=} \Phi \cup \{\theta_n \mid n \geq 0\},$$

where

$$\theta_n \stackrel{\text{def}}{=} \exists x_1\, \exists x_2\, \ldots\, \exists x_n \bigwedge_{1 \leq i < j \leq n} x_i \neq x_j.$$

The sentence θ_n says, "There are at least n elements." Then Φ^+ is consistent, since it has a model $\mathfrak{A}$. By Corollary 3.58, Φ^+ has a countable model, say $\mathfrak{B}$. But $\mathfrak{B}$ cannot be finite, since $\mathfrak{B} \vDash \theta_n$ for all n.

(ii) Let κ be any infinite cardinality, and let

$$X \stackrel{\text{def}}{=} \{x_\alpha \mid \alpha < \kappa\}$$

be a set of new variables. Then $\#X = \kappa$. Suppose Φ has an infinite model $\mathfrak{A}$. Let

$$\Phi^+ \stackrel{\text{def}}{=} \Phi \cup \{x_\alpha \neq x_\beta \mid \alpha < \beta < \kappa\}.$$

Note that $\#\Phi^+ = \kappa$, so we are no longer necessarily working in a countable language. Nevertheless, the technique of the completeness theorem (Theorem 3.54) still applies. The set Φ^+ is consistent, because any refutation would involve only a finite subset of Φ^+, and every such subset has a model, namely $\mathfrak{A}$. We then construct a maximal consistent extension of Φ^+ as in the proof of Theorem 3.54, the only difference here being that we need transfinite induction. We form a term algebra $\mathfrak{A}$ and valuation u such that $\mathfrak{A}, u \vDash \Phi^+$. Now $\#\mathfrak{A} \leq \kappa$, since the number of terms is at most κ; and $\#\mathfrak{A} \geq \kappa$, since $\mathfrak{A}, u \vDash x_\alpha \neq x_\beta$ for all $\alpha < \beta < \kappa$. ■

Undecidability

It is undecidable for given a sentence φ of first-order logic whether $\vDash \varphi$. In fact, the problem is Σ_1^0-complete.

THEOREM 3.60: The validity problem for first-order logic is Σ_1^0-complete.

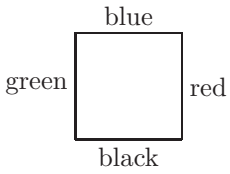
Proof That the problem is in Σ_1^0 follows from the completeness theorem. Since a formula is valid iff it has a proof, the set of valid formulas can be recursively enumerated in a uniform way simply by enumerating all proofs and checking their validity.

For Σ_1^0-hardness, we work with the complement of the validity problem, namely the *satisfiability problem*: given a first-order formula φ, is it satisfied in some model? We show that this problem is Π_1^0-hard by a reduction from the Π_1^0-complete tiling problem of Proposition 2.20: given a finite set of tile types, is there a tiling of the infinite $\omega \times \omega$ grid in which the south and west boundaries are colored blue?

We will work in a fixed first-order language consisting of one constant symbol a, one unary function symbol f, and four ternary relation symbols $\text{NORTH}(x, y, z)$, $\text{SOUTH}(x, y, z)$, $\text{EAST}(x, y, z)$, and $\text{WEST}(x, y, z)$. Intuitively, the arguments x, y will denote a grid position and z a color. The grid position $(i, j) \in \omega^2$ will be encoded by the pair $(f^i(a), f^j(a))$, where $f^0(a) \stackrel{\text{def}}{=} a$ and $f^{n+1}(a) \stackrel{\text{def}}{=} f(f^n(a))$. A color $c \in \mathbb{N}$ will be encoded by the term $f^c(a)$. Intuitively, the predicate $\text{EAST}(f^i(a), f^j(a), f^c(a))$ says, "The east edge of the tile at position i, j is colored c." Although we are thinking intuitively of an $\omega \times \omega$ grid, keep in mind that we are interpreting formulas over arbitrary structures, not just ω.

Let T be a given finite set of tile types. Each tile type is determined by the

colors of the four edges. Let C be the finite set of colors appearing in T. For each tile type $A \in T$, we can define a predicate that says that the tile at position x, y is of type A. For instance, if A is the type



we can define

$$\begin{aligned}\textsc{Tile}_A(x,y) \iff{}& \textsc{North}(x,y,f^{\text{blue}}(a)) \wedge \bigwedge_{\substack{c\in C\\ c\neq\text{blue}}} \neg\textsc{North}(x,y,f^c(a))\\ &\wedge \textsc{South}(x,y,f^{\text{black}}(a)) \wedge \bigwedge_{\substack{c\in C\\ c\neq\text{black}}} \neg\textsc{South}(x,y,f^c(a))\\ &\wedge \textsc{East}(x,y,f^{\text{red}}(a)) \wedge \bigwedge_{\substack{c\in C\\ c\neq\text{red}}} \neg\textsc{East}(x,y,f^c(a))\\ &\wedge \textsc{West}(x,y,f^{\text{green}}(a)) \wedge \bigwedge_{\substack{c\in C\\ c\neq\text{green}}} \neg\textsc{West}(x,y,f^c(a))\end{aligned}$$

This says that each of the four edges of the tile at position x, y is colored with exactly one color, and the colors correspond to the tile type A.

Let φ_T be the conjunction of the following five sentences:

$$\forall x\ \forall y\ \bigvee_{A\in T} \textsc{Tile}_A(x,y) \tag{3.4.3}$$

$$\forall x\ \textsc{South}(x,a,f^{\text{blue}}(a)) \tag{3.4.4}$$

$$\forall y\ \textsc{West}(a,y,f^{\text{blue}}(a)) \tag{3.4.5}$$

$$\forall x\ \forall y\ \bigwedge_{c\in C}(\textsc{East}(x,y,f^c(a)) \to \textsc{West}(f(x),y,f^c(a))) \tag{3.4.6}$$

$$\forall x\ \forall y\ \bigwedge_{c\in C}(\textsc{North}(x,y,f^c(a)) \to \textsc{South}(x,f(y),f^c(a))) \tag{3.4.7}$$

The predicate (3.4.3) says that every grid position is tiled with exactly one tile. The predicates (3.4.4) and (3.4.5) say that the south and west boundaries of the grid are colored blue. The predicates (3.4.6) and (3.4.7) say that the edges of adjacent tiles match.

We now argue that φ_T is satisfiable iff there exists a tiling of the grid with the given set of tile types.

First assume that $\mathfrak{A} \vDash \varphi_T$. Tile the grid as follows. Place a tile of type A at position (i, j), where A is the unique tile type such that $\mathfrak{A} \vDash \textsc{Tile}_A(f^i(a), f^j(a))$. At least one such tile type must exist, because $\mathfrak{A}$ satisfies (3.4.3); and no more than one such tile type can exist, because the definition of $\textsc{Tile}_A(f^i(a), f^j(a))$ rules out all other colorings. The remaining four clauses of φ_T assert that the local coloring conditions are satisfied by this tiling.

Conversely, suppose that the grid can be tiled with tile types from T. We can satisfy φ_T in a structure $\mathfrak{A}$ with carrier ω in which a is interpreted as 0 and f is interpreted as the successor function $x \mapsto x + 1$. The interpretation of the ternary relation symbols depends on the tiling. For example, we take $\textsc{East}(i, j, k)$ to be true if the east edge of the tile at position (i, j) has color k, and similarly for the other relation symbols. It is easy to see that φ_T holds in $\mathfrak{A}$. ■

3.5 Ehrenfeucht–Fraïssé Games

Ehrenfeucht–Fraïssé games are a technique for proving results about the expressiveness of logical languages involving quantification. There are different variations, depending on the application. Here is one:

Consider the following game between two players called the *duplicator* and the *spoiler*. The game board consists of two first-order structures $\mathfrak{A}$ and $\mathfrak{B}$. Each player is given n pebbles, one of each of n different colors.

The play alternates between the players with the spoiler going first. In each round, the spoiler places one of his pebbles on an element of either structure. The duplicator then places her pebble of the same color on an element of the other structure. The play alternates until all the pebbles have been played. If the final configuration of pebbles is a *local isomorphism*, then the duplicator wins; otherwise the spoiler wins. A configuration is a *local isomorphism* if for any atomic formula $\varphi(x_1, \ldots, x_n)$,

$$\mathfrak{A}, u \vDash \varphi(x_1, \ldots, x_n) \iff \mathfrak{B}, v \vDash \varphi(x_1, \ldots, x_n),$$

where u, v are valuations assigning to each variable x_i, $1 \leq i \leq n$, the element occupied by the pebble of color i in $\mathfrak{A}$ and $\mathfrak{B}$, respectively.

The interesting fact about this game is that the duplicator has a forced win—that is, can always assure a win for herself by playing optimally—if and only if $\mathfrak{A}$ and $\mathfrak{B}$ are indistinguishable by any first-order sentence of quantifier depth n or less. (The *quantifier depth* of a sentence is the maximum number of quantifiers in whose scope any symbol occurs.)

For example, consider the two-pebble game played on the total orders $(\mathbb{Z}, \leq)$ and $(\mathbb{Q}, \leq)$ (we ignore the algebraic structure of $\mathbb{Z}$ and $\mathbb{Q}$ and only consider their order structure). Think of the elements of these structures laid out on a line in increasing order from left to right. The duplicator can always achieve a win by the following strategy. In the first round, the spoiler plays his red pebble somewhere, then the duplicator plays her red pebble on an arbitrary element of the other structure. In the second round, if the spoiler plays his blue pebble to the left, on top, or to the right of the red pebble on either structure, then the duplicator plays her blue pebble to the left, on top, or to the right of the red pebble on the other structure, respectively. This always gives a local isomorphism, so the duplicator wins. This says that these two structures agree on any first-order sentence of quantifier depth two or less.

On the other hand, the spoiler can always win the three-pebble game on these structures by the following strategy. In the first round, he plays his red pebble on $0 \in \mathbb{Z}$. The duplicator must respond by playing her red pebble on some element $x \in \mathbb{Q}$. In the second round, the spoiler plays his blue pebble on $1 \in \mathbb{Z}$. The duplicator must respond by playing her blue pebble on some element $y \in \mathbb{Q}$, and she had better play it on some $y > x$, otherwise she loses immediately. In the third round, the spoiler plays his green pebble on some element of $\mathbb{Q}$ strictly between x and y, and the duplicator has nowhere to play on $\mathbb{Z}$ to maintain the local isomorphism. The spoiler wins.

These arguments reflect the fact that the ordered structure $(\mathbb{Q}, \leq)$ is dense, whereas $(\mathbb{Z}, \leq)$ is not. The two structures are distinguished by the sentence

$$\forall x\ \forall z\ (x < z \to \exists y\ (x < y \wedge y < z))$$

of quantifier depth three, and this is the minimum quantifier depth needed to express density.

3.6 Infinitary Logic

In some cases, we will find it convenient to allow infinite conjunctions and disjunctions of formulas; that is, formulas of form $\bigwedge_{\alpha \in A} \varphi_\alpha$ and $\bigvee_{\alpha \in A} \varphi_\alpha$, where $\{\varphi_\alpha \mid \alpha \in A\}$ is an indexed family of formulas, possibly infinite. The meaning of these formulas is just what one would expect: $\mathfrak{A}, u \vDash \bigwedge_{\alpha \in A} \varphi_\alpha$ iff for all $\alpha \in A$, $\mathfrak{A}, u \vDash \varphi_\alpha$, and $\mathfrak{A}, u \vDash \bigvee_{\alpha \in A} \varphi_\alpha$ iff for at least one $\alpha \in A$, $\mathfrak{A}, u \vDash \varphi_\alpha$.

Two particular infinitary systems that will arise in Chapter 12 and thereafter are $L_{\omega_1\omega}$ and $L_{\omega_1^{\mathrm{ck}}\omega}$. The language $L_{\omega_1\omega}$ is obtained by extending first-order logic

to allow formulas with countable conjunctions and disjunctions but only finitely many variables.

Unlike first-order logic, formulas are now possibly infinite objects. However, each formula may contain only finitely many variables. The ω_1 in $L_{\omega_1\omega}$ signifies that countable conjunctions and disjunctions are allowed, and the ω signifies the restriction to finitely many variables.

Syntax

Formally, we amend the inductive definition of formulas as follows. Let C be a fixed *finite* set of variables. The set L_C of *formulas over* C is the smallest set of formulas containing all atomic formulas all of whose variables are in C and closed under the usual closure rules for first-order logic as given Section 3.4, allowing quantification only over elements of C. In addition, we include in the inductive definition the extra clause

If $\{\varphi_\alpha \mid \alpha \in A\}$ is an indexed family of formulas of L_C and A is countable, then $\bigwedge_{\alpha\in A} \varphi_\alpha$ and $\bigvee_{\alpha\in A} \varphi_\alpha$ are formulas of L_C.

The set $L_{\omega_1\omega}$ is the union of all L_C for all finite subsets C of some fixed countable set of variables.

The language $L_{\omega_1^{\mathrm{ck}}\omega}$ is the sublanguage of $L_{\omega_1\omega}$ in which the countable conjunctions and disjunctions are further restricted to be over *recursively enumerable* sets of formulas. Thus we can form a countable conjunction $\bigwedge_{\varphi\in A} \varphi$ or disjunction $\bigvee_{\varphi\in A} \varphi$ provided the set A is r.e.

It is convenient to think of a formula of $L_{\omega_1\omega}$ as a well-founded infinitary labeled tree. Each vertex of the tree is labeled with $\forall x$, $\exists x$, $\neg$, $\bigvee$, $\bigwedge$, or an atomic formula. Vertices labeled $\forall x$, $\exists x$, or $\neg$ have one child; vertices labeled $\bigvee$ or $\bigwedge$ have countably many children; and atomic formulas label the leaves. The tree is well-founded (no infinite paths) because the definition of formulas is inductive.

Under a suitable encoding, the tree corresponding to a formula of $L_{\omega_1^{\mathrm{ck}}\omega}$ is recursively enumerable. Such trees were encountered in Section 2.2. This gives us a computational handle on infinitary formulas. For example, our encoding might represent the formula $\bigwedge_{\varphi\in A} \varphi$ as the pair of numbers $(5, i)$, where 5 indicates that the formula is a conjunction and the i is a description of a Turing machine enumerating the codes of the formulas in the r.e. set A. A universal Turing machine can then be used to enumerate the entire tree.

Another advantage of $L_{\omega_1^{\mathrm{ck}}\omega}$ over $L_{\omega_1\omega}$ is that over a countable signature, there are only countably many formulas of $L_{\omega_1^{\mathrm{ck}}\omega}$. This is not true for $L_{\omega_1\omega}$.

The language $L_{\omega_1^{\text{ck}}\omega}$ is not compact, nor does it satisfy the upward Löwenheim–Skolem theorem: the countable set

$$\{\bigvee_{n<\omega} p(f^n(a))\} \quad \cup \quad \{\neg p(f^n(a)) \mid n < \omega\}$$

is finitely satisfiable but not satisfiable (Exercise 3.35), and the sentence

$$\forall x \bigvee_{n<\omega} x = f^n(a) \tag{3.6.1}$$

has a countable model but no model of higher cardinality (Exercise 3.36). The same holds *a fortiori* for the language $L_{\omega_1\omega}$. However, the downward Löwenheim–Skolem holds, and one can give a complete infinitary deductive system for both $L_{\omega_1\omega}$ and $L_{\omega_1^{\text{ck}}\omega}$. These are the topics of the next section.

An Infinitary Deductive System

To obtain a deductive system for $L_{\omega_1\omega}$ and $L_{\omega_1^{\text{ck}}\omega}$, we augment the deductive system for first-order predicate logic given in Section 3.4 with the axioms

$$\varphi_\beta \rightarrow \bigvee_{\alpha\in A} \varphi_\alpha, \qquad \beta \in A, \tag{3.6.2}$$

$$\bigwedge_{\alpha\in A} \varphi_\alpha \rightarrow \varphi_\beta, \qquad \beta \in A, \tag{3.6.3}$$

as well as the infinitary rules of inference

$$\frac{\varphi_\alpha \rightarrow \psi,\ \alpha \in A}{(\bigvee_{\alpha\in A} \varphi_\alpha) \rightarrow \psi} \tag{3.6.4}$$

$$\frac{\varphi \rightarrow \psi_\alpha,\ \alpha \in A}{\varphi \rightarrow \bigwedge_{\alpha\in A} \psi_\alpha}. \tag{3.6.5}$$

The new rules of inference may have infinitely many premises. Thus proofs, like formulas, are no longer finite objects. However, like formulas, proofs can be represented as well-founded infinitary labeled trees, with the axioms labeling the leaves and the theorem labeling the root. Moreover, in $L_{\omega_1^{\text{ck}}\omega}$, because infinite conjunctions and disjunctions must be r.e., proof trees are r.e. as well.

Like formulas, we artificially restrict proofs to contain only finitely many variables. By definition, a proof is not a proof unless there is a finite set of variables C such that all formulas labeling the vertices of the proof tree are in L_C.

EXAMPLE 3.61: The deductive system can be used to prove infinitary versions of the basic propositional tautologies. For example, consider the infinitary De Morgan law

$$\neg \bigvee_\alpha \varphi_\alpha \quad \leftrightarrow \quad \bigwedge_\alpha \neg\varphi_\alpha. \tag{3.6.6}$$

We prove the implication in both directions using the deductive system.

($\rightarrow$) By (3.6.5), it suffices to show

$$\neg \bigvee_\alpha \varphi_\alpha \quad \rightarrow \quad \neg\varphi_\beta$$

for each β. By (finitary) propositional logic, this is equivalent to

$$\varphi_\beta \quad \rightarrow \quad \bigvee_\alpha \varphi_\alpha.$$

But this is just (3.6.2).

($\leftarrow$) By propositional logic, the implication is equivalent to

$$\bigvee_\alpha \varphi_\alpha \quad \rightarrow \quad \neg \bigwedge_\alpha \neg\varphi_\alpha.$$

By (3.6.4), it suffices to show

$$\varphi_\beta \quad \rightarrow \quad \neg \bigwedge_\alpha \neg\varphi_\alpha$$

for each β, which by propositional logic is equivalent to

$$\bigwedge_\alpha \neg\varphi_\alpha \quad \rightarrow \quad \neg\varphi_\beta.$$

But this is just an instance of (3.6.3).

EXAMPLE 3.62: For another example, consider the infinitary distributive law

$$\varphi \vee \bigwedge_\alpha \psi_\alpha \quad \leftrightarrow \quad \bigwedge_\alpha (\varphi \vee \psi_\alpha). \tag{3.6.7}$$

We prove the implication in both directions.

($\rightarrow$) By (3.6.5), it suffices to show

$$\varphi \vee \bigwedge_\alpha \psi_\alpha \quad \rightarrow \quad \varphi \vee \psi_\beta$$

for each β. This follows immediately from (3.6.3) and propositional logic.

($\leftarrow$) By propositional logic, the implication is equivalent to

$$\neg\varphi \wedge \bigwedge_\alpha (\varphi \vee \psi_\alpha) \quad \to \quad \bigwedge_\alpha \psi_\alpha. \tag{3.6.8}$$

For all β, we have

$$\begin{aligned} \neg\varphi \wedge \bigwedge_\alpha (\varphi \vee \psi_\alpha) \quad &\to \quad \neg\varphi \wedge (\varphi \vee \psi_\beta) \qquad \text{by (3.6.3)} \\ &\to \quad \psi_\beta; \end{aligned}$$

then (3.6.8) follows from (3.6.5).

Infinitary versions of other basic properties are given in Exercise 3.37.

THEOREM 3.63 (INFINITARY DEDUCTION THEOREM): For any set of formulas Φ and formulas φ, ψ of $L_{\omega_1\omega}$,

$$\Phi \cup \{\varphi\} \vdash \psi \quad \Longleftrightarrow \quad \Phi \vdash \varphi \to \psi.$$

Proof The proof is the same as for first-order logic (Theorem 3.52), except that in the direction ($\Longrightarrow$) there are extra cases for the rules (3.6.4) and (3.6.5). We argue the case (3.6.5) explicitly and leave the case (3.6.4) as an exercise (Exercise 3.38).

Suppose $\Phi \cup \{\varphi\} \vdash \psi \to \bigwedge_\alpha \psi_\alpha$ by an application of the rule (3.6.5). Then for each β, $\Phi \cup \{\varphi\} \vdash \psi \to \psi_\beta$ by a shorter proof; here "shorter" means the well-founded proof tree is shallower.[4] By the induction hypothesis, $\Phi \vdash \varphi \to (\psi \to \psi_\beta)$, and by propositional logic, $\Phi \vdash (\varphi \wedge \psi) \to \psi_\beta$. By (3.6.5), $\Phi \vdash (\varphi \wedge \psi) \to \bigwedge_\alpha \psi_\alpha$, and again by propositional logic, $\Phi \vdash \varphi \to (\psi \to \bigwedge_\alpha \psi_\alpha)$. ■

Now we show that the deductive system is complete. The proof mirrors closely that for first-order logic given in Theorem 3.52 with appropriate modifications to handle infinitary conjunctions and disjunctions.

First we note that for any formula φ of $L_{\omega_1\omega}$, the number of subformulas of φ is countable. This can be proved by induction using the fact that a countable union of countable sets is countable (Exercise 1.21). It follows that if Φ is a countable set of formulas, then the set of all subformulas of formulas in Φ and their negations is countable.

We form the sets X_n, L_n, X_ω, and L_ω as in the proof of Theorem 3.52, except that we must amend the definition slightly to ensure that the resulting set of

4 Formally, the ordinal $\mathrm{ord}(T)$ labeling the root of the proof tree T under the labeling scheme described in Section 2.2 is smaller.

formulas is countable whenever Φ is. Starting with a set of formulas L_0 over a set of variables X_0, we form L_n and X_n inductively as follows. For each φ in L_n, create a new variable $x_\varphi \in X_{n+1}$ and let

$$X_{n+1} \stackrel{\text{def}}{=} X_n \cup \{x_\varphi \mid \varphi \in L_n\}.$$

Now let L_{n+1} be the set of formulas obtained from L_n by changing any bound variable to one of the new variables in X_{n+1} (Lemma 3.48) and by substituting any term over X_{n+1} for any free variable. Thus in this construction we do not consider the set of all $L_{\omega_1\omega}$ formulas, which is uncountable, but only those that are similar to a formula of L_0 except for change of bound variable or substitution of a term for a free variable. By Exercise 1.21, if the original sets L_0 and X_0 are countable, then the resulting sets L_ω and X_ω will be countable as well. In our application, we will take L_0 to be the set of subformulas of formulas in Φ and their negations.

As in the proof of Theorem 3.52, we take Ψ to be the set of all formulas of L_ω of the form

$$\exists x\ \psi \quad \rightarrow \quad \psi[x/x_{\exists x\psi}]. \tag{3.6.9}$$

LEMMA 3.64: Let $\Phi \subseteq L_0$. If Φ is consistent, then so is $\Phi \cup \Psi$.

Proof In the proof of the corresponding theorem for first-order logic (Theorem 3.52), the first step was to observe that if $\Phi \cup \Psi$ is refutable, then there exists a finite subset $\Psi' \subseteq \Psi$ such that $\Phi \cup \Psi'$ is refutable. This was obvious there, since proofs were finite objects, therefore could refer to at most finitely many members of Ψ. Here formulas and proofs are no longer finite objects; nevertheless, the observation still holds, since proofs may contain only finitely many variables, and each formula (3.6.9) contains a distinct variable $x_{\exists x\psi}$, therefore at most finitely many of them can appear in the refutation. The remainder of the proof is the same as the proof for first-order logic (Theorem 3.52). ■

THEOREM 3.65 (COMPLETENESS): The infinitary deductive system is complete; that is, any consistent set of formulas has a model.

Proof Everything is the same as in the first-order case (Theorem 3.54), except that in the inductive argument that

$$\mathfrak{A}, u \vDash \varphi \quad \Longleftrightarrow \quad \varphi \in \widehat{\Phi},$$

we have two extra cases for infinitary join and meet. For infinitary meet, we have

$$\begin{aligned}
\mathfrak{A}, u \vDash \bigwedge_\alpha \varphi_\alpha \quad &\iff \text{for all } \beta, \quad \mathfrak{A}, u \vDash \varphi_\beta && \text{definition of } \vDash \\
&\iff \text{for all } \beta, \quad \varphi_\beta \in \widehat{\Phi} && \text{induction hypothesis} \\
&\iff \textstyle\bigwedge_\alpha \varphi_\alpha \in \widehat{\Phi} && \text{consistency and maximality of } \widehat{\Phi}.
\end{aligned}$$

The case of infinitary join is similar. ■

The Downward Löwenheim–Skolem Theorem

THEOREM 3.66 (DOWNWARD LÖWENHEIM–SKOLEM): Let Φ be a countable set of formulas of $L_{\omega_1\omega}$. If Φ has a model, then it has a countable model.

Proof If Φ has a model, then it is consistent, since the deductive system is sound. In the construction of the term model of the completeness theorem, if we restrict our attention to subformulas of formulas in Φ and their negations, the resulting model is countable. ■

Complexity

THEOREM 3.67: Deciding the validity of $L_{\omega_1^{\mathrm{ck}}\omega}$ formulas is Π^1_1-complete.

Proof The problem is in Π^1_1, because by the completeness theorem (Theorem 3.65), a formula is valid iff it has a recursively enumerable well-founded proof tree, and this is a statement of the form (2.2.3). Alternatively, one can give an explicit IND program (see Section 2.2) accepting the code of a formula of $L_{\omega_1^{\mathrm{ck}}\omega}$ iff it is provable (Exercise 3.39).

To show that the problem is Π^1_1-hard, we encode (the complement of) the tiling problem of Proposition 2.22. The construction is very similar to that of Theorem 3.60, except that we include the formula (3.6.1) to restrict to models consisting essentially of the natural numbers, as well as a formula ψ_{red} that says that red occurs only finitely often in the tiling:

$$\begin{aligned}
\mathrm{RED}(x,y) \quad &\stackrel{\mathrm{def}}{\iff} \quad \mathrm{NORTH}(x,y,f^{\mathrm{red}}(a)) \vee \mathrm{SOUTH}(x,y,f^{\mathrm{red}}(a)) \\
&\qquad\quad \vee\ \mathrm{EAST}(x,y,f^{\mathrm{red}}(a)) \vee \mathrm{WEST}(x,y,f^{\mathrm{red}}(a)) \\
\psi_{\mathrm{red}} \quad &\stackrel{\mathrm{def}}{\iff} \quad \exists x\ \forall y\ \forall z\ z \geq x \to (\neg\mathrm{RED}(y,z) \wedge \neg\mathrm{RED}(z,y)).
\end{aligned}$$

If φ is the formula constructed in the proof of Theorem 3.60, which says that we

have a valid tiling, and if ψ is the sentence (3.6.1), then the desired formula is

$$\varphi \wedge \psi \quad \rightarrow \quad \psi_{\text{red}},$$

which says that if the model represents a valid tiling of the $\omega \times \omega$ grid, then red is used only finitely often. Unlike the case of Theorem 3.60, we must include ψ here to ensure that the existential quantifiers in ψ_{red} refer to grid elements. ■

3.7 Modal Logic

Modal logic is the logic of *possibility* and *necessity*. There is not a single system of modal logic, but many different systems depending on the application. Modal logic is good for reasoning in situations involving incomplete information or dependence on time. It is also useful in applications involving knowledge, belief, and provability.

Propositional Modal Logic

Propositional logic (Section 3.2) can be extended to propositional modal logic by adding a new unary operator $\Box$, the *necessity* operator. Thus if φ is a formula, then so is $\Box\varphi$. This clause is added as part of the inductive definition of the language. There is a dual operator $\Diamond\varphi$, the *possibility* operator, defined by

$$\Diamond\varphi \quad \stackrel{\text{def}}{\Longleftrightarrow} \quad \neg\Box\neg\varphi. \tag{3.7.1}$$

The formula $\Box\varphi$ is read, "it is *necessary* that φ," or "φ holds in all possible worlds," or just "box φ." The formula $\Diamond\varphi$ is read, "it is *possible* that φ," or "there is a possible world that realizes φ," or just "diamond φ." The property (3.7.1) expresses a duality between $\Box$ and $\Diamond$; intuitively, φ is necessarily true iff it is impossible that φ is false.

Semantically, we interpret modal formulas in structures called *Kripke frames*. A *Kripke frame* is a structure $\mathfrak{K} = (K, R_{\mathfrak{K}}, \mathfrak{m}_{\mathfrak{K}})$, where K is a nonempty set, $R_{\mathfrak{K}}$ is a binary relation on K called the *accessibility relation*, and $\mathfrak{m}_{\mathfrak{K}}$ is a function assigning a subset of K to each atomic proposition. The class K is called the *universe* of $\mathfrak{K}$ and the elements of K are called *states* or *worlds*. Intuitively, $R_{\mathfrak{K}}$ specifies which worlds are *accessible* (or possible) from the point of view of a given world; that is, $(u, v) \in R_{\mathfrak{K}}$ says that v is a possible world from the point of view of u.

The function $\mathfrak{m}_{\mathfrak{K}}$ determines a truth assignment to the primitive propositions in each state; we write $u \vDash p$ if $u \in \mathfrak{m}_{\mathfrak{K}}(p)$. We extend $\mathfrak{m}_{\mathfrak{K}}$ inductively to all modal

formulas according to the following rules:

$$\mathfrak{m}_{\mathfrak{K}}(\varphi \to \psi) \stackrel{\text{def}}{=} (K - \mathfrak{m}_{\mathfrak{K}}(\varphi)) \cup \mathfrak{m}_{\mathfrak{K}}(\psi), \tag{3.7.2}$$

$$\mathfrak{m}_{\mathfrak{K}}(\mathbf{0}) \stackrel{\text{def}}{=} \varnothing, \tag{3.7.3}$$

$$\mathfrak{m}_{\mathfrak{K}}(\Box\varphi) \stackrel{\text{def}}{=} K - (R_{\mathfrak{K}} \circ (K - \mathfrak{m}_{\mathfrak{K}}(\varphi))), \tag{3.7.4}$$

where $\circ$ denotes relational composition (see Section 1.3). It follows that

$$\mathfrak{m}_{\mathfrak{K}}(\Diamond\varphi) = R_{\mathfrak{K}} \circ \mathfrak{m}_{\mathfrak{K}}(\varphi). \tag{3.7.5}$$

Writing $s \vDash \varphi$ for $s \in \mathfrak{m}_{\mathfrak{K}}(\varphi)$, we see that (3.7.2)–(3.7.5) are equivalent to

$$\begin{aligned}
u \vDash \varphi \to \psi &\iff (u \vDash \varphi \implies u \vDash \psi),\\
u &\nvDash \mathbf{0},\\
u \vDash \Box\varphi &\iff \text{for all } v, \text{ if } (u,v) \in R_{\mathfrak{K}} \text{ then } v \vDash \varphi,\\
u \vDash \Diamond\varphi &\iff \text{there exists } v \text{ such that } (u,v) \in R_{\mathfrak{K}} \text{ and } v \vDash \varphi,
\end{aligned}$$

respectively. The rules (3.7.2) and (3.7.3) are the same as in propositional logic, and (3.7.4) and (3.7.5) interpret the modalities.

We write $\mathfrak{K}, u \vDash \varphi$ if $u \vDash \varphi$ in the Kripke frame $\mathfrak{K}$, or just $u \vDash \varphi$ if $\mathfrak{K}$ is understood. We write $\mathfrak{K} \vDash \varphi$ iff $\mathfrak{K}, u \vDash \varphi$ for all states u of $\mathfrak{K}$. We write $\vDash \varphi$ if $\mathfrak{K} \vDash \varphi$ for all frames $\mathfrak{K}$ and say that φ is *valid*.

If Φ is a set of modal formulas, we write $\mathfrak{K}, u \vDash \Phi$ if $\mathfrak{K}, u \vDash \varphi$ for all $\varphi \in \Phi$, and we write $\mathfrak{K} \vDash \Phi$ if $\mathfrak{K}, u \vDash \Phi$ for all $u \in K$. If there exists a Kripke frame $\mathfrak{K}$ and state u of $\mathfrak{K}$ such that $\mathfrak{K}, u \vDash \Phi$, then we say that Φ is *satisfiable*. As in propositional and predicate logic, a formula φ is valid iff its negation is not satisfiable.

If $\varphi_1, \ldots, \varphi_n$ and φ are modal formulas, the rule of inference

$$\frac{\varphi_1, \ldots, \varphi_n}{\varphi}$$

is *sound* if $\mathfrak{K} \vDash \varphi$ whenever $\mathfrak{K} \vDash \varphi_i$, $1 \le i \le n$. Note that this is *not* the same as saying that $\mathfrak{K}, u \vDash \varphi$ whenever $\mathfrak{K}, u \vDash \varphi_i$, $1 \le i \le n$.

The modalities $\Box$ and $\Diamond$ capture various properties of our metalogic that we have been using in previous sections. The following are some examples.

EXAMPLE 3.68: Let P be a set of atomic propositions, and let $\mathfrak{K} = (K, R_{\mathfrak{K}}, \mathfrak{m}_{\mathfrak{K}})$

be the Kripke frame with

$$\begin{aligned} K &= \{\text{truth assignments to } P\}, \\ R_{\mathfrak{K}} &= K \times K, \\ \mathfrak{m}_{\mathfrak{K}}(p) &= \{u \mid u(p) = \mathbf{1}\}. \end{aligned}$$

Then a propositional formula φ is a tautology iff $\mathfrak{K} \vDash \Box\varphi$, and φ is satisfiable iff $\mathfrak{K} \vDash \Diamond\varphi$.

EXAMPLE 3.69: Let $(P, <)$ be a strict partial order with bottom element 0. In a Kripke frame $\mathfrak{K}$ with states P and accessibility relation $<$,

- $\mathfrak{K}, a \vDash \Box\mathbf{0}$ if a is a maximal element of P;
- $\mathfrak{K}, 0 \vDash \Diamond\Box\mathbf{0}$ iff P contains a maximal element;
- $\mathfrak{K}, 0 \vDash \Box\Diamond\Box\mathbf{0}$ iff every element is below a maximal element.

THEOREM 3.70: The following are valid formulas of propositional modal logic:

(i) $\Diamond(\varphi \vee \psi) \quad \leftrightarrow \quad \Diamond\varphi \vee \Diamond\psi$

(ii) $\Box(\varphi \wedge \psi) \quad \leftrightarrow \quad \Box\varphi \wedge \Box\psi$

(iii) $\Diamond\varphi \wedge \Box\psi \quad \rightarrow \quad \Diamond(\varphi \wedge \psi)$

(iv) $\Box(\varphi \rightarrow \psi) \quad \rightarrow \quad (\Box\varphi \rightarrow \Box\psi)$

(v) $\Diamond(\varphi \wedge \psi) \quad \rightarrow \quad \Diamond\varphi \wedge \Diamond\psi$

(vi) $\Box\varphi \vee \Box\psi \quad \rightarrow \quad \Box(\varphi \vee \psi)$

(vii) $\Diamond\mathbf{0} \quad \leftrightarrow \quad \mathbf{0}$

(viii) $\Box\varphi \quad \leftrightarrow \quad \neg\Diamond\neg\varphi$.

Proof These results are straightforward exercises in relational algebra. We prove (i) explicitly and leave the rest as exercises (Exercise 3.40).

To prove (i), we must show that for any Kripke frame $\mathfrak{K} = (K, R_{\mathfrak{K}}, \mathfrak{m}_{\mathfrak{K}})$,

$$\mathfrak{m}_{\mathfrak{K}}(\Diamond(\varphi \vee \psi)) \quad = \quad \mathfrak{m}_{\mathfrak{K}}(\Diamond\varphi \vee \Diamond\psi).$$

But

$\mathfrak{m}_{\mathfrak{K}}(\Diamond(\varphi \vee \psi))$

$=$	$R \circ \mathfrak{m}_{\mathfrak{K}}(\varphi \vee \psi)$	semantics of $\Diamond$
$=$	$R \circ (\mathfrak{m}_{\mathfrak{K}}(\varphi) \cup \mathfrak{m}_{\mathfrak{K}}(\psi))$	semantics of proposition logic
$=$	$(R \circ \mathfrak{m}_{\mathfrak{K}}(\varphi)) \cup (R \circ \mathfrak{m}_{\mathfrak{K}}(\psi))$	Lemma 1.1
$=$	$\mathfrak{m}_{\mathfrak{K}}(\Diamond\varphi) \cup \mathfrak{m}_{\mathfrak{K}}(\Diamond\psi)$	semantics of $\Diamond$
$=$	$\mathfrak{m}_{\mathfrak{K}}(\Diamond\varphi \vee \Diamond\psi)$	semantics of proposition logic.

■

THEOREM 3.71: The following rules are sound:

(i) Modal generalization (GEN):

$$\frac{\varphi}{\Box\varphi}$$

(ii) Monotonicity of $\Diamond$:

$$\frac{\varphi \to \psi}{\Diamond\varphi \to \Diamond\psi}$$

(iii) Monotonicity of $\Box$:

$$\frac{\varphi \to \psi}{\Box\varphi \to \Box\psi}.$$

Proof Let $\mathfrak{K} = (K, R_{\mathfrak{K}}, \mathfrak{m}_{\mathfrak{K}})$ be a Kripke frame.

(i) If $\mathfrak{m}_{\mathfrak{K}}(\varphi) = K$, then $R_{\mathfrak{K}} \circ (K - \mathfrak{m}_{\mathfrak{K}}(\varphi)) = \varnothing$, therefore $K - (R_{\mathfrak{K}} \circ (K - \mathfrak{m}_{\mathfrak{K}}(\varphi))) = K$.

(ii) By monotonicity of $\circ$ (Exercise 1.3), if $\mathfrak{m}_{\mathfrak{K}}(\varphi) \subseteq \mathfrak{m}_{\mathfrak{K}}(\psi)$, then $R_{\mathfrak{K}} \circ \mathfrak{m}_{\mathfrak{K}}(\varphi) \subseteq R_{\mathfrak{K}} \circ \mathfrak{m}_{\mathfrak{K}}(\psi)$.

(iii) If $\mathfrak{m}_{\mathfrak{K}}(\varphi) \subseteq \mathfrak{m}_{\mathfrak{K}}(\psi)$, then $(K - \mathfrak{m}_{\mathfrak{K}}(\psi)) \subseteq (K - \mathfrak{m}_{\mathfrak{K}}(\varphi))$. By (ii), we have $R_{\mathfrak{K}} \circ (K - \mathfrak{m}_{\mathfrak{K}}(\psi)) \subseteq R_{\mathfrak{K}} \circ (K - \mathfrak{m}_{\mathfrak{K}}(\varphi))$, therefore $K - (R_{\mathfrak{K}} \circ (K - \mathfrak{m}_{\mathfrak{K}}(\varphi))) \subseteq K - (R_{\mathfrak{K}} \circ (K - \mathfrak{m}_{\mathfrak{K}}(\psi)))$. ■

Multimodal Logic

More generally, let $A = \{a, \ldots\}$ be a set of *modalities*. Instead of augmenting propositional logic with one modality as in Section 3.7, we can augment it with a separate modality for each $a \in A$. We add to the inductive definition of formulas

the clause:

- If φ is a formula and $a \in A$, then $\texttt{[}a\texttt{]}\varphi$ is a formula.

We also define

$$\texttt{<}a\texttt{>}\varphi \stackrel{\text{def}}{=} \neg\texttt{[}a\texttt{]}\neg\varphi.$$

A *Kripke frame* is now a structure $\mathfrak{K} = (K, \mathfrak{m}_{\mathfrak{K}})$, where the map $\mathfrak{m}_{\mathfrak{K}}$, in addition to interpreting the atomic propositions as described in Section 3.7, associates a binary relation $\mathfrak{m}_{\mathfrak{K}}(a) \subseteq K \times K$ to each modality $a \in A$. The semantics of $\texttt{[}a\texttt{]}\varphi$ and $\texttt{<}a\texttt{>}\varphi$ is defined as for $\Box\varphi$ and $\Diamond\varphi$, respectively, with $\mathfrak{m}_{\mathfrak{K}}(a)$ taking the place of $R_{\mathfrak{K}}$.

Example 3.72: Consider a propositional logic whose atomic propositions are the atomic formulas $p(t_1, \ldots, t_n)$ of predicate logic over a signature Σ and a countable set X of first-order variables. Let $\mathfrak{A}$ be a first-order structure of signature Σ. The structure $\mathfrak{A}$ gives rise to a multimodal Kripke frame $(K, \mathfrak{m}_{\mathfrak{A}})$ with modalities X defined as follows:

$$\begin{aligned} K &\stackrel{\text{def}}{=} \{\text{valuations } u : T_\Sigma(X) \to |\mathfrak{A}|\}, \\ \mathfrak{m}_{\mathfrak{A}}(p(t_1, \ldots, t_n)) &\stackrel{\text{def}}{=} \{u \mid \mathfrak{A}, u \vDash p(t_1, \ldots, t_n)\}, \\ \mathfrak{m}_{\mathfrak{A}}(x) &\stackrel{\text{def}}{=} \{(u, v) \mid u(y) = v(y),\ y \neq x\}. \end{aligned}$$

That is, $\mathfrak{m}_{\mathfrak{A}}(x)$ is a symmetric relation connecting any pair of valuations over $\mathfrak{A}$ that agree on all variables except possibly x. For any quantifier-free formula φ and $u \in K$, $\mathfrak{A}, u \vDash \texttt{[}x\texttt{]}\varphi$ iff $\mathfrak{A}, u \vDash \forall x\ \varphi$ in the usual sense of predicate logic as defined in Section 3.4, and $\mathfrak{A}, u \vDash \texttt{<}x\texttt{>}\varphi$ iff $\mathfrak{A}, u \vDash \exists x\ \varphi$ in the usual sense of predicate logic. More generally, if φ is a first-order formula and φ' is obtained from φ by changing all $\forall x$ to $\texttt{[}x\texttt{]}$ and all $\exists x$ to $\texttt{<}x\texttt{>}$, then $\mathfrak{A}, u \vDash \varphi'$ in the modal sense iff $\mathfrak{A}, u \vDash \varphi$ in the usual sense of predicate logic.

Example 3.73: Consider a finite-state automaton with states Q, start state $s \in Q$, accept states $F \subseteq Q$, and input alphabet Σ. Let the set of modalities be Σ^*. Let there be a single atomic formula f satisfied by all and only the states in F. Let $\mathfrak{M} = (Q, \mathfrak{m}_{\mathfrak{M}})$, where

$$\mathfrak{m}_{\mathfrak{M}}(w) \stackrel{\text{def}}{=} \{(p, q) \mid p, q \in Q,\ q \text{ is reachable from } p \text{ under input string } w\}.$$

Then $\mathfrak{M}$ accepts w iff $\mathfrak{M}, s \vDash \texttt{<}w\texttt{>}f$.

Unwinding

For a multimodal logic with modalities A, one can without loss of generality restrict attention to models that resemble trees. Any Kripke frame can be "unwound" into an equivalent treelike structure. By *equivalent* we mean that the two structures cannot be distinguished by any modal formula.

Given a Kripke frame $\mathfrak{K} = (K, \mathfrak{m}_{\mathfrak{K}})$ and $s \in K$, we construct an equivalent treelike model $\mathfrak{K}'$ whose states are the paths in $\mathfrak{K}$ out of s. Formally, a *path* in $\mathfrak{K}$ is a finite sequence $\sigma = s_0 a_0 s_1 a_1 s_2 a_2 \cdots a_{n-1} s_n$ of alternating states of $\mathfrak{K}$ and modalities, beginning and ending with a state, such that $(s_i, s_{i+1}) \in \mathfrak{m}_{\mathfrak{K}}(a_i)$, $0 \leq i < n$. For a path σ, let $\mathrm{first}(\sigma)$ and $\mathrm{last}(\sigma)$ denote the first and last states of σ, respectively. We take the states K' of $\mathfrak{K}'$ to be the set of all paths σ in $\mathfrak{K}$ with $\mathrm{first}(\sigma) = s$. The modalities are interpreted in $\mathfrak{K}'$ as

$$\mathfrak{m}_{\mathfrak{K}'}(a) \stackrel{\text{def}}{=} \{(\sigma, \sigma a t) \mid (\mathrm{last}(\sigma), t) \in \mathfrak{m}_{\mathfrak{K}}(a)\}.$$

For the atomic propositions, we define

$$\mathfrak{m}_{\mathfrak{K}'}(p) \stackrel{\text{def}}{=} \{\sigma \mid \mathrm{last}(\sigma) \in \mathfrak{m}_{\mathfrak{K}}(p)\}.$$

Then $\mathfrak{K}'$ is a tree with root s. Moreover, the states s in the two models are indistinguishable by any modal formula:

THEOREM 3.74: For any propositional modal formula φ and any path σ in $\mathfrak{K}$,

$$\mathfrak{K}', \sigma \vDash \varphi \iff \mathfrak{K}, \mathrm{last}(\sigma) \vDash \varphi.$$

In particular,

$$\mathfrak{K}', s \vDash \varphi \iff \mathfrak{K}, s \vDash \varphi.$$

Proof The second statement is the special case of the first with $\sigma = s$. The first statement is proved by induction on the structure of φ and is left as an exercise (Exercise 3.41). ■

A useful corollary of this result is that every satisfiable formula is satisfied in a countable frame; that is, one with only countably many states. In fact, one can show that every satisfiable formula is satisfied in a tree model in which each state has only finitely many successors (Exercise 3.42). For propositional modal logic, one can show an even stronger result: every satisfiable formula is satisfied in a finite frame. We will prove a generalization of this result in Chapter 6 in the context of Propositional Dynamic Logic.

Modal Logic and Programs

Modal logic is particularly well suited for reasoning in *dynamic* situations—situations in which the truth values of statements are not fixed, but may vary over time. Classical first-order logic is *static*, in the sense that the truth values of its statements are immutable.

Sentences of classical first-order logic are interpreted over a single structure, or *world*. In modal logic, an interpretation consists of a collection K of many possible worlds or states. If states can change somehow, then so can truth values.

One successful dynamic interpretation of modal logic is *temporal logic*. In this approach, a state t is accessible from s if t lies in the future of s. The accessibility relation is sometimes taken to be a linear ordering of K (linear-time temporal logic) or a tree (branching-time temporal logic). We will have more to say about temporal logic in Section 17.2.

These ideas also fit nicely into the framework of program execution. We can take the set of states K to be the universe of all possible execution states of a program. With any program α, one can associate a binary accessibility relation over K such that (s, t) is in this relation iff t is a possible final state of the program α with initial state s; that is, iff there is a computation of α starting in s and terminating in t. We say "possible" here since we might wish to consider *nondeterministic programs*, which can have more than a single final state associated with a given initial one.

Syntactically, each program gives rise to a modality of a multimodal logic. We place the program α inside the modality symbol: $[\alpha]$, $\langle\alpha\rangle$. Thus programs become an explicit part of the language. The expression $\langle\alpha\rangle\varphi$ says that it is possible to execute α and halt in a state satisfying φ; the expression $[\alpha]\varphi$ says that whenever α halts, it does so in a state satisfying φ. The resulting system is called Dynamic Logic (DL). Since the inductive definition of formulas allows arbitrary prefixes of modal operators, the syntax is more flexible and expressive than the partial correctness assertions of Hoare Logic. For example, if $\langle\alpha\rangle\varphi$ and $\langle\beta\rangle\varphi$ are logically equivalent, then for every initial state s the program α can terminate in a state satisfying φ iff β can.

Dynamic Logic is not limited merely to augmenting classical logic with a fixed modality for each program; this would be little more than multimodal logic. Rather, it uses various calculi of programs, which in conjunction with the rules of classical propositional and predicate logic give a rich family of systems for analyzing the interaction of programs and formulas. By analogy with the construction of composite formulas from atomic ones, the calculi of programs allow the construction of complex programs from atomic ones. Typical atomic programs are assignment

statements and basic tests; the operators used to construct composite programs may be familiar programming constructs such as **if-then-else** and **while-do**. There are rules for analyzing the behavior of programs in terms of the behavior of their subprograms, as well as for analyzing the interaction of programs and formulas. The resulting framework gives a powerful set of tools for understanding the relative power and complexity of programming constructs. It constitutes the subject matter of the remainder of this book.

3.8 Bibliographical Notes

Most of the topics discussed in this chapter are classical. Good introductory sources are Kleene (1952); Bell and Slomson (1971); Chang and Keisler (1973); van Dalen (1994) (propositional and predicate logic), Grätzer (1978) (equational logic and universal algebra), Keisler (1971) (infinitary logic), and Emerson (1990); Chellas (1980); Hughes and Cresswell (1968) (modal logic).

Logic has been with us since the time of Aristotle. Perhaps the first person to view logic as mathematics was Boole (1847). Logic came to the fore as a foundation of mathematics around the beginning of the twentieth century as part of a trend toward increased rigor in mathematical arguments. This movement was championed by Whitehead and Russell (1913) and by Hilbert. The model-theoretic approach to logic owes much to the work of Tarski (1935). The relationship between propositional logic and set theory was observed by Stone (1936).

There are many proofs of the completeness of first-order predicate logic. The first such proof was given by Gödel (1930). The approach here is due to Henkin (1949).

The HSP Theorem (Theorem 3.42 and Corollary 3.43) is due to Birkhoff (1935). Ehrenfeucht–Fraïssé games were introduced in Ehrenfeucht (1961). Kripke (1963) developed the semantics of modal logic.

Exercises

3.1. Prove Theorem 3.4.

3.2. Let S_n be the set of all truth assignments to atomic propositional symbols $p_1, \ldots, p_n$. Elements of S_n are maps $u : \{p_1, \ldots, p_n\} \to \{\mathbf{0}, \mathbf{1}\}$ and $\#S_n = 2^n$. A *truth table* over $p_1, \ldots, p_n$ is a function $T : S_n \to \{\mathbf{0}, \mathbf{1}\}$. There are 2^{2^n} truth tables over $p_1, \ldots, p_n$.

Every propositional formula φ determines a truth table T_φ:

$$T_\varphi(u) \stackrel{\text{def}}{=} u(\varphi),$$

where the u on the right-hand side is the inductive extension of the truth assignment u to all formulas over $p_1, \ldots, p_n$ as defined in Section 3.2. An interesting question is the converse: is it true that for every truth table T there is a corresponding propositional formula φ such that $T = T_\varphi$? Show that this is so.

3.3. Truth tables and the notation T_φ were defined in Exercise 3.2. A set F of propositional operators is *complete* if for every n and every truth table T over $p_1, \ldots, p_n$ there is a propositional formula φ over $p_1, \ldots, p_n$ and F only such that $T = T_\varphi$. As shown in Exercise 3.2, the set $\{\mathbf{0}, \rightarrow\}$ is complete.

(a) Show that the sets $\{\wedge, \neg\}$ and $\{\vee, \neg\}$ are also complete.

(b) Show that none of the operators $\wedge, \vee, \neg, \mathbf{0}, \mathbf{1}, \rightarrow, \leftrightarrow$ by themselves are complete.

(c) Show that $\{\leftrightarrow, \neg, \mathbf{0}, \mathbf{1}\}$ is not complete. (*Hint.* Show by induction that for any truth assignment to a formula φ built from these connectives alone, if the truth value of p is changed, then the truth value of φ changes iff p has an odd number of occurrences in φ.)

(d) Show that $\{\vee, \wedge\}$ is not complete. Formulas built from the connectives $\vee, \wedge$ only are called *monotone*.

(e) Show that $\{\vee, \wedge, \rightarrow\}$ is not complete. (*Hint.* Consider the truth assignment that assigns $\mathbf{0}$ to every atomic proposition.)

(f) Define a single propositional operator that is complete. Specify the operator by giving its truth table (see Exercise 3.2). Prove that it is complete.

3.4. In this exercise we develop a useful duality principle for formulas expressed over the propositional connectives $\wedge$, $\vee$, and $\neg$. For any such propositional formula φ, define its *dual* φ' inductively as follows:

- $p' = p$ for atomic propositions p,
- $(\varphi \wedge \psi)' = \varphi' \vee \psi'$,
- $(\varphi \vee \psi)' = \varphi' \wedge \psi'$,
- $(\neg\varphi)' = \neg\varphi'$.

In other words, we just change all occurrences of $\vee$ to $\wedge$ and vice versa. Note that $\varphi'' = \varphi$.

(a) Considering $\varphi \to \psi$ as an abbreviation for $\neg\varphi \vee \psi$, $\mathbf{0}$ as an abbreviation for $p \wedge \neg p$ (where p is an arbitrary atomic proposition), $\mathbf{1}$ as an abbreviation for $p \vee \neg p$, $\varphi \leftrightarrow \psi$ as an abbreviation for $(\varphi \to \psi) \wedge (\psi \to \varphi)$, and $\varphi \oplus \psi$ as an abbreviation for $(\neg\varphi \wedge \psi) \vee (\neg\psi \wedge \varphi)$, show that

- $\mathbf{0}' = \mathbf{1}$ and $\mathbf{1}' = \mathbf{0}$,
- $(\varphi \leftrightarrow \psi)' = \varphi' \oplus \psi'$ and $(\varphi \oplus \psi)' = \varphi' \leftrightarrow \psi'$.

(b) Let φ be a propositional formula. Let $\overline{\varphi}$ denote the formula obtained by replacing all atomic propositions by their negations; that is, if all of the atomic propositions of φ are among $p_1, \ldots, p_n$, then $\overline{\varphi} = \varphi[p_1/\neg p_1, \ldots, p_n/\neg p_n]$. Prove that φ' and $\neg\overline{\varphi}$ are propositionally equivalent. (*Hint.* Prove this by induction on the structure of φ using Axioms 3.13(ii) and (iii)).

(c) Show that φ is satisfiable iff φ' is valid.

(d) Show that $\varphi \equiv \psi$ iff $\varphi' \equiv \psi'$.

(e) Formulate and prove a generalization of these duality results for predicate logic using (3.4.1).

(f) Formulate and prove a generalization of these duality results for modal logic using (3.7.1).

3.5. Intuitionistic propositional logic was defined in Section 3.2. Show that the following propositions are intuitionistically equivalent:

(i) law of double negation: $\neg\neg\varphi \to \varphi$;

(ii) reductio ad absurdum: $(\neg\varphi \to \mathbf{0}) \to \varphi$;

(iii) law of the excluded middle: $\neg\varphi \vee \varphi$;

(iv) law of contraposition: $(\neg\psi \to \neg\varphi) \to (\varphi \to \psi)$;

(v) Peirce's law: $((\varphi \to \psi) \to \varphi) \to \varphi$.

3.6. Prove the validity of axioms (ii)–(iv) and (vi)–(ix) of Axiom System 3.13.

3.7. Prove that the free Σ-algebra generated by a set of a given cardinality is unique up to isomorphism.

3.8. A *Boolean algebra* is a structure

$$\mathfrak{B} \quad = \quad (B, \wedge, \vee, \neg, 0, 1)$$

satisfying the following equations:

$$\begin{array}{rcl} (x \vee y) \vee z & = & x \vee (y \vee z) \\ x \vee y & = & y \vee x \\ x \vee (y \wedge z) & = & (x \vee y) \wedge (x \vee z) \\ x \vee 0 & = & x \\ x \vee 1 & = & 1 \\ x \vee \neg x & = & 1 \\ x \vee x & = & x \end{array} \qquad \begin{array}{rcl} (x \wedge y) \wedge z & = & x \wedge (y \wedge z) \\ x \wedge y & = & y \wedge x \\ x \wedge (y \vee z) & = & (x \wedge y) \vee (x \wedge z) \\ x \wedge 1 & = & x \\ x \wedge 0 & = & 0 \\ x \wedge \neg x & = & 0 \\ x \wedge x & = & x. \end{array}$$

For example, a *Boolean algebra of sets* is a structure

$$\mathfrak{B} \quad = \quad (B, \cap, \cup, \sim, \varnothing, S)$$

where S is a set, B is a collection of subsets of S, $\cap$ is set intersection, $\cup$ is set union, $\sim$ is set complementation in S, and $\varnothing$ is the empty set.

(a) Show that in any Boolean algebra $\mathfrak{B}$, for any a, $b \in \mathfrak{B}$, $a \vee b = b$ iff $a \wedge b = a$. (*Hint.* Prove first the equations $x \vee (x \wedge y) = x$ and $x \wedge (x \vee y) = x$.)

(b) Prove the *De Morgan laws*

$$\begin{array}{rcl} \neg(x \vee y) & = & \neg x \wedge \neg y \\ \neg(x \wedge y) & = & \neg x \vee \neg y \end{array}$$

and the double-negation law

$$\neg\neg x \quad = \quad x.$$

(*Hint.* Use (a) to show that for all $a, b \in \mathfrak{B}$, if $a \wedge b = 0$ and $a \vee b = 1$, then $a = \neg b$.)

3.9. Let A be a set of propositional letters and let T denote the set of propositions over A and $\wedge, \vee, \neg$. For $x, y \in T$, define $\varphi \equiv \psi$ if $\varphi \leftrightarrow \psi$ is a propositional tautology. Prove that $T/\equiv$ is the free Boolean algebra on $\#A$ generators. (*Hint.* Consider the set of Boolean functions on n inputs $f : \{0,1\}^n \to \{0,1\}$.)

3.10. For finite n, how many elements does the free Boolean algebra on n generators have?

3.11. Define $a \leq b$ in a Boolean algebra if $a \wedge b = a$ (equivalently, by Exercise 3.8(a), if $a \vee b = b$). Prove that $\leq$ is a partial order with bottom 0 and top 1.

3.12. A *filter* F on a Boolean algebra $\mathfrak{B}$ is a nonempty subset of $\mathfrak{B}$ such that

$$a \in F,\ b \in F \implies a \wedge b \in F$$
$$a \in F,\ a \leq b \implies b \in F.$$

Find a natural one-to-one correspondence between filters on $\mathfrak{B}$ and congruences on $\mathfrak{B}$. (*Hint.* If $h : \mathfrak{A} \to \mathfrak{B}$ is a homomorphism, consider the set $h^{-1}(1) = \{a \in \mathfrak{A} \mid h(a) = 1^{\mathfrak{B}}\}$.)

3.13. An *ideal* I on a Boolean algebra $\mathfrak{B}$ is a nonempty subset of $\mathfrak{B}$ such that

$$a \in I,\ b \in I \implies a \vee b \in I$$
$$b \in I,\ a \leq b \implies a \in I.$$

Find a natural one-to-one correspondence between filters and ideals on $\mathfrak{B}$ (see Exercise 3.12). State the relationship between ideals and congruences analogous to the hint for Exercise 3.12.

3.14. A filter is *consistent* if $0 \notin F$. An *ultrafilter* is a maximal consistent filter; that is, one that is not properly included in any consistent filter. Show that every consistent filter is contained in an ultrafilter. (*Hint.* Show first that if F is a consistent filter and $a \in \mathfrak{B}$, then either $F(a)$ of $F(\neg a)$ is a consistent filter, where $F(x) = \{y \in \mathfrak{B} \mid \exists z \in F\ x \wedge z \leq y\}$ is the smallest filter containing F and x. Then use Zorn's lemma (see Section 1.6)).

3.15. Show that every Boolean algebra is isomorphic to a Boolean algebra of sets. (*Hint.* Given $\mathfrak{B} = (B, \wedge, \vee, \neg, 0, 1)$, take

$$S \stackrel{\text{def}}{=} \{\text{ultrafilters of } \mathfrak{B}\},$$
$$a' \stackrel{\text{def}}{=} \{F \in S \mid a \in F\},$$
$$B' \stackrel{\text{def}}{=} \{a' \mid a \in B\},$$
$$\mathfrak{B}' \stackrel{\text{def}}{=} (B', \cap, \cup, \sim, \varnothing, S),$$

where $\sim$ denotes complementation in S.)

3.16. The rule of congruence is a special case of the following *substitution rule*:

$$\frac{s_i = t_i,\ 1 \leq i \leq n}{t(s_1, \ldots, s_n) = t(t_1, \ldots, t_n)}$$

where $t(x_1, \ldots, x_n)$ is a term and $t(t_1, \ldots, t_n)$ denotes the result of simultaneously replacing all occurrences of x_i in t with t_i, $1 \leq i \leq n$. Show how to derive this rule from the other rules of equality.

3.17. Prove Theorem 3.34(i).

3.18. Prove Theorem 3.35. (*Hint.* First establish that the map $h \mapsto \ker h$ is invertible up to $\approx$. The inverse operation takes a congruence $\equiv$ on $\mathfrak{A}$ to the canonical epimorphism $[\,] : \mathfrak{A} \to \mathfrak{A}/\equiv$. To verify that the correspondence preserves the lattice structure, argue that for epimorphisms $h_1 : \mathfrak{A} \to \mathfrak{B}_1$ and $h_2 : \mathfrak{A} \to \mathfrak{B}_2$, there exists an epimorphism $g : \mathfrak{B}_1 \to \mathfrak{B}_2$ such that $h_2 = h_1 \circ g$ iff $\ker h_1$ refines $\ker h_2$.)

3.19. Consider the class of all epimorphisms with domain $\mathfrak{A}$. For two such epimorphisms $h_1 : \mathfrak{A} \to \mathfrak{B}_1$ and $h_2 : \mathfrak{A} \to \mathfrak{B}_2$, write $h_1 \leq h_2$ if there exists an epimorphism $g : \mathfrak{B}_1 \to \mathfrak{B}_2$ such that $h_2 = h_1 \circ g$, and $h_1 \approx h_2$ if both $h_1 \leq h_2$ and $h_2 \leq h_1$. Show that $h_1 \approx h_2$ iff there is an isomorphism $\iota : \mathfrak{B}_1 \to \mathfrak{B}_2$ such that $h_2 = h_1 \circ \iota$.

3.20. Prove Theorem 3.35.

3.21. (a) Prove that the maps **Mod** and **Th** defined in Section 3.3 form a Galois connection (see Exercise 1.24). Conclude that for any set of equational formulas Φ, $\mathbf{Mod}\,\Phi = \mathbf{Mod}\,\mathbf{Th}\,\mathbf{Mod}\,\Phi$, as required in the proof of Corollary 3.43.

(b) By Exercise 1.24, the maps $\mathbf{Mod} \circ \mathbf{Th}$ and $\mathbf{Th} \circ \mathbf{Mod}$ are closure operators. What are their closed sets?

3.22. Prove that every homomorphism factors into a composition of a monomorphism and an epimorphism. In other words, for every homomorphism $f : \mathfrak{A} \to \mathfrak{C}$, there exist an intermediate algebra $\mathfrak{B}$, an epimorphism $g : \mathfrak{A} \to \mathfrak{B}$, and a monomorphism $h : \mathfrak{B} \to \mathfrak{C}$ such that $f = g \circ h$.

3.23. Prove that $0x = 0$ and $x0 = 0$ are logical consequences of the axioms for rings (see Example 3.24).

3.24. In Section 1.5, semilattices were defined as partial orders in which every finite set of elements has a join (lease upper bound). Show that semilattices form a variety over the signature $\vee$ (join) and $\bot$ (least element of the semilattice). (*Hint.* Consider $x \leq y$ an abbreviation for $x \vee y = y$. Your axiomatization must ensure that $\leq$ is a

partial order, that $\bot$ is the $\leq$-least element of the structure, and that $\vee$ gives the least upper bound of two elements.)

3.25. Extend Axiom System 3.40 to handle Horn formulas. Prove that your deductive system is sound and complete.

3.26. Define a *quasivariety* to be a class of models defined by infinitary Horn formulas of the form

$$\Phi \quad \rightarrow \quad s = t,$$

where Φ is a possibly infinite set of equations. Prove the following variant of Birkhoff's theorem (Corollary 3.43). Let $\mathcal{D}$ be a class of Σ-algebras. The following are equivalent:

(i) $\mathcal{D}$ is a quasivariety
(ii) $\mathcal{D} = \mathbf{S}\,\mathbf{P}\,\mathcal{D}$
(iii) $\mathcal{D} = \{\mathbf{S}, \mathbf{P}\}^*\,\mathcal{D}$.

(*Hint.* Define the infinitary Horn theory of a class of algebras $\mathcal{D}$. Formulate and prove a theorem similar to Theorem 3.42 for infinitary Horn theories. In the last part of the proof, modify the definition of $\mathfrak{B}_{s,t}$ as follows. Let B be a set of generators of $\mathfrak{A}$ and let Δ be the kernel of the unique homomorphism $T_\Sigma(B) \rightarrow \mathfrak{A}$ extending the identity on B. Define $\mathfrak{B}_{s,t}$ and $u_{s,t}$ such that

$$\mathfrak{B}_{s,t}, u_{s,t} \quad \not\models \quad \Delta \rightarrow s = t$$

whenever $\Delta \rightarrow s = t$ is not in the Horn theory of $\mathcal{D}$.)

3.27. Let x, y, z be first-order variables ranging over $\mathbb{N}$. Show how to express the following predicate in the language of first-order number theory (see Example 3.44): "At least one of y and z is nonzero, and x is their greatest common divisor."

3.28. Prove that the following first-order formulas are valid:

$$\begin{aligned} \exists x\ \varphi \vee \exists x\ \psi \quad &\leftrightarrow \quad \exists x\ (\varphi \vee \psi) \\ \varphi \quad &\leftrightarrow \quad \exists x\ \varphi, \quad x \text{ not free in } \varphi \end{aligned}$$

Show by example that the proviso "x not free in φ" is necessary in the second formula.

3.29. Show that if valuations u and v agree on all variables occurring free in φ, then

$$\mathfrak{A}, u \vDash \varphi \iff \mathfrak{A}, v \vDash \varphi.$$

Conclude that if φ is a sentence, that is, if φ has no free variables, then $\vDash$ does not depend on the valuation u; that is, if $\mathfrak{A}, u \vDash \varphi$ for some u, then $\mathfrak{A}, u \vDash \varphi$ for all u.

3.30. Lemma 3.48 gives conditions under which bound variables can be renamed, but the proof in the text establishes the equivalence of the two formulas by a semantic argument. Show that if the conditions of Lemma 3.48 hold, then the same equivalence can be derived using Axiom System 3.51.

3.31. Prove Lemma 3.49.

3.32. Prove the soundness of Axiom System 3.51.

3.33. A second-order number-theoretic formula in prenex form is *universal* if all second-order quantifiers are universal quantifiers; that is, if it is of the form

$$\forall f_1\ \overline{\mathsf{Q}}_1\overline{y}_1\ \ldots\ \forall f_n\ \overline{\mathsf{Q}}_n\overline{y}_n\ \varphi, \tag{3.8.1}$$

where each f_i ranges over functions $\mathbb{N}^{k_i} \to \mathbb{N}$ for some k_i, the $\overline{\mathsf{Q}}_i\overline{y}_i$ are blocks of arbitrary first order quantifiers over individual variables y_{ij} ranging over $\mathbb{N}$, and φ is quantifier-free. In the proof of Theorem 2.12, we needed to know that every universal second-order number-theoretic formula can be transformed to an equivalent formula of the form

$$\forall f\ \exists y\ \varphi, \tag{3.8.2}$$

where f is a single function variable ranging over functions $\mathbb{N} \to \mathbb{N}$, y is a single individual variable ranging over $\mathbb{N}$, and φ is quantifier-free. In this exercise we establish this normal form.

(a) Give rules for second-order quantifiers analogous to the rules of Lemma 3.49 for first-order quantifiers. State a theorem analogous to Lemma 3.50 for second-order formulas.

(b) Show that the formula $\forall y\ \psi$, where y is an individual variable, is equivalent to the formula $\forall g\ \psi[y/g(0)]$, where g is a function variable of type $\mathbb{N} \to \mathbb{N}$. Conclude that (3.8.1) can be transformed into an equivalent second-order universal formula containing no universally quantified individual variables.

(c) By (a), we can assume without loss of generality that the first-order quantifier blocks $\overline{\mathsf{Q}}_i\overline{y}_i$ in (3.8.1) contain only existential quantifiers. Argue that the formula

$$\forall y\ \exists f\ \psi,$$

where y is an individual variable and f is a function variable of type $\mathbb{N}^k \to \mathbb{N}$, is equivalent to the formula

$$\exists g\ \forall y\ \psi[f/g(y)],$$

where g is a function variable of type $\mathbb{N} \to (\mathbb{N}^k \to \mathbb{N})$. This transformation is called *Skolemization*, and the function g is called a *Skolem function*.

(d) Using the transformation of (b) and currying the resulting Skolem functions (see Exercise 1.19), argue that (3.8.1) can be transformed to an equivalent formula of the form

$$\forall f_1\ \ldots\ \forall f_m\ \exists y_1\ \ldots\ \exists y_n\ \varphi, \tag{3.8.3}$$

where each f_i is a function variable of type $\mathbb{N}^{k_i} \to \mathbb{N}$ for some k_i and the y_j are individual variables.

(e) Using the pairing function of Exercise 1.20, show how to transform the formula (3.8.3) into an equivalent formula of the desired form (3.8.2).

3.34. Show that complete lattices are not a variety. (*Hint.* Use Theorem 3.42.)

3.35. Show that the languages $L_{\omega_1^{\mathrm{ck}}\omega}$ and $L_{\omega_1\omega}$ are not compact. (*Hint.* Consider the countable set

$$\{\bigvee_{n<\omega} p(f^n(a))\} \quad \cup \quad \{\neg p(f^n(a)) \mid n < \omega\}$$

of infinitary formulas.)

3.36. Show that the languages $L_{\omega_1^{\mathrm{ck}}\omega}$ and $L_{\omega_1\omega}$ do not satisfy the upward Löwenheim–Skolem theorem. (*Hint.* Consider the sentence

$$\forall x \bigvee_{n<\omega} x = f^n(a)$$

of $L_{\omega_1^{\mathrm{ck}}\omega}$.)

3.37. Prove the following infinitary tautologies using the deductive system of Section 3.6.

(a) $\bigvee_\alpha \varphi \leftrightarrow \varphi$ (infinitary idempotence)
(b) $\neg \bigwedge_\alpha \varphi_\alpha \leftrightarrow \bigvee_\alpha \neg\varphi_\alpha$ (infinitary De Morgan law)
(c) $\varphi \wedge \bigvee_\alpha \psi_\alpha \leftrightarrow \bigvee_\alpha(\varphi \wedge \psi_\alpha)$ (infinitary distributive law)

3.38. Complete the proof of Theorem 3.63.

3.39. Give an IND program (see Section 2.2) that accepts a given code of a formula of $L_{\omega_1^{\mathrm{ck}}\omega}$ iff the formula is provable. Conclude from Exercise 2.11 that deciding validity of $L_{\omega_1^{\mathrm{ck}}\omega}$ formulas is in Π^1_1.

3.40. Prove clauses (ii)–(viii) of Theorem 3.70.

3.41. Prove Theorem 3.74.

3.42. Prove that every propositional modal formula is satisfied in a tree model in which each state has only finitely many successors. (*Hint*. Start with the tree model of Theorem 3.74 satisfying the given formula φ at the root. Describe an inductive procedure to move down the tree, labeling certain states with subformulas of φ that need to be satisfied at that state in order to make φ true at the root. Make sure only finitely many successors of each state are labeled. Delete unlabeled states. Prove by induction that each state of the resulting tree model satisfies its label.)